



---

THE CLASSIC WORK  
EXTENDED AND REFINED

---

# The Art of Computer Programming

VOLUME 4B

Combinatorial Algorithms  
Part 2

---

DONALD E. KNUTH

---

Note to readers:  
Please ignore these  
sidenotes; they're just  
hints to myself for  
preparing the index,  
and they're often flaky!

KNUTH

# THE ART OF COMPUTER PROGRAMMING

VOLUME 4    PRE-FASCICLE 5A

## MATHEMATICAL PRELIMINARIES REDUX

DONALD E. KNUTH *Stanford University*

ADDISON-WESLEY



October 3, 2015

Internet  
Stanford GraphBase  
MMIX

Internet page <http://www-cs-faculty.stanford.edu/~knuth/taocp.html> contains current information about this book and related books.

See also <http://www-cs-faculty.stanford.edu/~knuth/sgb.html> for information about *The Stanford GraphBase*, including downloadable software for dealing with the graphs used in many of the examples in Chapter 7.

See also <http://www-cs-faculty.stanford.edu/~knuth/mmixware.html> for downloadable software to simulate the MMIX computer.

Copyright © 2015 by Addison–Wesley

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher, except that the official electronic file may be used to print single copies for personal (not commercial) use.

Zeroth printing (revision 21), 01 October 2015

October 3, 2015

## PREFACE

*We — or the Black Chamber — have a little agreement with [Knuth];  
he doesn't publish the real Volume 4 of The Art of Computer Programming,  
and they don't render him metabolically challenged.*

— CHARLES STROSS, *The Atrocity Archive* (2001)

THIS BOOKLET contains draft material that I'm circulating to experts in the field, in hopes that they can help remove its most egregious errors before too many other people see it. I am also, however, posting it on the Internet for courageous and/or random readers who don't mind the risk of reading a few pages that have not yet reached a very mature state. *Beware:* This material has not yet been proofread as thoroughly as the manuscripts of Volumes 1, 2, 3, and 4A were at the time of their first printings. And those carefully-checked volumes, alas, were subsequently found to contain thousands of mistakes.

Given this caveat, I hope that my errors this time will not be so numerous and/or obtrusive that you will be discouraged from reading the material carefully. I did try to make the text both interesting and authoritative, as far as it goes. But the field is vast; I cannot hope to have surrounded it enough to corral it completely. So I beg you to let me know about any deficiencies that you discover.

To put the material in context, this pre-fascicle contains an exposition of mathematical material (mostly about probability theory) that I plan to include at the beginning of Volume 4B. Its *raison d'être* is explained below, in an excerpt from the preface to that volume.

\* \* \*

Probability theory has made huge strides since I “completed” my college education in 1963; hence I'm basically self-taught with respect to these new-fangled ideas, and I fear that in many respects my knowledge lags behind that of today's students. I've tried my best to get the story right, yet I fear that in many respects I'm woefully ignorant.

For example, I urgently need your help with respect to some exercises that I made up as I was preparing this material. I certainly don't like to receive credit for things that have already been published by others, and most of these results are quite natural “fruits” that were just waiting to be “plucked.” Therefore please tell me if you know who deserves to be credited, with respect to the ideas found in exercises 6, 8, 9, 19, 32, 33, 38, 73, 88, or 96.

\* \* \*

Special thanks are due to Persi Diaconis, Omid Etesami, Svante Janson, Sheldon Ross, Ernst Schulte-Geers, and ... for their detailed comments on my early attempts at exposition, as well as to numerous other correspondents who have contributed crucial corrections.

Diaconis  
Etesami  
Janson  
Ross  
Schulte-Geers  
Knuth

\* \* \*

I happily offer a “finder’s fee” of \$2.56 for each error in this draft when it is first reported to me, whether that error be typographical, technical, or historical. The same reward holds for items that I forgot to put in the index. And valuable suggestions for improvements to the text are worth 32¢ each. (Furthermore, if you find a better solution to an exercise, I’ll actually do my best to give you immortal glory, by publishing your name in the eventual book:—)

Cross references to yet-unwritten material sometimes appear as ‘00’; this impossible value is a placeholder for the actual numbers to be supplied later.

Happy reading!

*Stanford, California*  
*21 October 2012*

D. E. K.

MPR

## **Part of the Preface to Volume 4B**

During the years that I've been preparing Volume 4, I've often run across basic techniques of probability theory that I would have put into Section 1.2 of Volume 1 if I'd been clairvoyant enough to anticipate them in the 1960s. Finally I realized that I ought to collect most of them together in one place, near the beginning of Volume 4B, because the story of these developments is too interesting to be broken up into little pieces scattered here and there.

Therefore this volume begins with a special section entitled "Mathematical Preliminaries Redux," and future sections use the abbreviation 'MPR' to refer to its equations and its exercises.

VICIOUS

*In books of this nature I can only suggest you keep it  
as simple as the subject will allow.*

— KODE VICIOUS (2012)

October 3, 2015

discrete probabilities  
 atomic events  
 probability space  
 shuffle  
 playing cards  
 event  
 random variable  
 independent  
 random variables  
*k*-wise independent  
 pairwise independent random variables  
 conditional probability

## MATHEMATICAL PRELIMINARIES REDUX

MANY PARTS of this book deal with *discrete probabilities*, namely with a finite or countably infinite set  $\Omega$  of atomic events  $\omega$ , each of which has a given probability  $\Pr(\omega)$ , where

$$0 \leq \Pr(\omega) \leq 1 \quad \text{and} \quad \sum_{\omega \in \Omega} \Pr(\omega) = 1. \quad (1)$$

This set  $\Omega$ , together with the function  $\Pr$ , is called a “probability space.” For example,  $\Omega$  might be the set of all ways to shuffle a pack of 52 playing cards, with  $\Pr(\omega) = 1/52!$  for every such arrangement.

An *event* is, intuitively, a proposition that can be either true or false with certain probability. It might, for instance, be the statement “the top card is an ace,” with probability  $1/13$ . Formally, an event  $A$  is a subset of  $\Omega$ , namely the set of all atomic events for which the corresponding proposition  $A$  is true; and

$$\Pr(A) = \sum_{\omega \in A} \Pr(\omega) = \sum_{\omega \in \Omega} \Pr(\omega)[\omega \in A]. \quad (2)$$

A *random variable* is a function that assigns a value to every atomic event. We typically use uppercase letters for random variables, and lowercase letters for the values that they might assume; thus, we might say that the probability of the event  $X = x$  is  $\Pr(X = x) = \sum_{\omega \in \Omega} \Pr(\omega)[X(\omega) = x]$ . In our playing card example, the top card  $T$  is a random variable, and we have  $\Pr(T = \mathbf{Q}\spadesuit) = 1/52$ . (Sometimes, as here, the lowercase-letter convention is ignored.)

The random variables  $X_1, \dots, X_k$  are said to be *independent* if

$$\Pr(X_1 = x_1 \text{ and } \dots \text{ and } X_k = x_k) = \Pr(X_1 = x_1) \dots \Pr(X_k = x_k) \quad (3)$$

for all  $(x_1, \dots, x_k)$ . For example, if  $F$  and  $S$  denote the face value and suit of the top card  $T$ , clearly  $F$  and  $S$  are independent. Hence in particular we have  $\Pr(T = \mathbf{Q}\spadesuit) = \Pr(F = \mathbf{Q}) \Pr(S = \spadesuit)$ . But  $T$  is *not* independent of the bottom card,  $B$ ; indeed, we have  $\Pr(T = t \text{ and } B = b) \neq 1/52^2$  for *any* cards  $t$  and  $b$ .

A system of  $n$  random variables is called *k*-wise independent if no  $k$  of its variables are dependent. With pairwise (2-wise) independence, for example, we could have variable  $X$  independent of  $Y$ , variable  $Y$  independent of  $Z$ , and variable  $Z$  independent of  $X$ ; yet all three variables needn’t be independent (see exercise 6). Similarly, *k*-wise independence does not imply  $(k + 1)$ -wise independence. But  $(k + 1)$ -wise independence does imply *k*-wise independence.

The *conditional probability* of an event  $A$ , given an event  $B$ , is

$$\Pr(A \mid B) = \frac{\Pr(A \cap B)}{\Pr(B)} = \frac{\Pr(A \text{ and } B)}{\Pr(B)}, \quad (4)$$



when  $\Pr(B) > 0$ , otherwise it's  $\Pr(A)$ . Imagine breaking the whole probability space  $\Omega$  into two parts,  $\Omega' = B$  and  $\Omega'' = \overline{B} = \Omega \setminus B$ , with  $\Pr(\Omega') = \Pr(B)$  and  $\Pr(\Omega'') = 1 - \Pr(B)$ . If we assign new probabilities to atomic events by the rules

$$\Pr'(\omega) = \Pr(\omega|\Omega') = \frac{\Pr(\omega)[\omega \in \Omega']}{\Pr(\Omega')}, \quad \Pr''(\omega) = \Pr(\omega|\Omega'') = \frac{\Pr(\omega)[\omega \in \Omega'']}{\Pr(\Omega'')},$$

we obtain new probability spaces  $\Omega'$  and  $\Omega''$ , allowing us to contemplate a world where  $B$  is always true and another world where  $B$  is always false. It's like taking two branches in a tree, each of which has its own logic. Conditional probability is important for the analysis of algorithms because algorithms often get into different states where different probabilities are relevant. Notice that we always have

$$\Pr(A) = \Pr(A|B) \cdot \Pr(B) + \Pr(A|\overline{B}) \cdot \Pr(\overline{B}). \quad (5)$$

The events  $A_1, \dots, A_k$  are said to be independent if the random variables  $[A_1], \dots, [A_k]$  are independent. (Bracket notation applies in the usual way to events-as-statements, not just to events-as-subsets:  $[A] = 1$  if  $A$  is true, otherwise  $[A] = 0$ .) Exercise 20 proves that this happens if and only if

$$\Pr\left(\bigcap_{j \in J} A_j\right) = \prod_{j \in J} \Pr(A_j), \quad \text{for all } J \subseteq \{1, \dots, k\}. \quad (6)$$

In particular, events  $A$  and  $B$  are independent if and only if  $\Pr(A|B) = \Pr(A)$ .

When the values of a random variable  $X$  are real numbers or complex numbers, we've defined its *expected value*  $\mathbb{E}X$  in Section 1.2.10: We said that

$$\mathbb{E}X = \sum_{\omega \in \Omega} X(\omega) \Pr(\omega) = \sum_x x \Pr(X = x), \quad (7)$$

provided that this definition makes sense when the sums are taken over infinitely many nonzero values. (The sum should be absolutely convergent.) A simple but extremely important case arises when  $A$  is any event, and when  $X = [A]$  is a binary random variable representing the truth of that event; then

$$\mathbb{E}[A] = \sum_{\omega \in \Omega} [A](\omega) \Pr(\omega) = \sum_{\omega \in \Omega} [\omega \in A] \Pr(\omega) = \sum_{\omega \in A} \Pr(\omega) = \Pr(A). \quad (8)$$

We've also noted that the expectation of a sum,  $\mathbb{E}(X_1 + \dots + X_k)$ , always equals the sum of the expectations,  $(\mathbb{E}X_1) + \dots + (\mathbb{E}X_k)$ , whether or not the random variables  $X_j$  are independent. Furthermore the expectation of a product,  $\mathbb{E}X_1 \dots X_k$ , is the product of the expectations,  $(\mathbb{E}X_1) \dots (\mathbb{E}X_k)$ , if those variables do happen to be independent. In Section 3.3.2 we defined the covariance,

$$\text{covar}(X, Y) = \mathbb{E}((X - \mathbb{E}X)(Y - \mathbb{E}Y)) = (\mathbb{E}XY) - (\mathbb{E}X)(\mathbb{E}Y), \quad (9)$$

which tends to measure the way  $X$  and  $Y$  depend on each other. The variance,  $\text{var}(X)$ , is  $\text{covar}(X, X)$ ; the middle formula in (9) shows why it is nonnegative whenever the random variable  $X$  takes on only real values.

All of these notions of expected value carry over to *conditional expectation*,

$$\mathbb{E}(X|A) = \sum_{\omega \in A} X(\omega) \frac{\Pr(\omega)}{\Pr(A)} = \sum_x x \frac{\Pr(X = x \text{ and } A)}{\Pr(A)}, \quad (10)$$

notation  $\overline{B}$   
 probability spaces  
 independent  
 Bracket notation  
 expected value  
 absolutely convergent  
 binary random variable  
 covariance  
 variance  
 conditional expectation+

conditioned on any event  $A$ , when we want to work in the probability space for which  $A$  is true. One of the most important formulas, analogous to (5), is

$$\begin{aligned} E X &= \sum_y E(X | Y = y) \Pr(Y = y) \\ &= \sum_y \sum_x x \Pr(X = x | Y = y) \Pr(Y = y). \end{aligned} \tag{11}$$

binary random variables  
probability estimates–

Furthermore there’s also another important kind of conditional expectation: When  $X$  and  $Y$  are random variables, it’s often helpful to write ‘ $E(X | Y)$ ’ for “the expectation of  $X$  given  $Y$ .” Using that notation, Eq. (11) becomes simply

$$E X = E(E(X | Y)). \tag{12}$$

This is a truly marvelous identity, great for hand-waving and for impressing outsiders—except that it can be confusing until you understand what it means.

In the first place, if  $Y$  is a Boolean variable, ‘ $E(X | Y)$ ’ might look as if it means ‘ $E(X | Y=1)$ ’, thus asserting that  $Y$  is true, just as ‘ $E(X | A)$ ’ asserts the truth of  $A$  in (10). No; that interpretation is wrong, quite wrong. Be warned.

In the second place, you might think of  $E(X | Y)$  as a function of  $Y$ . Well, yes; but the best way to understand  $E(X | Y)$  is to regard it as a *random variable*. That’s why we’re allowed to compute its expected value in (12).

All random variables are functions of the atomic events  $\omega$ . The value of  $E(X | Y)$  at  $\omega$  is the average of  $X(\omega')$  over all events  $\omega'$  such that  $Y(\omega') = Y(\omega)$ :

$$E(X | Y)(\omega) = \sum_{\omega' \in \Omega} X(\omega') \Pr(\omega') [Y(\omega') = Y(\omega)] / \Pr(Y = Y(\omega)). \tag{13}$$

Similarly,  $E(X | Y_1, \dots, Y_r)$  averages over events with  $Y_j(\omega') = Y_j(\omega)$  for  $1 \leq j \leq r$ .

For example, suppose  $X_1$  through  $X_n$  are binary random variables constrained by the condition that  $\nu(X_1 \dots X_n) = X_1 + \dots + X_n = m$ , where  $m$  and  $n$  are constants with  $0 \leq m \leq n$ ; all  $\binom{n}{m}$  such bit vectors  $X_1 \dots X_n$  are assumed to be equally likely. Clearly  $E X_1 = m/n$ . But what is  $E(X_2 | X_1)$ ? If  $X_1 = 0$ , the expectation of  $X_2$  is  $m/(n - 1)$ ; otherwise that expectation is  $(m - 1)/(n - 1)$ ; consequently  $E(X_2 | X_1) = (m - X_1)/(n - 1)$ . And what is  $E(X_k | X_1, \dots, X_{k-1})$ ? The answer is easy, once you get used to the notation: If  $\nu(X_1 \dots X_{k-1}) = r$ , then  $X_k \dots X_n$  is a random bit vector with  $\nu(X_k \dots X_n) = m - r$ ; hence the average value of  $X_k$  will be  $(m - r)/(n + 1 - k)$  in that case. We conclude that

$$E(X_k | X_1, \dots, X_{k-1}) = \frac{m - \nu(X_1 \dots X_{k-1})}{n + 1 - k}, \quad \text{for } 1 \leq k \leq n. \tag{14}$$

The random variables on both sides of these equations are the same.

**Inequalities.** In practice we often want to prove that certain events are rare, in the sense that they occur with very small probability. Conversely, our goal is sometimes to show that an event is *not* rare. And we’re in luck, because mathematicians have devised several fairly easy ways to derive upper bounds or lower bounds on probabilities, even when the exact values are unknown.

We've already discussed the most important technique of this kind in Section 1.2.10. Stated in highly general terms, the basic idea can be formulated as follows: *Let  $f$  be any nonnegative function such that  $f(x) \geq s > 0$  when  $x \in S$ . Then*

$$\Pr(X \in S) \leq E f(X)/s, \tag{15}$$

provided that  $\Pr(X \in S)$  and  $E f(X)$  both exist. For example,  $f(x) = |x|$  yields

$$\Pr(|X| \geq m) \leq E|X|/m \tag{16}$$

whenever  $m > 0$ . The proof is amazingly simple, because we obviously have

$$E f(X) \geq \Pr(X \in S) \cdot s + \Pr(X \notin S) \cdot 0. \tag{17}$$

Formula (15) is often called *Markov's inequality*, because A. A. Markov discussed the special case  $f(x) = |x|^a$  in *Izvēstīnā Imp. Akad. Nauk* (6) **1** (1907), 707–716. If we set  $f(x) = (x - EX)^2$ , we get the famous 19th-century inequality of Bienaymé and Chebyshev:

$$\Pr(|X - EX| \geq r) \leq \text{var}(X)/r^2. \tag{18}$$

The case  $f(x) = e^{ax}$  is also extremely useful.

Another fundamental estimate, known as *Jensen's inequality* [*Acta Mathematica* **30** (1906), 175–193], applies to *convex* functions  $f$ ; we've seen it so far only as a “hint” to exercise 6.2.2–36(!). The real-valued function  $f$  is said to be convex in an interval  $I$  of the real line, and  $-f$  is said to be concave in  $I$ , if

$$f(px + qy) \leq pf(x) + qf(y) \quad \text{for all } x, y \in I, \tag{19}$$

whenever  $p \geq 0, q \geq 0$ , and  $p+q = 1$ . This condition turns out to be equivalent to saying that  $f''(x) \geq 0$  for all  $x \in I$ , if  $f$  has a second derivative  $f''$ . For example, the functions  $e^{ax}$  and  $x^{2n}$  are convex for all constants  $a$  and all nonnegative integers  $n$ ; and if we restrict consideration to positive values of  $x$ , then  $f(x) = x^n$  is convex for *all* integers  $n$  (notably  $f(x) = 1/x$  when  $n = -1$ ). The functions  $\ln(1/x)$  and  $x \ln x$  are also convex for  $x > 0$ . Jensen's inequality states that

$$f(EX) \leq E(f(X)) \tag{20}$$

when  $f$  is convex in the interval  $I$  and the random variable  $X$  takes values only in  $I$ . (See exercise 42 for a proof.) For example, we have  $1/EX \leq E(1/X)$  and  $\ln EX \geq E \ln X$  and  $(EX) \ln EX \leq E(X \ln X)$ , when  $X$  is positive. Notice that (20) actually reduces to the very definition of convexity, (19), in the special case when  $X = x$  with probability  $p$  and  $X = y$  with probability  $q$ .

Third and fourth on our list of remarkably useful inequalities are two classical results that apply to any random variable  $X$  whose values are nonnegative integers:

$$\Pr(X > 0) \leq EX; \tag{21} \quad \text{("the first moment principle")}$$

$$\Pr(X > 0) \geq (EX)^2/(EX^2). \tag{22} \quad \text{("the second moment principle")}$$

Formula (21) is obvious, because the left side is  $p_1 + p_2 + p_3 + \dots$  when  $p_k$  is the probability that  $X = k$ , while the right side is  $p_1 + 2p_2 + 3p_3 + \dots$ .

tail inequalities  
 Markov's inequality  
 Bienaymé  
 Chebyshev  
 Jensen's inequality  
 convex  
 concave  
 first moment principle  
 second moment principle

Formula (22) isn't quite so obvious; it is  $p_1 + p_2 + p_3 + \dots$  on the left and  $(p_1 + 2p_2 + 3p_3 + \dots)^2 / (p_1 + 4p_2 + 9p_3 + \dots)$  on the right. However, as we saw with Markov's inequality, there is a remarkably simple proof, once we happen to discover it:

$$\begin{aligned} \mathbb{E} X^2 &= \mathbb{E}(X^2 | X > 0) \Pr(X > 0) + \mathbb{E}(X^2 | X = 0) \Pr(X = 0) \\ &= \mathbb{E}(X^2 | X > 0) \Pr(X > 0) \\ &\geq (\mathbb{E}(X | X > 0))^2 \Pr(X > 0) = (\mathbb{E} X)^2 / \Pr(X > 0). \end{aligned} \tag{23}$$

Markov's inequality  
binary  
Ross  
conditional expectation inequality  
reliability polynomial  
monotone Boolean function  
BDD  
prime implicants  
FKG inequality

In fact this proof shows that the second moment principle is valid even when  $X$  is not restricted to integer values (see exercise 46). Furthermore the argument can be strengthened to show that (22) holds even when  $X$  can take arbitrary *negative* values, provided only that  $\mathbb{E} X \geq 0$  (see exercise 47). See also exercise 118.

Exercise 54 applies (21) and (22) to the study of random graphs.

Another important inequality, which applies in the special case where  $X = X_1 + \dots + X_m$  is the sum of *binary* random variables  $X_j$ , was introduced more recently by S. M. Ross [*Probability, Statistics, and Optimization* (New York: Wiley, 1994), 185–190], who calls it the “conditional expectation inequality”:

$$\Pr(X > 0) \geq \sum_{j=1}^m \frac{\mathbb{E} X_j}{\mathbb{E}(X | X_j = 1)}. \tag{24}$$

Ross showed that the right-hand side of this inequality is always at least as big as the bound  $(\mathbb{E} X)^2 / (\mathbb{E} X^2)$  that we get from the second moment principle (see exercise 50). Furthermore, (24) is often easier to compute, even though it may look more complicated at first glance.

For example, his method applies nicely to the problem of estimating a reliability polynomial,  $f(p_1, \dots, p_n)$ , when  $f$  is a monotone Boolean function; here  $p_j$  represents the probability that component  $j$  of a system is “up.” We observed in Section 7.1.4 that reliability polynomials can be evaluated exactly, using BDD methods, when  $n$  is reasonably small; but approximations are necessary when  $f$  gets complicated. The simple example  $f(x_1, \dots, x_5) = x_1 x_2 x_3 \vee x_2 x_3 x_4 \vee x_4 x_5$  illustrates Ross's general method: Let  $(Y_1, \dots, Y_5)$  be independent binary random variables, with  $\mathbb{E} Y_j = p_j$ ; and let  $X = X_1 + X_2 + X_3$ , where  $X_1 = Y_1 Y_2 Y_3$ ,  $X_2 = Y_2 Y_3 Y_4$ , and  $X_3 = Y_4 Y_5$  correspond to the prime implicants of  $f$ . Then  $\Pr(X > 0) = \Pr(f(Y_1, \dots, Y_5) = 1) = \mathbb{E} f(Y_1, \dots, Y_5) = f(p_1, \dots, p_5)$ , because the  $Y$ 's are independent. And we can evaluate the bound in (24) easily:

$$\Pr(X > 0) \geq \frac{p_1 p_2 p_3}{1 + p_4 + p_4 p_5} + \frac{p_2 p_3 p_4}{p_1 + 1 + p_5} + \frac{p_4 p_5}{p_1 p_2 p_3 + p_2 p_3 + 1}. \tag{25}$$

If, for example, each  $p_j$  is 0.9, this formula gives  $\approx 0.848$ , while  $(\mathbb{E} X)^2 / (\mathbb{E} X^2) \approx 0.847$ ; the true value,  $p_1 p_2 p_3 + p_2 p_3 p_4 + p_4 p_5 - p_1 p_2 p_3 p_4 - p_2 p_3 p_4 p_5$ , is 0.9558.

Many other important inequalities relating to expected values have been discovered, of which the most significant for our purposes in this book is the *FKG inequality* discussed in exercise 61. It yields easy proofs that certain events are correlated, as illustrated in exercise 62.

**Martingales.** A sequence of dependent random variables can be difficult to analyze, but if those variables obey invariant constraints we can often exploit their structure. In particular, the “martingale” property, named after a classic betting strategy (see exercise 67), proves to be amazingly useful when it applies. Joseph L. Doob featured martingales in his pioneering book *Stochastic Processes* (New York: Wiley, 1953), and developed their extensive theory.

martingales-  
Doob  
Pólya  
urn model  
Eggenberger  
Pólya  
martingale with respect to the sequence

The sequence  $\langle Z_n \rangle = Z_0, Z_1, Z_2, \dots$  of real-valued random variables is called a *martingale* if it satisfies the condition

$$E(Z_{n+1} | Z_0, \dots, Z_n) = Z_n \quad \text{for all } n \geq 0. \tag{26}$$

(We also implicitly assume, as usual, that the expectations  $E Z_n$  are well defined.) For example, when  $n = 0$ , the random variable  $E(Z_1 | Z_0)$  must be the same as the random variable  $Z_0$  (see exercise 63).

Figure 1 illustrates George Pólya’s famous “urn model” [F. Eggenberger and G. Pólya, *Zeitschrift für angewandte Math. und Mech.* **3** (1923), 279–289], which is associated with a particularly interesting martingale. Imagine an urn that initially contains two balls, one red and one black. Repeatedly remove a randomly chosen ball from the urn, then replace it and contribute a new ball of the same color. The numbers  $(r, b)$  of red and black balls will follow a path in the diagram, with the respective local probabilities indicated on each branch.

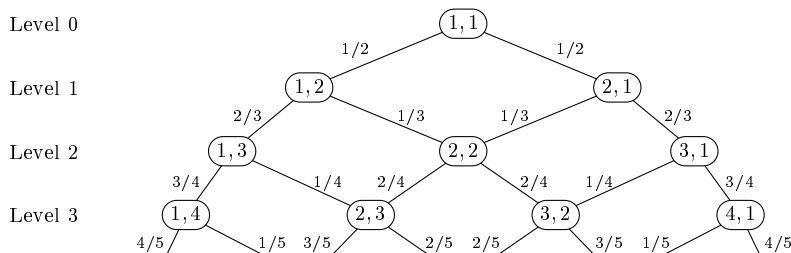
One can show without difficulty that all  $n+1$  nodes on level  $n$  of Fig. 1 will be reached with the same probability,  $1/(n+1)$ . Furthermore, the probability that a red ball is chosen when going from any level to the next is always  $1/2$ . Thus the urn scheme might seem at first glance to be rather tame and uniform. But in fact the process turns out to be full of surprises, because any inequity between red and black tends to perpetuate itself. For example, if the first ball chosen is black, so that we go from  $(1, 1)$  to  $(1, 2)$ , the probability is only  $2 \ln 2 - 1 \approx .386$  that the red balls will ever overtake the black ones in the future (see exercise 88).

One good way to analyze Pólya’s process is to use the fact that the ratios  $r/(r+b)$  form a martingale. Each visit to the urn changes this ratio either to  $(r+1)/(r+b+1)$  (with probability  $r/(r+b)$ ) or to  $r/(r+b+1)$  (with probability  $b/(r+b)$ ); so the expected new ratio is  $(rb+r^2+r)/((r+b)(r+b+1)) = r/(r+b)$ , no different from what it was before. More formally, let  $X_0 = 1$ , and for  $n > 0$  let  $X_n$  be the random variable ‘[the  $n$ th ball chosen is red]’. Then there are  $X_0 + \dots + X_n$  red balls and  $\bar{X}_0 + \dots + \bar{X}_n + 1$  black balls at level  $n$  of Fig. 1; and the sequence  $\langle Z_n \rangle$  is a martingale if we define

$$Z_n = (X_0 + \dots + X_n)/(n+2). \tag{27}$$

In practice it’s usually most convenient to define martingales  $Z_0, Z_1, \dots$  in terms of auxiliary random variables  $X_0, X_1, \dots$ , as we’ve just done. The sequence  $\langle Z_n \rangle$  is said to be a *martingale with respect to the sequence*  $\langle X_n \rangle$  if  $Z_n$  is a function of  $(X_0, \dots, X_n)$  that satisfies

$$E(Z_{n+1} | X_0, \dots, X_n) = Z_n \quad \text{for all } n \geq 0. \tag{28}$$



fair with respect to the sequence  
 martingale differences, see fair sequences  
 fair  
 independent  
 stopping rule

**Fig. 1.** Pólya's urn model. The probability of taking any downward path from  $(1, 1)$  to  $(r, b)$  is the product of the probabilities shown on the branches.

Furthermore we say that a sequence  $\langle Y_n \rangle$  is *fair with respect to the sequence*  $\langle X_n \rangle$  if  $Y_n$  is a function of  $(X_0, \dots, X_n)$  that satisfies the simpler condition

$$E(Y_{n+1} | X_0, \dots, X_n) = 0 \quad \text{for all } n \geq 0; \tag{29}$$

and we call  $\langle Y_n \rangle$  *fair* whenever

$$E(Y_{n+1} | Y_0, \dots, Y_n) = 0 \quad \text{for all } n \geq 0. \tag{30}$$

Exercise 77 proves that (28) implies (26) and that (29) implies (30); thus an auxiliary sequence  $\langle X_n \rangle$  is sufficient but not necessary for defining martingales and fair sequences.

Whenever  $\langle Z_n \rangle$  is a martingale, we obtain a fair sequence  $\langle Y_n \rangle$  by letting  $Y_0 = Z_0$  and  $Y_n = Z_n - Z_{n-1}$  for  $n > 0$ , because the identity  $E(Y_{n+1} | Z_0, \dots, Z_n) = E(Z_{n+1} - Z_n | Z_0, \dots, Z_n) = Z_n - Z_n$  shows that  $\langle Y_n \rangle$  is fair with respect to  $\langle Z_n \rangle$ . Conversely, whenever  $\langle Y_n \rangle$  is fair, we obtain a martingale  $\langle Z_n \rangle$  by letting  $Z_n = Y_0 + \dots + Y_n$ , because the identity  $E(Z_{n+1} | Y_0, \dots, Y_n) = E(Z_n + Y_{n+1} | Y_0, \dots, Y_n) = Z_n$  shows that  $\langle Z_n \rangle$  is a martingale with respect to  $\langle Y_n \rangle$ . In other words, fairness and martingaleness are essentially equivalent. The  $Y$ 's represent unbiased "tweaks" that change one  $Z$  to its successor.

It's easy to construct fair sequences. For example, every sequence of *independent* random variables with mean 0 is fair. And if  $\langle Y_n \rangle$  is fair with respect to  $\langle X_n \rangle$ , so is the sequence  $\langle Y'_n \rangle$  defined by  $Y'_n = f_n(X_0, \dots, X_{n-1})Y_n$  when  $f_n(X_0, \dots, X_{n-1})$  is almost *any* function whatsoever! (We need only keep  $f_n$  small enough that  $EY'_n$  is well defined.) In particular, we can let  $f_n(X_0, \dots, X_{n-1}) = 0$  for all large  $n$ , thereby making  $\langle Z_n \rangle$  eventually fixed.

A sequence of functions  $N_n(x_0, \dots, x_{n-1})$  is called a *stopping rule* if each value is either 0 or 1 and if  $N_n(x_0, \dots, x_{n-1}) = 0$  implies  $N_{n+1}(x_0, \dots, x_n) = 0$ . We can assume that  $N_0 = 1$ . The number of steps before stopping, with respect to a sequence of random variables  $\langle X_n \rangle$ , is then the random variable

$$N = N_1(X_0) + N_2(X_0, X_1) + N_3(X_0, X_1, X_2) + \dots \tag{31}$$

(Intuitively,  $N_n(x_0, \dots, x_{n-1})$  means [the values  $X_0 = x_0, \dots, X_{n-1} = x_{n-1}$  do *not* stop the process]; hence it's really more about "going" than "stopping.") Any martingale  $Z_n = Y_0 + \dots + Y_n$  with respect to  $\langle X_n \rangle$  can be adapted to

stop with this strategy if we change it to  $Z'_n = Y'_0 + \dots + Y'_n$ , where  $Y'_n = N_n(X_0, \dots, X_{n-1})Y_n$ . Gamblers who wish to “quit when ahead” are using the stopping rule  $N_{n+1}(X_0, \dots, X_n) = [Z'_n \leq 0]$ , when  $Z'_n$  is their current balance.

Notice that if the stopping rule always stops after at most  $m$  steps—in other words, if the function  $N_m(x_0, \dots, x_{m-1})$  is identically zero—then we have  $Z'_m = Z'_N$ , because  $Z'_n$  doesn't change after the process has stopped. Therefore  $E Z'_N = E Z'_m = E Z'_0 = E Z_0$ : *No stopping rule can change the expected outcome of a martingale when the number of steps is bounded.*

An amusing game of chance called Ace Now illustrates this optional stopping principle. Take a deck of cards, shuffle it and place the cards face down; then turn them face up one at a time as follows: Just before seeing the  $n$ th card, you are supposed to say either “Stop” or “Deal,” based on the cards you've already observed. (If  $n = 52$  you *must* say “Stop.”) After you've decided to stop, you win \$12 if the next card is an ace; otherwise you lose \$1. What is the best strategy for playing this game? Should you hold back until you have a pretty good chance at the \$12? What is the worst strategy? Exercise 82 has the answer.

Ace Now  
 optional stopping principle  
 playing cards  
 Tail inequalities  
 large deviations, see tail inequalities  
 submartingale  
 supermartingale  
 convex function  
 stopping rule  
 maximal inequality

**Tail inequalities from martingales.** The essence of martingales is *equality* of expectations. Yet martingales turn out to be important in the analysis of algorithms because we can use them to derive *inequalities*, namely to show that certain events occur with very small probability.

To begin our study, let's introduce inequality into Eq. (26): A sequence  $\langle Z_n \rangle$  is called a *submartingale* if it satisfies

$$E(Z_{n+1} | Z_0, \dots, Z_n) \geq Z_n \quad \text{for all } n \geq 0. \tag{32}$$

Similarly, it's called a *supermartingale* if ' $\geq$ ' is changed to ' $\leq$ ' in the left-hand part of this definition. (Thus a martingale is both sub- and super-.) In a submartingale we have  $E Z_0 \leq E Z_1 \leq E Z_2 \leq \dots$ , by taking expectations in (32). A supermartingale, similarly, has ever *smaller* expectations as  $n$  grows. One way to remember the difference between submartingales and supermartingales is to observe that their names are the reverse of what you might expect.

Submartingales are significant largely because of the fact that they're quite common. Indeed, if  $\langle Z_n \rangle$  is any martingale and if  $f$  is any convex function, then  $\langle f(Z_n) \rangle$  is a submartingale (see exercise 84). For example, the sequences  $\langle |Z_n| \rangle$  and  $\langle \max(Z_n, c) \rangle$  and  $\langle Z_n^2 \rangle$  and  $\langle e^{Z_n} \rangle$  all are submartingales whenever  $\langle Z_n \rangle$  is known to be a martingale. If, furthermore,  $Z_n$  is always positive, then  $\langle Z_n^3 \rangle$  and  $\langle 1/Z_n \rangle$  and  $\langle \ln(1/Z_n) \rangle$  and  $\langle Z_n \ln Z_n \rangle$ , etc., are submartingales.

If we modify a submartingale by applying a stopping rule, it's easy to see that we get another submartingale. Furthermore, if that stopping rule is guaranteed to quit within  $m$  steps, we'll have  $E Z_m \geq E Z_N = E Z'_N = E Z'_m$ . Therefore *no stopping rule can increase the expected outcome of a submartingale, when the number of steps is bounded.*

That comparatively simple observation has many important consequences. For example, exercise 86 uses it to give a simple proof of the so-called “maximal

inequality”: If  $\langle Z_n \rangle$  is a nonnegative submartingale then

$$\Pr(\max(Z_0, Z_1, \dots, Z_n) \geq x) \leq (E Z_n)/x, \quad \text{for all } x > 0. \quad (33)$$

Special cases of this inequality are legion. For instance, martingales  $\langle Z_n \rangle$  satisfy

$$\Pr(\max(|Z_0|, |Z_1|, \dots, |Z_n|) \geq x) \leq E(|Z_n|)/x, \quad \text{for all } x > 0; \quad (34)$$

$$\Pr(\max(Z_0^2, Z_1^2, \dots, Z_n^2) \geq x) \leq E(Z_n^2)/x, \quad \text{for all } x > 0. \quad (35)$$

Relation (35) is known as *Kolmogorov’s inequality*, because A. N. Kolmogorov proved it when  $Z_n = X_1 + \dots + X_n$  is the sum of independent random variables with  $E X_k = 0$  and  $\text{var}(X_k) = \sigma_k^2$  for  $1 \leq k \leq n$  [*Math. Annalen* **99** (1928), 309–311]. In that case  $\text{var}(Z_n) = \sigma_1^2 + \dots + \sigma_n^2 = \sigma^2$ , and the inequality can be written

$$\Pr(|X_1| < t\sigma, |X_1 + X_2| < t\sigma, \dots, |X_1 + \dots + X_n| < t\sigma) \geq 1 - 1/t^2. \quad (36)$$

Chebyshev’s inequality gives only  $\Pr(|X_1 + \dots + X_n| < t\sigma) \geq 1 - 1/t^2$ , which is a considerably weaker result.

Another important inequality applies in the common case where we have good bounds on the terms  $Y_1, \dots, Y_n$  that enter into the standard representation  $Z_n = Y_0 + Y_1 + \dots + Y_n$  of a martingale. This one is called the *Hoeffding–Azuma inequality*, after papers by W. Hoeffding [*J. Amer. Statistical Association* **58** (1963), 13–30] and K. Azuma [*Tôhoku Math. Journal* (2) **19** (1967), 357–367]. It reads as follows: *If  $\langle Y_n \rangle$  is any fair sequence with  $a_n \leq Y_n \leq b_n$ , then*

$$\Pr(Y_1 + \dots + Y_n \geq x) \leq e^{-2x^2/((b_1 - a_1)^2 + \dots + (b_n - a_n)^2)}. \quad (37)$$

The same bound applies to  $\Pr(Y_1 + \dots + Y_n \leq -x)$ , since  $-b_n \leq -Y_n \leq -a_n$ ; so

$$\Pr(|Y_1 + \dots + Y_n| \geq x) \leq 2e^{-2x^2/((b_1 - a_1)^2 + \dots + (b_n - a_n)^2)}. \quad (38)$$

Exercise 90 breaks the proof of this result into small steps. In fact, the proof even shows that  $a_n$  and  $b_n$  may be functions of  $\{Y_0, \dots, Y_{n-1}\}$ .

**Applications.** The Hoeffding–Azuma inequality is useful in the analysis of many algorithms because it applies to “Doob martingales,” a very general class of martingales that J. L. Doob featured as Example 1 in his *Stochastic Processes* (1953), page 92. (In fact, he had already considered them many years earlier, in *Trans. Amer. Math. Soc.* **47** (1940), 486.) Doob martingales arise from *any* sequence of random variables  $\langle X_n \rangle$ , independent or not, and from any *other* random variable  $Q$ : We simply define

$$Z_n = E(Q | X_0, \dots, X_n). \quad (39)$$

Then, as Doob pointed out, the resulting sequence is a martingale (see exercise 91). In our applications,  $Q$  is an aspect of some algorithm that we wish to study, and the variables  $X_0, X_1, \dots$  reflect the inputs to the algorithm. For example, in an algorithm that uses random bits, the  $X$ ’s are those bits.

Consider a hashing algorithm in which  $t$  objects are placed into  $m$  random lists, where the  $n$ th object goes into list  $X_n$ ; thus  $1 \leq X_n \leq m$  for  $1 \leq n \leq t$ , and we assume that each of the  $m^t$  possibilities is equally likely. Let  $Q(x_1, \dots, x_t)$  be

Kolmogorov’s inequality  
 independent  
 Chebyshev’s inequality  
 Hoeffding–Azuma inequality  
 Hoeffding  
 Azuma  
 Hoeffding–Azuma inequality  
 Doob martingales  
 Doob  
 analysis of algorithms  
 random bits  
 hashing



the number of lists that remain empty after the objects have been placed into lists  $x_1, \dots, x_t$ , and let  $Z_n = E(Q | X_1, \dots, X_n)$  be the associated Doob martingale. Then  $Z_0 = E(Q)$  is the *average* number of empty lists; and  $Z_t = Q(X_1, \dots, X_t)$  is the *actual* number, in any particular run of the algorithm.

fair sequence  
method of bounded differences  
Lipschitz condition  
McDiarmid

What fair sequence corresponds to this martingale? If  $1 \leq n \leq t$ , the random variable  $Y_n = Z_n - Z_{n-1}$  is  $f_n(X_1, \dots, X_n)$ , where  $f_n(x_1, \dots, x_n)$  is the average of

$$\Delta(x_1, \dots, x_t) = \sum_{x=1}^m \Pr(X_n = x) (Q(x_1, \dots, x_{n-1}, x, x_{n+1}, \dots, x_t) - Q(x_1, \dots, x_{n-1}, x, x_{n+1}, \dots, x_t)) \quad (40)$$

taken over all  $m^{t-n}$  values of  $(x_{n+1}, \dots, x_t)$ .

In our application the function  $Q(x_1, \dots, x_t)$  has the property that

$$|Q(x_1, \dots, x_{n-1}, x', x_{n+1}, \dots, x_t) - Q(x_1, \dots, x_{n-1}, x, x_{n+1}, \dots, x_t)| \leq 1 \quad (41)$$

for all  $x$  and  $x'$ , because a change to any one hash address always changes the number of empty lists by either 1, 0, or  $-1$ . Consequently, for any fixed setting of the variables  $(x_1, \dots, x_{n-1}, x_{n+1}, \dots, x_t)$ , we have

$$\max_{x_n} \Delta(x_1, \dots, x_t) \leq \min_{x_n} \Delta(x_1, \dots, x_t) + 1. \quad (42)$$

The Hoeffding–Azuma inequality (37) therefore allows us to conclude that

$$\Pr(Z_t - Z_0 \geq x) = \Pr(Y_1 + \dots + Y_t \geq x) \leq e^{-2x^2/t}. \quad (43)$$

Furthermore,  $Z_0$  in this example is  $m(m-1)^t/m^t$ , because exactly  $(m-1)^t$  of the  $m^t$  possible hash sequences leave any particular list empty. And the random variable  $Z_t$  is the actual number of empty lists when the algorithm is run. Hence we can, for example, set  $x = \sqrt{t \ln f(t)}$  in (43), thereby proving that

$$\Pr(Z_t \geq (m-1)^t/m^{t-1} + \sqrt{t \ln f(t)}) \leq 1/f(t)^2, \quad \text{whenever } f(t) > 1. \quad (44)$$

The same upper bound applies to  $\Pr(Z_t \leq (m-1)^t/m^{t-1} - \sqrt{t \ln f(t)})$ .

Notice that the inequality (41) was crucial in this analysis. Therefore the strategy we’ve used to prove (43) is often called the “method of bounded differences.” In general, a function  $Q(x_1, \dots, x_t)$  is said to satisfy a *Lipschitz condition* in coordinate  $n$  if we have

$$|Q(x_1, \dots, x_{n-1}, x, x_{n+1}, \dots, x_t) - Q(x_1, \dots, x_{n-1}, x', x_{n+1}, \dots, x_t)| \leq c_n \quad (45)$$

for all  $x$  and  $x'$ . (This terminology mimics a well-known but only slightly similar constraint that was introduced long ago into functional analysis by Rudolf Lipschitz [*Crelle* **63** (1864), 296–308].) Whenever condition (45) holds, for a function  $Q$  associated with a Doob martingale for *independent* random variables  $X_1, \dots, X_t$ , we can prove that  $\Pr(Y_1 + \dots + Y_t \geq x) \leq \exp(-2x^2/(c_1^2 + \dots + c_t^2))$ .

Let’s work out one more example, due to Colin McDiarmid [*London Math. Soc. Lecture Notes* **141** (1989), 148–188, §8(a)]: Again we consider independent integer-valued random variables  $X_1, \dots, X_t$  with  $1 \leq X_n \leq m$  for  $1 \leq n \leq t$ ;

but this time we allow each  $X_n$  to have a different probability distribution. Furthermore we define  $Q(x_1, \dots, x_t)$  to be the *minimum number of bins* into which objects of sizes  $x_1, \dots, x_t$  can be packed, where each bin has capacity  $m$ .

This bin-packing problem sounds a lot harder than the hashing problem that we just solved. Indeed, the task of evaluating  $Q(x_1, \dots, x_t)$  is well known to be NP-complete [see M. R. Garey and D. S. Johnson, *SICOMP* 4 (1975), 397–411]. Yet  $Q$  obviously satisfies the condition (45) with  $c_n = 1$  for  $1 \leq n \leq t$ . Therefore the method of bounded differences tells us that inequality (43) is true, in spite of the apparent difficulty of this problem!

The only difference between this bin-packing problem and the hashing problem is that we're clueless about the value of  $Z_0$ . Nobody knows how to compute  $\mathbb{E}Q(X_1, \dots, X_t)$ , except for very special distributions of the random variables. However — and this is the magic of martingales — we do know that, whatever the value is, the actual numbers  $Z_t$  will be tightly concentrated around that average.

If all the  $X$ 's have the same distribution, the values  $\beta_t = \mathbb{E}Q(X_1, \dots, X_t)$  satisfy  $\beta_{t+t'} \leq \beta_t + \beta_{t'}$ , because we could always pack the  $t$  and  $t'$  items separately. Therefore, by the subadditive law (see the answer to exercise 2.5–39),  $\beta_t/t$  approaches a limit  $\beta$  as  $t \rightarrow \infty$ . Still, however, random trials won't give us decent bounds on that limit, because we have no good way to compute the  $Q$  function.

*If only he could have enjoyed Martingale for its beauty and its peace  
without being chained to it by this band of responsibility and guilt!*

— P. D. JAMES, *Cover Her Face* (1962)

**Statements that are almost sure, or quite sure.** Probabilities that depend on an integer  $n$  often have the property that they approach 0 or 1 as  $n \rightarrow \infty$ , and special terminology simplifies the discussion of such phenomena. If, say,  $A_n$  is an event for which  $\lim_{n \rightarrow \infty} \Pr(A_n) = 1$ , it's convenient to express this fact in words by saying, “ $A_n$  occurs almost surely, when  $n$  is large.” (Indeed, we usually don't bother to state that  $n$  is large, if we already understand that  $n$  is approaching infinity in the context of the current discussion.)

For example, if we toss a fair coin  $n$  times, we'll find that the coin almost surely comes up heads more than  $.49n$  times, but fewer than  $.51n$  times.

Furthermore, we'll occasionally want to express this concept tersely in formulas, by writing just ‘a.s.’ instead of spelling out the words “almost surely.” For instance, the statement just made about  $n$  coin tosses can be formulated as

$$.49n < X_1 + \dots + X_n < .51n \text{ a.s.}, \tag{46}$$

if  $X_1, \dots, X_n$  are independent binary random variables, each with  $\mathbb{E}X_j = 1/2$ . In general a statement such as “ $A_n$  a.s.” means that  $\lim_{n \rightarrow \infty} \Pr(A_n) = 1$ ; or, equivalently, that  $\lim_{n \rightarrow \infty} \Pr(\overline{A}_n) = 0$ .

If  $A_n$  and  $B_n$  are both a.s., then the combined event  $C_n = A_n \cap B_n$  is also a.s., regardless of whether those events are independent. The reason is that  $\Pr(\overline{C}_n) = \Pr(\overline{A}_n \cup \overline{B}_n) \leq \Pr(\overline{A}_n) + \Pr(\overline{B}_n)$ , which approaches 0 as  $n \rightarrow \infty$ .

Thus, to prove (46) we need only show that  $X_1 + \dots + X_n > .49n$  a.s. and that  $X_1 + \dots + X_n < .51n$  a.s., or in other words that  $\Pr(X_1 + \dots + X_n \leq .49n)$

bin-packing problem  
NP-complete  
Garey  
Johnson  
subadditive law  
JAMES  
-martingales  
almost surely  
a.s.  
coin tosses

and  $\Pr(X_1 + \dots + X_n \geq .51n)$  both approach 0. Those probabilities are actually equal, by symmetry between heads and tails; so we need only show that  $p_n = \Pr(X_1 + \dots + X_n \leq .49n)$  approaches 0. And that's no sweat, because we know from exercise 1.2.10–21 that  $p_n \leq e^{-.0001n}$ .

In fact, we've proved more: We've shown that  $p_n$  is *superpolynomially small*, namely that

$$p_n = O(n^{-K}) \quad \text{for all fixed numbers } K. \tag{47}$$

When the probability of an event  $\bar{A}_n$  is superpolynomially small, we say that  $A_n$  holds "quite surely," and abbreviate that by 'q.s.'. In other words, we've proved

$$.49n < X_1 + \dots + X_n < .51n \quad \text{q.s.} \tag{48}$$

We've seen that the combination of any two a.s. events is a.s.; hence the combination of any finite number of a.s. events is also a.s. That's nice, but q.s. events are even nicer: *The combination of any polynomial number of q.s. events is also q.s.* For example, if  $n^4$  different people each toss  $n$  coins, it is quite sure that *every one of them*, without exception, will obtain between  $.49n$  and  $.51n$  heads!

(When making such asymptotic statements we ignore the inconvenient truth that our bound on the failure of the assertion,  $2n^4 e^{-.0001n}$  in this case, becomes negligible only when  $n$  is greater than 700,000 or so.)

superpolynomially small  
quite surely  
q.s.  
asymptotic  
Nontransitive dice  
dice  
Fibonacci dice  
phi  
Lake Wobegon dice  
Bingo  
NanoBingo

## EXERCISES

1. [M21] (*Nontransitive dice.*) Suppose three biased dice with the respective faces

$$A = \begin{array}{|c|c|c|} \hline \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet \\ \hline \end{array}, \quad B = \begin{array}{|c|c|c|} \hline \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet \\ \hline \end{array}, \quad C = \begin{array}{|c|c|c|} \hline \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet \\ \hline \end{array}$$

are rolled independently at random.

- Show that  $\Pr(A > B) = \Pr(B > C) = \Pr(C > A) = 5/9$ .
- Find dice with  $\Pr(A > B)$ ,  $\Pr(B > C)$ ,  $\Pr(C > A)$  all *greater* than  $5/9$ .
- If Fibonacci dice have  $F_m$  faces instead of just six, show that we could have

$$\Pr(A > B) = \Pr(B > C) = F_{m-1}/F_m \quad \text{and} \quad \Pr(C > A) = F_{m-1}/F_m \pm 1/F_m^2.$$

2. [M32] Prove that the previous exercise is asymptotically *optimum*, in the sense that  $\min(\Pr(A > B), \Pr(B > C), \Pr(C > A)) < 1/\phi$ , regardless of the number of faces.

3. [22] (*Lake Wobegon dice.*) Continuing the previous exercises, find three dice such that  $\Pr(A > \frac{1}{3}(A + B + C)) \geq \Pr(B > \frac{1}{3}(A + B + C)) \geq \Pr(C > \frac{1}{3}(A + B + C)) \geq 16/27$ . Each face of each die should be  $\square$  or  $\blacksquare$  or  $\blacklozenge$  or  $\blacktriangle$  or  $\blacktriangledown$  or  $\blacktriangleright$ .

4. [22] (*Nontransitive Bingo.*) Each player in the game of NanoBingo has a card containing four numbers from the set  $S = \{1, 2, 3, 4, 5, 6\}$ , arranged in two rows. An announcer calls out the elements of  $S$ , in random order; the first player whose card has a horizontal row with both numbers called shouts "Bingo!" and wins. (Or victory is

shared when there are multiple Bingos.) For example, consider the four cards

$$A = \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 5 \\ \hline \end{array}, \quad B = \begin{array}{|c|c|} \hline 2 & 3 \\ \hline 4 & 6 \\ \hline \end{array}, \quad C = \begin{array}{|c|c|} \hline 3 & 4 \\ \hline 1 & 5 \\ \hline \end{array}, \quad D = \begin{array}{|c|c|} \hline 1 & 4 \\ \hline 2 & 6 \\ \hline \end{array}.$$

If the announcer calls “6, 2, 5, 1” when  $A$  plays against  $B$ , then  $A$  wins; but the sequence “1, 3, 2” would yield a tie. One can show that  $\Pr(A \text{ beats } B) = \frac{336}{720}$ ,  $\Pr(B \text{ beats } A) = \frac{312}{720}$ , and  $\Pr(A \text{ and } B \text{ tie}) = \frac{72}{720}$ . Determine the probabilities of all possible outcomes when there are (a) two (b) three (c) four different players using those cards.

Cover  
games  
joint distribution  
pairwise independent  
 $k$ -wise independent  
Schulte-Geers  
parity  
sideways sum

- **5.** [HM22] (T. M. Cover, 1989.) Common wisdom asserts that longer games favor the stronger player, because they provide more evidence of the relative skills.

However, consider an  $n$ -round game in which Alice scores  $A_1 + \dots + A_n$  points while Bob scores  $B_1 + \dots + B_n$  points, where each of  $A_1, \dots, A_n$  are independent random variables representing Alice’s strength, and each of  $B_1, \dots, B_n$  independently represent Bob’s (and are independent of the  $A$ ’s). Suppose Alice wins with probability  $P_n$ .

- a) Show that it’s possible to have  $P_1 = .99$  but  $P_{1000} < .0001$ .
- b) Let  $m_k = 2^{k^3}$ ,  $n_k = 2^{k^2+k}$ , and  $q_k = 2^{-k^2}/D$ , where  $D = 2^{-0} + 2^{-1} + 2^{-4} + 2^{-9} + \dots \approx 1.56447$ . Suppose  $A$  and  $B$  are zero except that  $A = m_k$  with probability  $q_k$  when  $k \geq 0$  is even,  $B = m_k$  with probability  $q_k$  when  $k \geq 1$  is odd. What are  $\Pr(A > B)$ ,  $\Pr(A < B)$ , and  $\Pr(A = B)$ ?
- c) With the distributions in (b), prove that  $P_{n_k} \rightarrow [k \text{ even}]$  as  $k \rightarrow \infty$ .
- **6.** [M22] Consider  $n \geq 2$  random Boolean (or binary) variables  $X_1 \dots X_n$  with the following joint distribution: The vector  $x_1 \dots x_n$  occurs with probability  $1/(n-1)^2$  if  $x_1 + \dots + x_n = 2$ , with probability  $(n-2)/(2n-2)$  if  $x_1 + \dots + x_n = 0$ , and with probability 0 otherwise. Show that the variables are pairwise independent (that is,  $X_i$  is independent of  $X_j$  when  $i \neq j$ ); but they are not  $k$ -wise independent for  $k > 2$ .

Also find a joint distribution, depending only on  $\nu x = x_1 + \dots + x_n$ , that is  $k$ -wise independent for  $k = 2$  and  $k = 3$  but not  $k = 4$ .

- 7.** [M30] (Ernst Schulte-Geers, 2012.) Generalizing exercise 6, construct a  $\nu x$ -based distribution that has  $k$ -wise but not  $(k+1)$ -wise independence, given  $k \geq 1$ .
- **8.** [M20] Suppose the Boolean vector  $x_1 \dots x_n$  occurs with probability  $(2 + (-1)^{\nu x})/2^{n+1}$ , where  $\nu x = x_1 + \dots + x_n$ . For what  $k$  is this distribution  $k$ -wise independent?
- 9.** [M20] Find a distribution of Boolean vectors  $x_1 \dots x_n$  such that any two variables are dependent; yet if we know the value of any  $x_j$ , the remaining variables are  $(n-1)$ -wise independent. *Hint:* The answer is so simple, you might feel hornswoggled.
- **10.** [M21] Let  $Y_1, \dots, Y_m$  be independent and uniformly distributed elements of  $\{0, 1, \dots, p-1\}$ , where  $p$  is prime. Also let  $X_j = (j^m + Y_1 j^{m-1} + \dots + Y_m) \bmod p$ , for  $1 \leq j \leq n$ . For what  $k$  are the  $X$ ’s  $k$ -wise independent?

**11.** [M20] If  $X_1, \dots, X_{2n}$  are independent random variables with the same discrete distribution, and if  $\alpha$  is any real number whatsoever, prove that

$$\Pr\left(\left|\frac{X_1 + \dots + X_{2n}}{2n} - \alpha\right| \leq \left|\frac{X_1 + \dots + X_n}{n} - \alpha\right|\right) > \frac{1}{2}.$$

**12.** [18] Which of the following four statements are equivalent to the statement that  $\Pr(A|B) > \Pr(A)$ ? (i)  $\Pr(B|A) > \Pr(B)$ ; (ii)  $\Pr(A|B) > \Pr(A|\bar{B})$ ; (iii)  $\Pr(B|A) > \Pr(B|\bar{A})$ ; (iv)  $\Pr(\bar{A}|\bar{B}) > \Pr(\bar{A}|B)$ .

**13.** [15] True or false:  $\Pr(A|C) > \Pr(A)$  if  $\Pr(A|B) > \Pr(A)$  and  $\Pr(B|C) > \Pr(B)$ .

14. [10] (Thomas Bayes, 1763.) Prove the “chain rule” for conditional probability:  

$$\Pr(A_1 \cap \dots \cap A_n) = \Pr(A_1) \Pr(A_2 | A_1) \dots \Pr(A_n | A_1 \cap \dots \cap A_{n-1}).$$
15. [12] True or false:  $\Pr(A | B \cap C) \Pr(B | C) = \Pr(A \cap B | C)$ .
16. [M15] Under what circumstances is  $\Pr(A | B) = \Pr(A \cup C | B)$ ?
- 17. [15] Evaluate the conditional probability  $\Pr(T \text{ is an ace} | B = \spadesuit)$  in the playing card example of the text, where  $T$  and  $B$  denote the top and bottom cards.
18. [20] Let  $M$  and  $m$  be the maximum and minimum values of the random variable  $X$ . Prove that  $\text{var}(X) \leq (M - \text{E} X)(\text{E} X - m)$ .
- 19. [HM28] Let  $X$  be a random nonnegative integer, with  $\Pr(X = x) = 1/2^{x+1}$ , and suppose that  $X = (\dots X_2 X_1 X_0)_2$  and  $X + 1 = (\dots Y_2 Y_1 Y_0)_2$  in binary notation.
- What is  $\text{E} X_n$ ? *Hint:* Express this number in the binary number system.
  - Prove that the random variables  $\{X_0, X_1, \dots, X_{n-1}\}$  are independent.
  - Find the mean and variance of  $S = X_0 + X_1 + X_2 + \dots$ .
  - Find the mean and variance of  $R = X_0 \oplus X_1 \oplus X_2 \oplus \dots$ .
  - Let  $\pi = (11.p_0 p_1 p_2 \dots)_2$ . What is the probability that  $X_n = p_n$  for all  $n \geq 0$ ?
  - What is  $\text{E} Y_n$ ? Show that  $Y_0$  and  $Y_1$  are *not* independent.
  - Find the mean and variance of  $T = Y_0 + Y_1 + Y_2 + \dots$ .
20. [M18] Let  $X_1, \dots, X_k$  be binary random variables for which we know that  $\text{E}(\prod_{j \in J} X_j) = \prod_{j \in J} \text{E} X_j$  for all  $J \subseteq \{1, \dots, k\}$ . Prove that the  $X$ 's are independent.
21. [M20] Find a small-as-possible example of random variables  $X$  and  $Y$  that satisfy  $\text{covar}(X, Y) = 0$ , that is,  $\text{E} XY = (\text{E} X)(\text{E} Y)$ , although they aren't independent.
22. [M20] Use Eq. (8) to prove the “union inequality”

$$\Pr(A_1 \cup \dots \cup A_n) \leq \Pr(A_1) + \dots + \Pr(A_n).$$

- 23. [M21] If each  $X_k$  is an independent binary random variable with  $\text{E} X_k = p$ , the *cumulative binomial distribution*  $B_{m,n}(p)$  is the probability that  $X_1 + \dots + X_n \leq m$ . Thus it's easy to see that  $B_{m,n}(p) = \sum_{k=0}^m \binom{n}{k} p^k (1-p)^{n-k}$ .
- Show that  $B_{m,n}(p)$  is *also* equal to  $\sum_{k=0}^m \binom{n-m-1+k}{k} p^k (1-p)^{n-m}$ , for  $0 \leq m \leq n$ .  
*Hint:* Consider the random variables  $J_1, J_2, \dots$ , and  $T$  defined by the rule that  $X_j = 0$  if and only if  $j$  has one of the  $T$  values  $\{J_1, J_2, \dots, J_T\}$ , where  $1 \leq J_1 < J_2 < \dots < J_T \leq n$ . What is  $\Pr(T \geq r \text{ and } J_r = s)$ ?
- 24. [HM27] The cumulative binomial distribution also has many other properties.
- Prove that  $B_{m,n}(p) = (n-m) \binom{n}{m} \int_p^1 x^m (1-x)^{n-1-m} dx$ , for  $0 \leq m < n$ .
  - Use that formula to prove that  $B_{m,n}(m/n) > \frac{1}{2}$ , for  $0 \leq m < n/2$ . *Hint:* Show that  $\int_0^{m/n} x^m (1-x)^{n-1-m} dx < \int_{m/n}^1 x^m (1-x)^{n-1-m} dx$ .
  - Show furthermore that  $B_{m,n}(m/n) > \frac{1}{2}$  when  $n/2 \leq m \leq n$ . [Thus  $m$  is the *median* value of  $X_1 + \dots + X_n$ , when  $p = m/n$  and  $m$  is an integer.]
25. [M25] Suppose  $X_1, X_2, \dots$  are independent random binary variables, with means  $\text{E} X_k = p_k$ . Let  $\binom{n}{k}$  be the probability that  $X_1 + \dots + X_n = k$ ; thus  $\binom{n}{k} = p_n \binom{n-1}{k-1} + q_n \binom{n-1}{k} = [z^k] (q_1 + p_1 z) \dots (q_n + p_n z)$ , where  $q_k = 1 - p_k$ .
- Prove that  $\binom{n}{k} \geq \binom{n}{k+1}$ , if  $p_j \leq (k+1)/(n+1)$  for  $1 \leq j \leq n$ .
  - Furthermore  $\binom{n}{k} \leq \binom{n}{k} p^k q^{n-k}$ , if  $p_j \leq p \leq k/n$  for  $1 \leq j \leq n$ .
26. [M27] Continuing exercise 25, prove that  $\binom{n}{k}^2 \geq \binom{n}{k-1} \binom{n}{k+1} (1 + \frac{1}{k})(1 + \frac{1}{n-k})$  for  $0 < k < n$ . *Hint:* Consider  $r_{n,k} = \binom{n}{k} / \binom{n}{k}$ .

Bayes  
 chain rule  
 conditional probability  
 playing card  
 binary notation  
 pi  
 union inequality  
 binary random variable  
 cumulative binomial distribution  
 negative binomial distribution, cumulative  
 Beta distribution  
 incomplete beta function  
 median  
 multivariate Bernoulli distribution  
 binomial distribution

- 27.** [M22] Find an expression for the generalized cumulative binomial distribution  $\sum_{k=0}^m \binom{n}{k}$  that is analogous to the alternative formula in exercise 23.
- 28.** [HM28] (W. Hoeffding, 1956.) Let  $X = X_1 + \dots + X_n$  and  $p_1 + \dots + p_n = np$  in exercise 25, and suppose that  $Eg(X) = \sum_{k=0}^n g(k) \binom{n}{k} p^k (1-p)^{n-k}$  for some function  $g$ .
- Prove that  $Eg(X) \leq \sum_{k=0}^n g(k) \binom{n}{k} p^k (1-p)^{n-k}$  if  $g$  is convex in  $[0..n]$ .
  - If  $g$  isn't convex, show that the maximum of  $Eg(X)$ , over all choices of  $\{p_1, \dots, p_n\}$  with  $p_1 + \dots + p_n = np$  can always be attained by a set of probabilities for which at most three distinct values  $\{0, a, 1\}$  occur among the  $p_j$ .
  - Furthermore  $\sum_{k=0}^m \binom{n}{k} \leq B_{m,n}(p)$ , whenever  $p_1 + \dots + p_n = np \geq m + 1$ .
- 29.** [HM29] (S. M. Samuels, 1965.) Continuing exercise 28, prove that we have  $B_{m,n}(p) \geq ((1-p)(m+1)/((1-p)m+1))^{n-m}$  whenever  $np \leq m + 1$ .
- 30.** [HM34] Let  $X_1, \dots, X_n$  be independent random variables whose values are non-negative integers, where  $EX_k = 1$  for all  $k$ , and let  $p = \Pr(X_1 + \dots + X_n \leq n)$ .
- What is  $p$ , if each  $X_k$  takes only the values 0 and  $n + 1$ ?
  - Show that, in any set of distributions that minimize  $p$ , each  $X_k$  assumes only two integer values, 0 and  $m_k$ , where  $1 \leq m_k \leq n + 1$ .
  - Furthermore we have  $p > 1/e$ , if each  $X_k$  has the same two-valued distribution.
- **31.** [M20] Assume that  $A_1, \dots, A_n$  are random events such that, for every subset  $I \subseteq \{1, \dots, n\}$ , the probability  $\Pr(\bigcap_{i \in I} A_i)$  that all  $A_i$  for  $i \in I$  occur simultaneously is  $\pi_I$ ; here  $\pi_I$  is a number with  $0 \leq \pi_I \leq 1$ , and  $\pi_\emptyset = 1$ . Show that the probability of any combination of the events,  $\Pr(f([A_1], \dots, [A_n]))$  for any Boolean function  $f$ , can be found by expanding  $f$ 's multilinear reliability polynomial  $f([A_1], \dots, [A_n])$  and replacing each term  $\prod_{i \in I} [A_i]$  by  $\pi_I$ . For example, the reliability polynomial of  $x_1 \oplus x_2 \oplus x_3$  is  $x_1 + x_2 + x_3 - 2x_1x_2 - 2x_1x_3 - 2x_2x_3 + 4x_1x_2x_3$ ; hence  $\Pr([A_1] \oplus [A_2] \oplus [A_3]) = \pi_1 + \pi_2 + \pi_3 - 2\pi_{12} - 2\pi_{13} - 2\pi_{23} + 4\pi_{123}$ . (Here ' $\pi_{12}$ ' is short for  $\pi_{\{1,2\}}$ , etc.)
- 32.** [M21] Not all sets of numbers  $\pi_I$  in the preceding exercise can arise in an actual probability distribution. For example, if  $I \subseteq J$  we must have  $\pi_I \geq \pi_J$ . What is a necessary and sufficient condition for the  $2^n$  values of  $\pi_I$  to be legitimate?
- 33.** [M20] Suppose  $X$  and  $Y$  are binary random variables whose joint distribution is defined by the probability generating function  $G(w, z) = E(w^X z^Y) = pw + qz + rwz$ , where  $p, q, r > 0$  and  $p + q + r = 1$ . Use the definitions in the text to compute the probability generating function  $E(z^{E(X|Y)})$  for the conditional expectation  $E(X|Y)$ .
- 34.** [M17] Write out an algebraic proof of (12), using the definitions (7) and (13).
- **35.** [M22] True or false: (a)  $E(E(X|Y)|Y) = E(X|Y)$ ; (b)  $E(E(X|Y)|Z) = E(X|Z)$ .
- 36.** [M21] Simplify the formulas (a)  $E(f(X)|X)$ ; (b)  $E(f(Y)E(g(X)|Y))$ .
- **37.** [M20] Suppose  $X_1 \dots X_n$  is a random permutation of  $\{1, \dots, n\}$ , with every permutation occurring with probability  $1/n!$ . What is  $E(X_k | X_1, \dots, X_{k-1})$ ?
- 38.** [M26] Let  $X_1 \dots X_n$  be a random restricted growth string of length  $n$ , each with probability  $1/\varpi_n$  (see Section 7.2.1.5). What is  $E(X_k | X_1, \dots, X_{k-1})$ ?
- **39.** [HM21] A hen lays  $N$  eggs, where  $\Pr(N = n) = e^{-\mu} \mu^n / n!$  obeys the Poisson distribution. Each egg hatches with probability  $p$ , independent of all other eggs. Let  $K$  be the resulting number of chicks. Express (a)  $E(K|N)$ , (b)  $E K$ , and (c)  $E(N|K)$  in terms of  $N, K, \mu$ , and  $p$ .
- 40.** [M16] Suppose  $X$  is a random variable with  $X \leq M$ , and let  $m$  be any value with  $m < M$ . Show that  $\Pr(X > m) \geq (EX - m)/(M - m)$ .

generalized  
Hoeffding  
convex  
Samuels  
reliability polynomial  
integer multilinear representation of a Boolean function  
probability generating function  
generating function  
random permutation  
restricted growth string  
Poisson distribution  
chicks  
tail inequality

41. [HM21] Which of the following functions are convex in the set of all real numbers  $x$ ? (a)  $|x|^a$ , where  $a$  is a constant; (b)  $\sum_{k \geq n} x^k/k!$ , where  $n \geq 0$  is an integer; (c)  $e^{e^{|x|}}$ ; (d)  $f(x)[x \in I] + \infty[x \notin I]$ , where  $f$  is convex in the interval  $I$ .
42. [HM21] Prove Jensen's inequality (20).
- 43. [M18] Use (12) and (20) to strengthen (20): *If  $f$  is convex in  $I$  and if the random variable  $X$  takes values in  $I$ , then  $f(\mathbb{E} X) \leq \mathbb{E}(f(\mathbb{E}(X|Y))) \leq \mathbb{E}(f(X))$ .*
- 44. [M25] If  $f$  is convex on the real line and if  $\mathbb{E} X = 0$ , prove that  $\mathbb{E} f(aX) \leq \mathbb{E} f(bX)$  whenever  $0 \leq a \leq b$ .
45. [M18] Derive the first moment principle (21) from Markov's inequality (15).
46. [M15] Explain why  $\mathbb{E}(X^2 | X > 0) \geq (\mathbb{E}(X | X > 0))^2$  in (23).
47. [M15] If  $X$  is random and  $Y = \max(0, X)$ , show that  $\mathbb{E} Y \geq \mathbb{E} X$  and  $\mathbb{E} Y^2 \leq \mathbb{E} X^2$ .
- 48. [M20] Suppose  $X_1, \dots, X_n$  are independent random variables with  $\mathbb{E} X_k = 0$  and  $\mathbb{E} X_k^2 = \sigma_k^2$  for  $1 \leq k \leq n$ . Chebyshev's inequality tells us that  $\Pr(|X_1 + \dots + X_n| \geq a) \leq (\sigma_1^2 + \dots + \sigma_n^2)/a^2$ ; show that the second moment principle gives a somewhat better one-sided estimate,  $\Pr(X_1 + \dots + X_n \geq a) \leq (\sigma_1^2 + \dots + \sigma_n^2)/(a^2 + \sigma_1^2 + \dots + \sigma_n^2)$ , if  $a \geq 0$ .
49. [M20] If  $X$  is random and  $\geq 0$ , prove that  $\Pr(X = 0) \leq (\mathbb{E} X^2)/(\mathbb{E} X)^2 - 1$ .
- 50. [M27] Let  $X = X_1 + \dots + X_m$  be the sum of binary random variables, with  $\mathbb{E} X_j = p_j$ . Let  $J$  be independent of the  $X$ 's, and uniformly distributed in  $\{1, \dots, m\}$ .
- Prove that  $\Pr(X > 0) = \sum_{j=1}^m \mathbb{E}(X_j/X | X_j > 0) \cdot \Pr(X_j > 0)$ .
  - Therefore (24) holds. *Hint:* Use Jensen's inequality with  $f(x) = 1/x$ .
  - What are  $\Pr(X_J = 1)$  and  $\Pr(J = j | X_J = 1)$ ?
  - Let  $t_j = \mathbb{E}(X | J = j \text{ and } X_J = 1)$ . Prove that  $\mathbb{E} X^2 = \sum_{j=1}^m p_j t_j$ .
  - Jensen's inequality now implies that the right side of (24) is  $\geq (\mathbb{E} X)^2/(\mathbb{E} X^2)$ .
- 51. [M21] Show how to use the conditional expectation inequality (24) to obtain also an *upper* bound on the value of a reliability polynomial, and apply your method to the case illustrated in (25).
52. [M21] What lower bound does inequality (24) give for the reliability polynomial of the symmetric function  $S_{\geq k}(x_1, \dots, x_n)$ , when  $p_1 = \dots = p_n = p$ ?
53. [M20] Use (24) to obtain a lower bound for the reliability polynomial of the *non-monotonic* Boolean function  $f(x_1, \dots, x_6) = x_1 x_2 \bar{x}_3 \vee x_2 x_3 \bar{x}_4 \vee \dots \vee x_5 x_6 \bar{x}_1 \vee x_6 x_1 \bar{x}_2$ .
- 54. [M22] Suppose each edge of a random graph on the vertices  $\{1, \dots, n\}$  is present with probability  $p$ , independent of every other edge. If  $u, v, w$  are distinct vertices, let  $X_{uvw}$  be the probability that  $\{u, v, w\}$  is a 3-clique, namely the probability that  $u - v$ ,  $u - w$ , and  $v - w$ . Also let  $X = \sum_{1 \leq u < v < w \leq n} X_{uvw}$  be the total number of 3-cliques. Use the (a) first and (b) second moment principle to derive bounds on the probability that the graph contains at least one 3-clique.
55. [23] Evaluate the upper and lower bounds in the previous exercise numerically in the case  $n = 10$ , and compare them to the true probability, when (a)  $p = 1/2$ ; (b)  $p = 1/10$ .
56. [HM20] Evaluate the upper and lower bounds of exercise 54 asymptotically when  $p = \lambda/n$  and  $n \rightarrow \infty$ .
- 57. [M21] Obtain a lower bound for the probability in exercise 54(b) by using the conditional expectation inequality (24) instead of the second moment principle (22).

convex  
 Jensen's inequality  
 first moment principle  
 Markov's inequality  
 Chebyshev's inequality  
 second moment principle  
 one-sided estimate  
 Jensen's inequality  
 conditional expectation inequality  
 reliability polynomial  
 Symmetric Boolean function  
 $S_{\geq m}$   
 random graph  
 clique  
 triangles (3-cliques)  
 second moment principle  
 first moment principle  
 asymptotically  
 conditional expectation inequality  
 second moment principle

58. [M22] Generalizing exercise 54, find bounds on the probability that a random graph on  $n$  vertices has a  $k$ -clique, when each edge has probability  $p$ .

► 59. [HM25] (*The four functions theorem.*) The purpose of this exercise is to prove an inequality that applies to four sequences  $\langle a_n \rangle, \langle b_n \rangle, \langle c_n \rangle, \langle d_n \rangle$  of nonnegative numbers:

$$a_j b_k \leq c_{j|k} d_{j&k} \quad \text{for } 0 \leq j, k < \infty \quad \text{implies} \quad \sum_{j=0}^{\infty} \sum_{k=0}^{\infty} a_j b_k \leq \sum_{j=0}^{\infty} \sum_{k=0}^{\infty} c_j d_k. \quad (*)$$

(The sums will be  $\infty$  if they don't converge.) Although the inequality might appear at first to be merely a curiosity, of interest only to a few lovers of esoteric formulas, we shall see that it's a fundamental result with many applications of great importance.

a) Prove the special case where  $a_j = b_j = c_j = d_j = 0$  for  $j \geq 2$ , namely that

$$a_0 b_0 \leq c_0 d_0, \quad a_0 b_1 \leq c_1 d_0, \quad a_1 b_0 \leq c_1 d_0, \quad \text{and} \quad a_1 b_1 \leq c_1 d_1$$

$$\text{implies} \quad (a_0 + a_1)(b_0 + b_1) \leq (c_0 + c_1)(d_0 + d_1).$$

Can equality hold in the first four relations but not in the last one? Can equality hold in the last relation but not in the first four?

b) Use that result to prove (\*) when  $a_j = b_j = c_j = d_j = 0$  for all  $j \geq 2^n$ , given  $n > 0$ .  
 c) Conclude that (\*) is true in general.

► 60. [M21] If  $\mathcal{F}$  is a family of sets, and if  $\alpha$  is a function that maps sets into real numbers, let  $\alpha(\mathcal{F}) = \sum_{S \in \mathcal{F}} \alpha(S)$ . Suppose  $\mathcal{F}$  and  $\mathcal{G}$  are finite families of sets for which nonnegative set functions  $\alpha, \beta, \gamma$ , and  $\delta$  have been defined with the property that

$$\alpha(S) \beta(T) \leq \gamma(S \cup T) \delta(S \cap T) \quad \text{for all } S \in \mathcal{F} \text{ and } T \in \mathcal{G}.$$

a) Use exercise 59 to prove that  $\alpha(\mathcal{F}) \beta(\mathcal{G}) \leq \gamma(\mathcal{F} \sqcup \mathcal{G}) \delta(\mathcal{F} \cap \mathcal{G})$ .  
 b) In particular,  $|\mathcal{F}| |\mathcal{G}| \leq |\mathcal{F} \sqcup \mathcal{G}| |\mathcal{F} \cap \mathcal{G}|$  for all families  $\mathcal{F}$  and  $\mathcal{G}$ .

► 61. [M28] Consider random sets in which  $S$  occurs with probability  $\mu(S)$ , where

$$\mu(S) \geq 0 \quad \text{and} \quad \mu(S) \mu(T) \leq \mu(S \cup T) \mu(S \cap T) \quad \text{for all sets } S \text{ and } T. \quad (**)$$

Assume also that  $U = \bigcup_{\mu(S) > 0} S$  is a finite set.

a) Prove the *FKG inequality* (which is named for C. M. Fortuin, P. W. Kasteleyn, and J. Ginibre): If  $f$  and  $g$  are real-valued set functions, then

$$f(S) \leq f(T) \text{ and } g(S) \leq g(T) \text{ for all } S \subseteq T \quad \text{implies} \quad E(fg) \geq E(f) E(g).$$

Here, as usual,  $E(f)$  stands for  $\sum_S \mu(S) f(S)$ . The conclusion can also be written 'covar( $f, g$ )  $\geq 0$ ', using the notation of (9); we say that  $f$  and  $g$  are "positively correlated" when this is true. (The awkward term "nonnegatively correlated" would be more accurate, because  $f$  and  $g$  might actually be independent.) *Hint:* Prove the result first in the special case that both  $f$  and  $g$  are nonnegative.

b) Furthermore,

$$f(S) \geq f(T) \text{ and } g(S) \geq g(T) \text{ for all } S \subseteq T \quad \text{implies} \quad E(fg) \geq E(f) E(g);$$

$$f(S) \leq f(T) \text{ and } g(S) \geq g(T) \text{ for all } S \subseteq T \quad \text{implies} \quad E(fg) \leq E(f) E(g).$$

c) It isn't necessary to verify condition (\*\*) for all sets, if (\*\*) is known to hold for sufficiently many pairs of "neighboring" sets. Given  $\mu$ , let's say that set  $S$  is *supported* if  $\mu(S) \neq 0$ . Prove that (\*\*) holds for all  $S$  and  $T$  whenever the following three conditions are satisfied: (i) If  $S$  and  $T$  are supported, so are  $S \cup T$  and  $S \cap T$ .

$k$ -clique  
 four functions theorem  
 family of sets  
 multivariate total positivity, see FKG inequ  
 Fortuin  
 Kasteleyn  
 Ginibre  
 positively correlated  
 correlation inequalities  
 covariance  
 nonnegatively correlated  
 supported



- (ii) If  $S$  and  $T$  are supported and  $S \subseteq T$ , the elements of  $T \setminus S$  can be labeled  $t_1, \dots, t_k$  such that each of the intermediate sets  $S \cup \{t_1, \dots, t_j\}$  is supported, for  $1 \leq j \leq k$ . (iii) Condition  $(**)$  holds whenever  $S = R \cup s$  and  $T = R \cup t$  and  $s, t \notin R$ .
- d) The *multivariate Bernoulli distribution*  $B(p_1, \dots, p_m)$  on subsets of  $\{1, \dots, m\}$  is

$$\mu(S) = \left( \prod_{j=1}^m p_j^{[j \in S]} \right) \left( \prod_{j=1}^m (1 - p_j)^{[j \notin S]} \right),$$

given  $0 \leq p_1, \dots, p_m \leq 1$ . (Thus each element  $j$  is included independently with probability  $p_j$ , as in exercise 25.) Show that this distribution satisfies  $(**)$ .

- e) Describe other simple distributions for which  $(**)$  holds.
- **62.** [M20] Suppose the  $m = \binom{n}{2}$  edges  $E$  of a random graph  $G$  on  $n$  vertices are chosen with the Bernoulli distribution  $B(p_1, \dots, p_m)$ . Let  $f(E) = [G \text{ is connected}]$  and  $g(E) = [G \text{ is 4-colorable}]$ . Prove that  $f$  is negatively correlated with  $g$ .
- 63.** [M17] Suppose  $Z_0$  and  $Z_1$  are random ternary variables with  $\Pr(Z_0 = a \text{ and } Z_1 = b) = p_{ab}$  for  $0 \leq a, b \leq 2$ , where  $p_{00} + p_{01} + \dots + p_{22} = 1$ . What can you say about those probabilities  $p_{ab}$  when  $E(Z_1 | Z_0) = Z_0$ ?
- **64.** [M22] (a) If  $E(Z_{n+1} | Z_n) = Z_n$  for all  $n \geq 0$ , is  $\langle Z_n \rangle$  a martingale? (b) If  $\langle Z_n \rangle$  is a martingale, is  $E(Z_{n+1} | Z_n) = Z_n$  for all  $n \geq 0$ ?
- 65.** [M21] If  $\langle Z_n \rangle$  is any martingale, show that any subsequence  $\langle Z_{m(n)} \rangle$  is also a martingale, where the nonnegative integers  $\langle m(n) \rangle$  satisfy  $m(0) < m(1) < m(2) < \dots$ .
- **66.** [M22] Find all martingales  $Z_0, Z_1, \dots$  such that each random variable  $Z_n$  assumes only the values  $\pm n$ .
- 67.** [M20] The Equitable Bank of El Dorado features a money machine such that, if you insert  $k$  dollars, you receive  $2k$  dollars back with probability exactly  $1/2$ ; otherwise you get nothing. Thus you either gain  $\$k$  or lose  $\$k$ , and your expected profit is  $\$0$ . (Of course these transactions are all done electronically.)
- Consider, however, the following scheme: Insert  $\$1$ ; if that loses, insert  $\$2$ ; if that also loses, insert  $\$4$ ; then  $\$8$ , etc. If you first succeed after inserting  $2^n$  dollars, stop (and take the  $2^{n+1}$  dollars). What's your expected net profit at the end?
  - Continuing (a), what's the expected total amount that you put into the machine?
  - If  $Z_n$  is your net profit after  $n$  trials, show that  $\langle Z_n \rangle$  is a martingale.
- 68.** [HM23] When J. H. Quick (a student) visited El Dorado, he decided to proceed by making repeated bets of  $\$1$  each, and to stop when he first came out ahead. (He was in no hurry, and was well aware of the perils of the high-stakes strategy in exercise 67.)
- What martingale  $\langle Z_n \rangle$  corresponds to this more conservative strategy?
  - Let  $N$  be the number of bets that Quick made before stopping. What is the probability that  $N = n$ ?
  - What is the probability that  $N \geq n$ ?
  - What is  $E N$ ?
  - What is the probability that  $\min(Z_0, Z_1, \dots) = -m$ ? (Possible "gambler's ruin.")
  - What is the expected number of indices  $n$  such that  $Z_n = -m$ , given  $m \geq 0$ ?
- 69.** [M20] Section 1.2.5 discusses two basic ways by which we can go from permutations of  $\{1, \dots, n-1\}$  to permutations of  $\{1, \dots, n\}$ : "Method 1" inserts  $n$  among the previous elements in all possible ways; "Method 2" puts a number  $k$  from 1 to  $n$  in the final position, and adds 1 to each previous number that was  $\geq k$ .

Bernoulli, James, distribution  
random graph  
negatively correlated  
subsequence  
Quick

Show that, using either method, every permutation can be associated with a node of Fig. 1, using a rule that obeys the probability assumptions of Pólya's urn model.

Pólya's urn model  
 Pólya  
 Friedman's urn  
 multiplicatively fair  
 De Moivre's martingale  
 coin tosses  
 fair sequence  
 Fibonacci martingale  
 Ace Now

70. [M25] If Pólya's urn model is generalized so that we start with  $c$  balls of *different* colors, is there a martingale that generalizes Fig. 1?

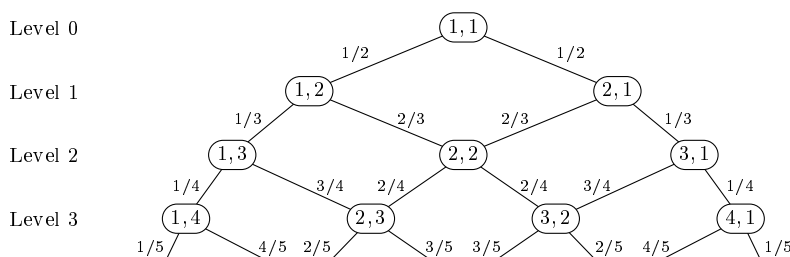
71. [M21] (G. Pólya.) What is the probability of going from node  $(r, b)$  to node  $(r', b')$  in Fig. 1, given  $r, r', b,$  and  $b'$  with  $r' \geq r$  and  $b' \geq b$ ?

72. [M21] Let  $X_n$  be the red-ball indicator for Pólya's urn, as discussed in the text. What is  $E(X_{n_1} X_{n_2} \dots X_{n_m})$  when  $0 < n_1 < n_2 < \dots < n_m$ ?

73. [M24] The ratio  $Z_n = r/(n+2)$  at node  $(r, n+2-r)$  of Fig. 1 is not the only martingale definable on Pólya's urn. For example,  $r[n=r-1]$  is another; so is  $r \binom{n+1}{r} / 2^n$ .

Find the most general martingale  $\langle Z_n \rangle$  for this model: Given any sequence  $a_0, a_1, \dots$ , show that there's exactly one suitable function  $Z_n = f(r, n)$  such that  $f(1, k) = a_k$ .

74. [M20] (*Bernard Friedman's urn.*) Instead of contributing a ball of the same color, as in Fig. 1, suppose we use the *opposite* color. Then the process changes to



and the probabilities of reaching each node become quite different. What are they?

75. [M25] Find an interesting martingale for Bernard Friedman's urn.

76. [M20] If  $\langle Z_n \rangle$  and  $\langle Z'_n \rangle$  are martingales, is  $\langle Z_n + Z'_n \rangle$  a martingale?

77. [M21] Prove or disprove: If  $\langle Z_n \rangle$  is a martingale with respect to  $\langle X_n \rangle$ , then  $\langle Z_n \rangle$  is a martingale with respect to itself (that is, a martingale).

78. [M20] A sequence of random variables  $\langle V_n \rangle$  for which  $E(V_{n+1} | V_0, \dots, V_n) = 1$  is called "multiplicatively fair." Show that  $Z_n = V_0 V_1 \dots V_n$  is a martingale in such a case. Conversely, does every martingale lead to a multiplicatively fair sequence?

79. [M20] (*De Moivre's martingale.*) Let  $X_1, X_2, \dots$  be a sequence of independent coin tosses, with  $\Pr(\text{"heads" occurred on the } n\text{th toss}) = \Pr(X_n = 1) = p$  for each  $n$ . Show that  $Z_n = (q/p)^{2(X_1 + \dots + X_n) - n}$  defines a martingale, where  $q = 1 - p$ .

80. [M20] Are the following statements true or false for every fair sequence  $\langle Y_n \rangle$ ? (a)  $E(Y_3^2 Y_5) = 0$ . (b)  $E(Y_3 Y_5^2) = 0$ . (c)  $E(Y_{n_1} Y_{n_2} \dots Y_{n_m}) = 0$  if  $n_1 < n_2 < \dots < n_m$ .

81. [M21] Suppose  $E(X_{n+1} | X_0, \dots, X_n) = X_n + X_{n-1}$  for  $n \geq 0$ , where  $X_{-1} = 0$ . Find sequences  $a_n$  and  $b_n$  of coefficients so that  $Z_n = a_n X_n + b_n X_{n-1}$  is a martingale, where  $Z_0 = X_0$  and  $Z_1 = 2X_0 - X_1$ . (We might call this a "Fibonacci martingale.")

- 82. [M20] In the game of Ace Now, let  $X_n = [\text{the } n\text{th card is an ace}]$ , with  $X_0 = 0$ .
  - a) Show that  $Z_n = (4 - X_1 - \dots - X_n)/(52 - n)$  satisfies (28) for  $0 \leq n < 52$ .
  - b) Consequently  $E Z_N = 1/13$ , regardless of the stopping rule employed.
  - c) Hence all strategies are equally good (or bad); you win \$0 on average.

- ▶ **83.** [HM22] Given a sequence  $\langle X_n \rangle$  of independent and nonnegative random variables, let  $S_n = X_1 + \dots + X_n$ . If  $N_n(x_0, \dots, x_{n-1})$  is any stopping rule and if  $N$  is defined by (31), prove that  $E S_N = E \sum_{k=1}^N E X_k$ . (In particular, if  $E X_n = E X_1$  for all  $n > 0$  we have “Wald’s equation,” which states that  $E S_N = (E N)(E X_1)$ .)
- 84.** [HM21] Let  $f(x)$  be a convex function for  $a \leq x \leq b$ , and assume that  $\langle Z_n \rangle$  is a martingale such that  $a \leq Z_n \leq b$  for all  $n \geq 0$ . (Possibly  $a = -\infty$  and/or  $b = +\infty$ .)
  - a) Prove that  $\langle f(Z_n) \rangle$  is a submartingale.
  - b) What can you say if the sequence  $\langle Z_n \rangle$  is assumed only to be a submartingale?
- 85.** [M20] Suppose there are  $R_n$  red balls and  $B_n$  black balls at level  $n$  of Pólya’s urn (Fig. 1). Prove that the sequence  $\langle R_n/B_n \rangle$  is a submartingale.
- ▶ **86.** [M22] Prove (33) by inventing a suitable stopping rule  $N_{n+1}(Z_0, \dots, Z_n)$ .
- 87.** [M17] What does the maximal inequality (33) reveal about the chances that Pólya’s urn will hold thrice as many red balls as black balls at some point?
- ▶ **88.** [HM30] Let  $S = \sup Z_n$  be the least upper bound of  $Z_n$  as  $n \rightarrow \infty$  in Fig. 1.
  - a) Prove that  $S > 1/2$  with probability  $\ln 2 \approx .693$ .
  - b) Similarly, show that  $\Pr(S > 2/3) = \ln 3 - \pi/\sqrt{27} \approx .494$ .
  - c) Generalize to  $\Pr(S > (t-1)/t)$ , for all  $t \geq 2$ . *Hint:* See exercise 7.2.1.6–36.
- 89.** [M16] Let  $(X_1, \dots, X_n)$  be random variables that have the Bernoulli distribution  $B(p_1, \dots, p_n)$ . Use (37) to show that  $\Pr(X_1 + \dots + X_n \geq p_1 + \dots + p_n + x) \leq e^{-2x^2/n}$ .
- 90.** [HM25] The Hoeffding–Azuma inequality (37) can be derived as follows:
  - a) Show first that  $\Pr(Y_1 + \dots + Y_n \geq x) \leq E(e^{(Y_1 + \dots + Y_n)t})/e^{tx}$  for all  $t > 0$ .
  - b) If  $0 \leq p \leq 1$  and  $q = 1 - p$ , show that  $e^{yt} \leq e^{f(t)} + ye^{g(t)}$  when  $-p \leq y \leq q$  and  $t > 0$ , where  $f(t) = -pt + \ln(q + pe^t)$  and  $g(t) = -pt + \ln(e^t - 1)$ .
  - c) Prove that  $f(t) \leq t^2/8$ . *Hint:* Use Taylor’s formula, Eq. 1.2.11.3–(5).
  - d) Consequently  $a \leq Y \leq b$  implies  $e^{Yt} \leq e^{(b-a)^2 t^2/8} + Yh(t)$ , for some function  $h(t)$ .
  - e) Let  $c = (c_1^2 + \dots + c_n^2)/2$ , where  $c_k = b_k - a_k$ . Prove that  $E(e^{(Y_1 + \dots + Y_n)t}) \leq e^{ct^2/4}$ .
  - f) We obtain (37) by choosing the best value of  $t$ .
- 91.** [M20] Prove that Doob’s general formula (39) always defines a martingale.
- ▶ **92.** [M20] Let  $\langle Q_n \rangle$  be the Doob martingale that corresponds to Pólya’s urn (27) when  $Q = X_m$ , for some fixed  $m > 0$ . Calculate  $Q_0, Q_1, Q_2$ , etc.
- 93.** [M20] Solve the text’s hashing problem under the more general model considered in the bin-packing problem: Each variable  $X_n$  has probability  $p_{nk}$  of being equal to  $k$ , for  $1 \leq n \leq t$  and  $1 \leq k \leq m$ . What formula do you get instead of (44)?
- ▶ **94.** [M22] Where is the fact that the variables  $\{X_1, \dots, X_t\}$  are independent used in the previous exercise?
- 95.** [M20] True or false: “Pólya’s urn q.s. accumulates more than 100 red balls.”
- 96.** [HM22] Let  $X$  be the number of heads seen in  $n$  flips of an unbiased coin. Decide whether each of the following statements about  $X$  is a.s., q.s., or neither, as  $n \rightarrow \infty$ :
  - (i)  $|X - n/2| < \sqrt{n \ln n}$ ;      (ii)  $|X - n/2| < \sqrt{n \ln n}$ ;
  - (iii)  $|X - n/2| < \sqrt{n \ln \ln n}$ ;      (iv)  $|X - n/2| < \sqrt{n}$ .
- ▶ **97.** [HM21] Suppose  $\lfloor n^{1+\delta} \rfloor$  items are hashed into  $n$  bins, where  $\delta$  is a positive constant. Prove that every bin q.s. gets between  $\frac{1}{2}n^\delta$  and  $2n^\delta$  of them.

stopping rule  
 Wald’s equation  
 convex function  
 submartingale  
 Pólya’s urn  
 stopping rule  
 maximal inequality  
 Pólya’s urn  
 Bernoulli distribution  
 Hoeffding–Azuma inequality  
 Taylor’s formula  
 Doob martingale  
 Pólya’s urn  
 hashing  
 independent  
 Pólya’s urn  
 coin  
 a.s.  
 q.s.  
 hashed

- **98.** [M21] Many algorithms are governed by a loop of the form

$$X \leftarrow n; \text{ while } X > 0, \text{ set } X \leftarrow X - F(X)$$

where  $F(X)$  is a random integer in the range  $[1..X]$ . We assume that each integer  $F(X)$  is completely independent of any previously generated values, subject only to the requirement that  $E F(j) \geq g_j$ , where  $0 < g_1 \leq g_2 \leq \dots \leq g_n$ .

Prove that the loop sets  $X \leftarrow X - F(X)$  at most  $1/g_1 + 1/g_2 + \dots + 1/g_n$  times, on the average. (“If one step reduces by  $g_n$ , then perhaps  $(1/g_n)$ th of a step reduces by 1.”)

- 99.** [HM30] Show that the result in the previous exercise holds even when the range of  $F(X)$  is  $(-\infty..X]$ , given  $0 < g_1 \leq \dots \leq g_n \leq g_{n+1} \leq \dots$ . (Thus  $X$  might *increase*.)

- 100.** [HM17] A certain randomized algorithm takes  $T$  steps, where  $\Pr(T = t) = p_t$  for  $1 \leq t \leq \infty$ . Prove that (a)  $\lim_{m \rightarrow \infty} E \min(m, T) = ET$ ; (b)  $ET < \infty$  implies  $p_\infty = 0$ .

- 101.** [HM22] Suppose  $X = X_1 + \dots + X_m$  is the sum of independent geometrically distributed random integers, with  $\Pr(X_k = n) = p_k(1 - p_k)^{n-1}$  for  $n \geq 1$ . Prove that  $\Pr(X \geq r\mu) \leq re^{1-r}$  for all  $r \geq 1$ , where  $\mu = EX = \sum_{k=1}^m 1/p_k$ .

- 102.** [M20] Cora collects coupons, using a random process. After already owning  $k - 1$  of them, her chance of success when trying for the  $k$ th is at least one chance in  $s_k$ , independent of any previous successes or failures. Prove that she will a.s. own  $m$  coupons before making  $(s_1 + \dots + s_m) \ln n$  trials. And she will q.s. need at most  $s_k \ln n \ln \ln n$  trials to obtain the  $k$ th coupon, for each  $k \leq m$ , if  $m = O(n^{1000})$ .

- **103.** [M30] This exercise is based on two functions of the ternary digits  $\{0, 1, 2\}$ :

$$f_0(x) = \max(0, x - 1); \quad f_1(x) = \min(2, x + 1).$$

- a) What is  $\Pr(f_{X_1}(f_{X_2}(\dots(f_{X_n}(i))\dots))) = j$ , for each  $i, j \in \{0, 1, 2\}$ , assuming that  $X_1, X_2, \dots, X_n$  are independent, uniformly random bits?  
 b) Here's an algorithm that computes  $f_{X_1}(f_{X_2}(\dots(f_{X_n}(i))\dots))$  for  $i \in \{0, 1, 2\}$ , and stops when all three values have coalesced to a common value:

Set  $a_0 a_1 a_2 \leftarrow 012$  and  $n \leftarrow 0$ . Then while  $a_0 \neq a_2$ , set  $n \leftarrow n + 1$ ,  $t_0 t_1 t_2 \leftarrow (X_n? 122: 001)$ , and  $a_0 a_1 a_2 \leftarrow a_{t_0} a_{t_1} a_{t_2}$ . Output  $a_0$ .

(Notice that  $a_0 \leq a_1 \leq a_2$  always holds.) What is the probability that this algorithm outputs  $j$ ? What are the mean and variance of  $N$ , the final value of  $n$ ?

- c) A similar algorithm computes  $f_{X_n}(\dots(f_{X_2}(f_{X_1}(i))\dots))$ , if we change ‘ $a_{t_0} a_{t_1} a_{t_2}$ ’ to ‘ $t_{a_0} t_{a_1} t_{a_2}$ ’. What's the probability of output  $j$  in *this* algorithm?  
 d) Why on earth are the results of (b) and (c) so different?  
 e) The algorithm in (c) doesn't really use  $a_1$ . Therefore we might try to speed up process (b) by cleverly evaluating the functions in the opposite direction. Consider the following subroutine, called  $\text{sub}(T)$ :

Set  $a_0 a_2 \leftarrow 02$  and  $n \leftarrow 0$ . Then while  $n < T$  set  $n \leftarrow n + 1$ ,  $X \leftarrow$  random bit, and  $a_0 a_2 \leftarrow (X_n? f_1(a_0)f_1(a_2): f_0(a_0)f_0(a_2))$ . If  $a_0 = a_2$  output  $a_0$ , otherwise output  $-1$ .

Then the algorithm of (b) would seem to be equivalent to

Set  $T \leftarrow 1$ ,  $a \leftarrow -1$ ; while  $a < 0$  set  $T \leftarrow 2T$  and  $a \leftarrow \text{sub}(T)$ ; output  $a$ .

Prove, however, that this fails. (Randomized algorithms can be quite delicate!)

- f) Patch the algorithm of (e) and obtain a correct alternative to (b).

- 104.** [M21] Solve exercise 103(b) and 103(c) when each  $X_k$  is 1 with probability  $p$ .

loop  
 analysis of algorithms  
 randomized algorithm  
 geometrically distributed  
 tail inequalities  
 Larrie, Cora Mae  
 a.s.  
 q.s.  
 monus  
 saturating addition/subtraction  
 coalescing random walk  
 analysis of algorithms  
 forward versus backward  
 backward versus forward

- ▶ **105.** [M30] (*Random walk on an  $n$ -cycle.*) Given integers  $a$  and  $n$ , with  $0 \leq a \leq n$ , let  $N$  be minimum such that  $(a + (-1)^{X_1} + (-1)^{X_2} + \dots + (-1)^{X_N}) \bmod n = 0$ , where  $X_1, X_2, \dots$  is a sequence of independent random bits. Find the generating function  $g_a = \sum_{k=0}^{\infty} \Pr(N = k) z^k$ . What are the mean and variance of  $N$ ?
- 106.** [M25] Consider the algorithm of exercise 103(b) when the digits are  $d$ -ary instead of ternary; thus  $f_0(x) = \max(0, x - 1)$  and  $f_1(x) = \min(d - 1, x + 1)$ . Find the generating function, mean, and variance of the number  $N$  of steps required before  $a_0 = a_1 = \dots = a_{d-1}$  is first reached in this more general situation.
- ▶ **107.** [M22] (*Coupling.*) If  $X$  is a random variable on the probability space  $\Omega'$  and  $Y$  is another random variable on another probability space  $\Omega''$ , we can study them together by redefining them on a common probability space  $\Omega$ . All conclusions about  $X$  or  $Y$  are valid with respect to  $\Omega$ , provided that we have  $\Pr(X = x) = \Pr'(X = x)$  and  $\Pr(Y = y) = \Pr''(Y = y)$  for all  $x$  and  $y$ .

Random walk  
 $n$ -cycle  
 generating function  
 generating function  
 analysis of algs  
 Coupling  
 partially ordered set  
 row sums  
 column sums  
 flow in a network  
 max-flow min-cut theorem

Such “coupling” is obviously possible if we let  $\Omega$  be the set  $\Omega' \times \Omega''$  of pairs  $\{\omega'\omega'' \mid \omega' \in \Omega' \text{ and } \omega'' \in \Omega''\}$ , and if we define  $\Pr(\omega'\omega'') = \Pr'(\omega') \Pr''(\omega'')$  for each pair of events. But coupling can also be achieved in many other ways.

For example, suppose  $\Omega'$  and  $\Omega''$  each contain only two events,  $\{Q, K\}$  and  $\{\clubsuit, \spadesuit\}$ , with  $\Pr'(Q) = p$ ,  $\Pr'(K) = 1 - p$ ,  $\Pr''(\clubsuit) = q$ ,  $\Pr''(\spadesuit) = 1 - q$ . We could couple them with a four-event space  $\Omega = \{Q\clubsuit, Q\spadesuit, K\clubsuit, K\spadesuit\}$ , having  $\Pr(Q\clubsuit) = pq$ ,  $\Pr(Q\spadesuit) = p(1 - q)$ ,  $\Pr(K\clubsuit) = (1 - p)q$ ,  $\Pr(K\spadesuit) = (1 - p)(1 - q)$ . But if  $p < q$  we could also get by with just three events, letting  $\Pr(Q\clubsuit) = p$ ,  $\Pr(K\clubsuit) = q - p$ ,  $\Pr(K\spadesuit) = 1 - q$ . A similar scheme works when  $p > q$ , omitting  $K\clubsuit$ . And if  $p = q$  we need only two events,  $Q\clubsuit$  and  $K\spadesuit$ .

- a) Show that if  $\Omega'$  and  $\Omega''$  each have just three events, with respective probabilities  $\{p_1, p_2, p_3\}$  and  $\{q_1, q_2, q_3\}$ , they can always be coupled in a five-event space  $\Omega$ .
- b) Also, four events suffice if  $\{p_1, p_2, p_3\} = \{\frac{1}{12}, \frac{5}{12}, \frac{6}{12}\}$ ,  $\{q_1, q_2, q_3\} = \{\frac{2}{12}, \frac{3}{12}, \frac{7}{12}\}$ .
- c) But some three-event distributions cannot be coupled with fewer than five.

**108.** [HM21] If  $X$  and  $Y$  are integer-valued random variables such that  $\Pr'(X \geq n) \leq \Pr''(Y \geq n)$  for all integers  $n$ , find a way to couple them so that  $X \leq Y$  always holds.

**109.** [M27] Suppose  $X$  and  $Y$  have values in a finite partially ordered set  $P$ , and that

$$\Pr'(X \succeq a \text{ for some } a \in A) \leq \Pr''(Y \succeq a \text{ for some } a \in A), \quad \text{for all } A \subseteq P.$$

We will show that there's a coupling in which  $X \preceq Y$  always holds.

- a) Write out exactly what needs to be proved, in the simple case where  $P = \{1, 2, 3\}$  and the partial order has  $1 \prec 3, 2 \prec 3$ . (Let  $p_k = \Pr'(X = k)$  and  $q_k = \Pr''(Y = k)$  for  $k \in P$ . When  $P = \{1, \dots, n\}$ , a coupling is an  $n \times n$  matrix  $(p_{ij})$  of nonnegative probabilities whose row sums are  $\sum_j p_{ij} = p_i$  and column sums are  $\sum_i p_{ij} = q_j$ .) Compare this to the result proved in the preceding exercise.
- b) Prove that  $\Pr'(X \preceq b \text{ for some } b \in B) \geq \Pr''(Y \preceq b \text{ for some } b \in B)$ , for all  $B \subseteq P$ .
- c) A coupling between  $n$  pairs of events can be viewed as a flow in a network that has  $2n + 2$  vertices  $\{s, x_1, \dots, x_n, y_1, \dots, y_n, t\}$ , where there are  $p_i$  units of flow from  $s$  to  $x_i$ ,  $p_{ij}$  units of flow from  $x_i$  to  $y_j$ , and  $q_j$  units of flow from  $y_j$  to  $t$ . The “max-flow min-cut theorem” [see Section 7.5.3] states that such a flow is possible if and only if there are no subsets  $I, J \subseteq \{1, \dots, n\}$  such that (i) every path from  $s$  to  $t$  goes through some arc  $s \rightarrow x_i$  for  $i \in I$  or some arc  $y_j \rightarrow t$  for  $j \in J$ , and (ii)  $\sum_{i \in I} p_i + \sum_{j \in J} q_j < 1$ . Use that theorem to prove the desired result.

**110.** [M25] If  $X$  and  $Y$  take values in  $\{1, \dots, n\}$ , let  $p_k = \Pr'(X = k)$ ,  $q_k = \Pr''(Y = k)$ , and  $r_k = \min(p_k, q_k)$  for  $1 \leq k \leq n$ . The probability that  $X = Y$  in any coupling is obviously at most  $r = \sum_{k=1}^n r_k$ .

- a) Show that there always is a coupling with  $\Pr(X = Y) = r$ .
- b) Can the result of the previous exercise be extended, so that we have not only  $\Pr(X \leq Y) = 1$  but also  $\Pr(X = Y) = r$ ?

minwise independent  
 sketch  
 least common multiple  
 combinatorial nullstellensatz  
 Nullstellensatz, combinatorial  
 polynomial  
 degree  
 grid  
 covered  
 diagonal lines

► **111.** [M20] A family of  $N$  permutations of the numbers  $\{1, \dots, n\}$  is called *minwise independent* if, whenever  $1 \leq j \leq k \leq n$  and  $\{a_1, \dots, a_k\} \subseteq \{1, \dots, n\}$ , exactly  $N/k$  of the permutations  $\pi$  have  $\min(a_1\pi, \dots, a_k\pi) = a_j$ .  
 For example, the family  $F$  of  $N = 60$  permutations obtained by cyclic shifts of  
 123456, 126345, 152346, 152634, 164235, 154263, 165324, 164523, 156342, 165432

can be shown to be minwise independent permutations of  $\{1, 2, 3, 4, 5, 6\}$ .

- a) Verify the independence condition for  $F$  in the case  $k = 3, a_1 = 1, a_2 = 3, a_3 = 4$ .
- b) Suppose we choose a random  $\pi$  from a minwise independent family, and assign the “sketch”  $S_A = \min_{a \in A} a\pi$  to every  $A \subseteq \{1, \dots, n\}$ . Prove that, if  $A$  and  $B$  are arbitrary subsets,  $\Pr(S_A = S_B) = |A \cap B| / |A \cup B|$ .
- c) Given three subsets  $A, B, C$ , what is  $\Pr(S_A = S_B = S_C)$ ?

**112.** [M25] The size of a family  $F$  of minwise independent permutations must be a multiple of  $k$  for each  $k \leq n$ , by definition. In this exercise we’ll see how to construct such a family with the minimum possible size, namely  $N = \text{lcm}(1, 2, \dots, n)$ .

The basic idea is that, if all elements of the permutations in  $F$  that exceed  $m$  are replaced by  $\infty$ , the “truncated” family is still minwise independent in the sense that, if  $\min_{a \in \pi} a\pi = \infty$ , we can imagine that the minimum occurs at a random element of  $A$ . (This can happen only if  $\pi$  takes *all* elements of  $A$  to  $\infty$ .)

- a) Conversely, show that an  $m$ -truncated family can be lifted to an  $(m+1)$ -truncated family if, for each subset  $B$  of size  $n - m$ , we insert  $m + 1$  equally often into each of  $B$ ’s  $n - m$  positions, within the permutations whose  $\infty$ ’s are in  $B$ .
- b) Use this principle to construct minimum-size families  $F$ .

**113.** [M25] Although minwise permutations are defined only in terms of the minimum operation, a minwise independent family actually turns out to be also maxwise independent — and even more is true!

- a) Let  $E$  be the event that  $a_i\pi < k, b\pi = k$ , and  $c_j\pi > k$ , for any disjoint sets  $\{a_1, \dots, a_i\}, \{b\}, \{c_1, \dots, c_r\} \subseteq \{1, \dots, n\}$ . Prove that, if  $\pi$  is chosen randomly from a minwise independent set,  $\Pr(E)$  is the same as the probability that  $E$  occurs when  $\pi$  is chosen randomly from the set of all permutations. (For example,  $\Pr(5\pi < 7, 2\pi = 7, 1\pi > 7, 8\pi > 7) = 6(n - 7)(n - 8)(n - 4)!/n!$ , whenever  $n \geq 8$ .)
- b) Furthermore, if  $\{a_1, \dots, a_k\} \subseteq \{1, \dots, n\}$ , the probability that  $a_j$  is the  $r$ th largest element of  $\{a_1\pi, \dots, a_k\pi\}$  is  $1/k$ , whenever  $1 \leq j, r \leq k$ .

► **114.** [M28] (*The “combinatorial nullstellensatz.”*) Let  $f(x_1, \dots, x_n)$  be a polynomial in which the coefficient of  $x_1^{d_1} \dots x_n^{d_n}$  is nonzero and each term has degree  $\leq d_1 + \dots + d_n$ . Given subsets  $S_1, \dots, S_n$  of the field of coefficients, with  $|S_j| > d_j$  for  $1 \leq j \leq n$ , choose  $X_1, \dots, X_n$  independently and uniformly, with each  $X_j \in S_j$ . Prove that

$$\Pr(f(X_1, \dots, X_n) \neq 0) \geq \frac{|S_1| + \dots + |S_n| - (d_1 + \dots + d_n + n) + 1}{|S_1| \dots |S_n|}.$$

*Hint:* See exercise 4.6.1–16.

**115.** [M21] Prove that an  $m \times n$  grid cannot be fully covered by  $p$  horizontal lines,  $q$  vertical lines,  $r$  diagonal lines of slope  $+1$ , and  $r$  diagonal lines of slope  $-1$ , if  $m = p + 2\lfloor r/2 \rfloor + 1$  and  $n = q + 2\lfloor r/2 \rfloor + 1$ . *Hint:* Apply exercise 114 to a suitable polynomial  $f(x, y)$ .

**116.** [HM25] Use exercise 114 to prove that, if  $p$  is prime, any multigraph  $G$  on  $n$  vertices with more than  $(p - 1)n$  edges contains a nonempty subgraph in which the degree of every vertex is a multiple of  $p$ . (In particular, if each vertex of  $G$  has fewer than  $2p$  neighbors,  $G$  contains a  $p$ -regular subgraph. A loop from  $v$  to itself adds two to  $v$ 's degree.) *Hint:* Let the polynomial contain a variable  $x_e$  for each edge  $e$  of  $G$ .

► **117.** [HM25] Let  $X$  have the binomial distribution  $B_n(p)$ , so that  $\Pr(X = k) = \binom{n}{k} p^k (1 - p)^{n - k}$  for  $0 \leq k \leq n$ . Prove that  $X \bmod m$  is approximately uniform:

$$\left| \Pr(X \bmod m = r) - \frac{1}{m} \right| < \frac{2}{m} \sum_{j=1}^{\infty} e^{-8p(1-p)j^2 n/m^2}, \quad \text{for } 0 \leq r < m.$$

**118.** [M20] Prove that the second moment principle implies the *Paley-Zygmund inequality*

$$\Pr(X \geq x) \geq \frac{(\mathbb{E} X - x)^2}{\mathbb{E} X^2}, \quad \text{if } 0 \leq x \leq \mathbb{E} X.$$

**119.** [HM24] Let  $x$  be a fixed value in  $[0..1]$ . Prove that, if we independently and uniformly choose  $U \in [0..x]$ ,  $V \in [x..1]$ ,  $W \in [0..1]$ , then the median  $\langle UVW \rangle$  is uniformly distributed in  $[\min(U, V, W) .. \max(U, V, W)]$ .

**120.** [M20] Consider random binary search trees  $T_n$  obtained by successively inserting independent uniform deviates  $U_1, U_2, \dots$  into an initially empty tree. Let  $T_{nk}$  be the number of external nodes on level  $k$ , and define  $T_n(z) = \sum_{k=0}^{\infty} T_{nk} z^k / (n+1)$ . Prove that  $Z_n = T_n(z) / g_{n+1}(z)$  is a martingale, where  $g_n(z) = (2z + n - 2)(2z + n - 3) \dots (2z) / n!$  is the generating function for the cost of the  $n$ th insertion (exercise 6.2.2-6).

► **121.** [M25] Let  $X$  and  $Y$  be random variables with the distributions  $\Pr(X = t) = x(t)$  and  $\Pr(Y = t) = y(t)$ . The ratio  $\rho(t) = y(t)/x(t)$ , which may be infinity, is called the probability density of  $Y$  with respect to  $X$ . We define the *relative entropy of  $X$  with respect to  $Y$* , also called the *Kullback-Leibler divergence of  $Y$  from  $X$* , by the formulas

$$D(y||x) = \mathbb{E}(\rho(X) \lg \rho(X)) = \mathbb{E} \lg \rho(Y) = \sum_t y(t) \lg \frac{y(t)}{x(t)},$$

with  $0 \lg 0$  and  $0 \lg(0/0)$  understood to mean 0. It can be viewed intuitively as the number of bits of information that are lost when  $X$  is used to approximate  $Y$ .

- a) Suppose  $X$  is a random six-sided die with the uniform distribution, but  $Y$  is a “loaded” die in which  $\Pr(Y = \square) = \frac{1}{5}$  and  $\Pr(Y = \blacksquare) = \frac{2}{15}$ , instead of  $\frac{1}{6}$ . Compute  $D(y||x)$  and  $D(x||y)$ .
- b) Prove that  $D(y||x) \geq 0$ . When is it zero?
- c) If  $p = \Pr(X \in T)$  and  $q = \Pr(Y \in T)$ , show that  $\mathbb{E}(\lg \rho(Y) | Y \in T) \geq \lg(q/p)$ .
- d) Suppose  $x(t) = 1/m$  for all  $t$  in an  $m$ -element set  $S$ , and  $y(t) \neq 0$  only when  $t \in S$ . Express  $D(y||x)$  in terms of the *entropy*  $H_Y = \mathbb{E} \lg(1/Y)$  (see Eq. 6.2.2-(18)).
- e) Let  $Z(u, v) = \Pr(X = u \text{ and } Y = v)$  when  $X$  and  $Y$  have any joint distribution, and let  $W(u, v)$  be that same probability under the assumption that  $X$  and  $Y$  are independent. The *joint entropy*  $H_{X,Y}$  is defined to be  $H_Z$ , and the *mutual information*  $I_{X,Y}$  is defined to be  $D(z||w)$ . Prove that  $H_W = H_X + H_Y$  and  $I_{X,Y} = H_W - H_Z$ . (Consequently  $H_{X,Y} \leq H_X + H_Y$ , and  $I_{X,Y}$  measures the difference.)

**122.** [HM24] Continuing exercise 121, compute  $D(y||x)$  and  $D(x||y)$  when

- a)  $x(t) = 1/2^{t+1}$  and  $y(t) = 3^t/4^{t+1}$  for  $t = 0, 1, 2, \dots$ ;
- b)  $x(t) = e^{-np} (np)^t / t!$  and  $y(t) = \binom{n}{t} p^t (1 - p)^{n-t}$ , for  $t \geq 0$  and  $0 < p < 1$ . (Give asymptotic answers with absolute error  $O(1/n)$ , for fixed  $p$  as  $n \rightarrow \infty$ .)

multigraph  
 regular  
 loop  
 binomial distribution  
 second moment principle  
 Paley  
 Zygmund  
 median  
 uniformly distributed  
 binary search trees  
 uniform deviates  
 martingale  
 generating function  
 Density, relative  
 Entropy, relative  
 Kullback  
 Leibler  
 die  
 uniform distribution  
 “loaded” die  
 entropy  
 joint distribution  
 joint entropy  
 mutual information  
 geometric distribution  
 Poisson distribution  
 binomial distribution

► **123.** [M20] Let  $X$  and  $Y$  be as in exercise 121. The random variable  $Z = A? Y: X$  either has the distribution  $x(t)$  or  $y(t)$ , but we don't know whether  $A$  is true or false. If we believe that the hypothesis  $Z = Y$  holds with the *a priori* probability  $\Pr(A) = p_k$ , we assume that  $z_k(t) = \Pr_k(Z = t) = p_k x(t) + (1 - p_k)y(t)$ . But after seeing a new value of  $Z$ , say  $Z = Z_k$ , we will believe the hypothesis with the *a posteriori* probability  $p_{k+1} = \Pr(A | Z_k)$ . Show that  $D(y||x)$  is the expected "information gained,"  $\lg(p_{k+1}/(1 - p_{k+1})) - \lg(p_k/(1 - p_k))$ , averaged with respect to the distribution of  $Y$ .

a priori  
information gained  
Importance sampling  
DARWIN  
von Mengden

**124.** [HM22] (*Importance sampling.*) In the setting of exercise 121, we have  $E f(Y) = E(\rho(X) f(X))$  for any function  $f$ ; thus  $\rho(t)$  measures the "importance" of the  $X$ -value  $t$  with respect to the  $Y$ -value  $t$ . Many situations arise when it's easy to generate random variables with an approximate distribution  $x(t)$ , but difficult to generate them with the exact distribution  $y(t)$ . In such cases we can estimate the average value  $E(f) = E f(Y)$  by calculating  $E_n(f) = (\rho(X_1)f(X_1) + \dots + \rho(X_n)f(X_n))/n$ , where the  $X_j$  are independent random variables, each distributed as  $x(t)$ .

Let  $n = c^4 2^{D(y||x)}$ . Prove that if  $c > 1$ , this estimate  $E_n$  is relatively accurate:

$$|E(f) - E_n(f)| \leq \|f\| (1/c + 2\sqrt{\Delta_c}), \quad \text{where } \Delta_c = \Pr(\rho(Y) > c^2 2^{D(y||x)}).$$

(Here  $\|f\|$  denotes  $(E f(Y)^2)^{1/2}$ .) On the other hand if  $c < 1$  the estimate is poor:

$$\Pr(E_n(1) \geq a) \leq c^2 + (1 - \Delta_c)/a. \quad \text{for } 0 < a < 1,$$

Here '1' denotes the constant function  $f(y) = 1$  (hence  $E(1) = 1$ ).

*Every man must judge for himself between conflicting vague probabilities.*

— CHARLES DARWIN, letter to N. A. von Mengden (5 June 1879)



## ANSWERS TO EXERCISES

*It isn't that they can't see the solution.  
It is that they can't see the problem.*

— G. K. CHESTERTON, *The Scandal of Father Brown* (1935)

### MATHEMATICAL PRELIMINARIES REDUX

1. (a)  $A$  beats  $B$  in  $5+0+5+5+0+5$  cases out of 36;  $B$  beats  $C$  in  $4+2+4+4+2+4$ ;  $C$  beats  $A$  in  $2+2+2+6+2+6$ .

(b) The unique solution, without going to more than six spots per face, is

$$A = \begin{array}{|c|c|c|c|} \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \end{array}, \quad B = \begin{array}{|c|c|c|c|} \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \end{array}, \quad C = \begin{array}{|c|c|c|c|} \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \end{array}.$$

(c)  $A = \{F_{m-2} \times 1, F_{m-1} \times 4\}$ ,  $B = \{F_m \times 3\}$ ,  $C = \{F_{m-1} \times 2, F_{m-2} \times 5\}$  makes  $\Pr(C > A) = F_{m-2}F_{m+1}/F_m^2$ ; and we have  $F_{m-2}F_{m+1} = F_{m-1}F_m - (-1)^m$ . [Similarly, with  $n$  faces and  $A = \{\lfloor n/\phi^2 \rfloor \times 1, \lceil n/\phi \rceil \times 4\}$ , etc. the probabilities are  $1/\phi - O(1/n)$ . See R. P. Savage, Jr., *AMM* **101** (1994), 429–436.]

2. Let  $\Pr(A > B) = \mathcal{A}$ ,  $\Pr(B > C) = \mathcal{B}$ ,  $\Pr(C > A) = \mathcal{C}$ . We can assume that no  $x$  appears on more than one die; if it did, we could replace it by  $x + \epsilon$  in  $A$  and  $x - \epsilon$  in  $C$  (for small enough  $\epsilon$ ) without decreasing  $\mathcal{A}$ ,  $\mathcal{B}$ , or  $\mathcal{C}$ . So we can list the face elements in nondecreasing order and replace each one by the name of its die; for example, the previous answer (b) yields  $CBBBAAAAACCCCBBA$ . Clearly  $AB$ ,  $BC$ , and  $CA$  are never consecutive in an optimal arrangement of this kind:  $BA$  is always better than  $AB$ .

Suppose the sequence is  $C^{c_1}B^{b_1}A^{a_1} \dots C^{c_k}B^{b_k}A^{a_k}$  where  $c_i > 0$  for  $1 \leq i \leq k$  and  $b_i, a_i > 0$  for  $1 \leq i < k$ . Let  $\alpha_i = a_i/(a_1 + \dots + a_k)$ ,  $\beta_i = b_i/(b_1 + \dots + b_k)$ ,  $\gamma_i = c_i/(c_1 + \dots + c_k)$ ; then  $\mathcal{A} = \alpha_1\beta_1 + \alpha_2(\beta_1 + \beta_2) + \dots$ ,  $\mathcal{B} = \beta_1\gamma_1 + \beta_2(\gamma_1 + \gamma_2) + \dots$ ,  $\mathcal{C} = \gamma_2\alpha_1 + \gamma_3(\alpha_1 + \alpha_2) + \dots$ . We will show that  $\min(\mathcal{A}, \mathcal{B}, \mathcal{C}) \leq 1/\phi$  when the  $\alpha$ 's,  $\beta$ 's, and  $\gamma$ 's are nonnegative real numbers; then it is  $< 1/\phi$  when they are rational.

The key idea is that we can assume  $k \leq 2$  and  $\alpha_2 = 0$ . Otherwise the following transformation leads to a shorter array without decreasing  $\mathcal{A}$ ,  $\mathcal{B}$ , or  $\mathcal{C}$ :

$$\gamma'_2 = \lambda\gamma_2, \quad \gamma'_1 = \gamma_1 + \gamma_2 - \gamma'_2, \quad \beta'_2 = \lambda\beta_2, \quad \beta'_1 = \beta_1 + \beta_2 - \beta'_2, \quad \alpha'_1 = \alpha_1/\lambda, \quad \alpha'_2 = \alpha_1 + \alpha_2 - \alpha'_1.$$

Indeed,  $\mathcal{A}' = \mathcal{A}$ ,  $\mathcal{C}' = \mathcal{C}$ , and  $\mathcal{B}' - \mathcal{B} = (1 - \lambda)(\beta_1 - \lambda\beta_2)\gamma_2$ , and we can choose  $\lambda$  thus:

*Case 1:*  $\beta_1 \geq \beta_2$ . Choose  $\lambda = \alpha_1/(\alpha_1 + \alpha_2)$ , making  $\alpha'_2 = 0$ .

*Case 2:*  $\beta_1 < \beta_2$  and  $\gamma_1/\gamma_2 \leq \beta_1/\beta_2$ . Choose  $\lambda = 1 + \gamma_1/\gamma_2$ , making  $\gamma'_1 = 0$ .

*Case 3:*  $\beta_1 < \beta_2$  and  $\gamma_1/\gamma_2 > \beta_1/\beta_2$ . Choose  $\lambda = 1 + \beta_1/\beta_2$ , making  $\beta'_1 = 0$ .

Finally, then,  $\mathcal{A} = \beta_1$ ,  $\mathcal{B} = 1 - \beta_1\gamma_2$ ,  $\mathcal{C} = \gamma_2$ ; they can't all be greater than  $1/\phi$ .

[Similarly, with  $n$  dice, the asymptotic optimum probability  $p_n$  satisfies  $p_n = \alpha_2^{(n)} = 1 - \alpha_1^{(n-1)}\alpha_2^{(n)} = \dots = 1 - \alpha_1^{(2)}\alpha_2^{(3)} = \alpha_1^{(2)}$ . One can show that  $f_n(1 - p_n) = 0$ ,

where  $f_{n+1}(x) = f_n(x) - x f_{n-1}(x)$ ,  $f_0(x) = 1$ ,  $f_1(x) = 1 - x$ . Then  $f_n(x^2)$  is expressible as the Chebyshev polynomial  $x^{n+1} U_{n+1}(\frac{1}{2x})$ ; and we have  $p_n = 1 - 1/(4 \cos^2 \pi/(n+2))$ . See Z. Usiskin, *Annals of Mathematical Statistics* **35** (1964), 857–862; S. Trybuła, *Zastosowania Matematyki* **8** (1965), 143–156.]

Chebyshev polynomial  
Usiskin  
Trybuła  
Moraleda  
Stork  
quite surely

3. Brute force (namely a program) finds eight solutions, of which the simplest is

$$A = \begin{array}{|c|c|c|c|} \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \end{array}, \quad B = C = \begin{array}{|c|c|c|c|} \hline \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet \\ \hline \end{array},$$

all with respective probabilities  $\frac{17}{27}, \frac{16}{27}, \frac{16}{27}$ . [If  $\begin{array}{|c|c|} \hline \bullet & \bullet \\ \hline \end{array}$  is also allowed, the unique solution

$$A = \begin{array}{|c|c|c|c|c|c|} \hline \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \hline \end{array}, \quad B = \begin{array}{|c|c|c|c|c|c|} \hline \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \hline \end{array}, \quad C = \begin{array}{|c|c|c|c|c|c|} \hline \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \hline \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \hline \end{array}$$

has the property that every roll has exactly one die below the average and two above, with each of  $A, B, C$  equally likely to be below; hence all three probabilities are  $2/3$ . See J. Moraleda and D. G. Stork, *College Mathematics Journal* **43** (2012), 152–159.]

4. (a) The permutation  $(1234)(56)$  takes  $A \rightarrow B \rightarrow C \rightarrow D \rightarrow A$ . So  $B$  versus  $C$  is like  $A$  versus  $B$ , etc. Also  $\Pr(A \text{ beats } C) = \Pr(C \text{ beats } A) = \Pr(B \text{ beats } D) = \Pr(D \text{ beats } B) = \frac{288}{720}$ ;  $\Pr(A \text{ and } C \text{ tie}) = \Pr(B \text{ and } D \text{ tie}) = \frac{144}{720}$ .

(b) Assume by symmetry the players are  $A, B, C$ . Then the bingers are  $(A, B, C, AB, AC, BC, ABC)$  with respective probabilities  $(168, 216, 168, 48, 72, 36, 12)/720$ .

(c) It's  $(A, AB, AC, ABC, ABCD)$  with probabilities  $(120, 24, 48, 12, 0)/720$ .

5. (a) If  $A_k = 1001$  with probability .99, otherwise  $A_k = 0$ , but  $B_k = 1000$  always, then  $P_{1000} = .99^{1000} \approx .000043$ . (This example gives the smallest possible  $P_{1000}$ , because  $\Pr((A_1 - B_1) + \dots + (A_n - B_n) > 0) \geq \Pr([A_1 > B_1] \dots [A_n > B_n]) = P_1^n$ .)

(b) Let  $E = q_0 + q_2 + q_4 + \dots \approx 0.67915$  be the probability that  $B = 0$ . Then  $\Pr(A > B) = \sum_{k=0}^{\infty} q_{2k}(E + \sum_{j=0}^{k-1} q_{2j+1}) \approx .47402$ ;  $\Pr(A < B) = \sum_{k=0}^{\infty} q_{2k+1}(1 - E + \sum_{j=0}^k q_{2j}) \approx .30807$ ; and  $\Pr(A = B) = \Pr(A = B = 0) = E(1 - E) \approx .21790$  is also the probability that  $AB > 0$ .

(c) During the first  $n_k$  rounds, the probability that either Alice or Bob has scored more than  $m_k$  is at most  $n_k(q_{k+1} + q_{k+2} + \dots) = O(2^{-k})$ ; and the probability that neither has ever scored  $m_k$  is  $(1 - q_k)^{n_k} < \exp(-q_k n_k) = \exp(-2^k/D)$ . Also  $m_k > n_k m_{k-1}$  when  $k > 1$ . Thus Alice “quite surely” wins when  $k$  is even, but loses when  $k$  is odd, as  $k \rightarrow \infty$ . [*The American Statistician* **43** (1989), 277–278.]

6. The probability that  $X_j = 1$  is clearly  $p_1 = 1/(n-1)$ ; hence  $X_j = 0$  with probability  $p_0 = (n-2)/(n-1)$ . And the probability that  $X_i = X_j = 1$  when  $i < j$  is  $p_1^2$ . Thus (see exercise 20),  $(X_i, X_j)$  will equal  $(0, 1), (1, 0)$ , or  $(0, 0)$  with the correct probabilities  $p_0 p_1, p_1 p_0, p_0 p_0$ . But  $X_i = X_j = X_k = 1$  with probability 0 when  $i < j < k$ .

For 3-wise independence let  $\Pr(X_1 \dots X_n = x_1 \dots x_n) = a_{x_1 + \dots + x_n} / (n-2)^3$ , where  $a_0 = 2 \binom{n-2}{3}$ ,  $a_1 = \binom{n-2}{2}$ ,  $a_3 = 1$ , otherwise  $a_j = 0$ .

7. Let  $f_m(n) = \sum_{j=0}^m \binom{n}{j} (-1)^j (n+1-m)^{m-j}$ , and define probabilities via  $a_j = f_{k-j}(n-j)$  as in answer 6. (In particular, we have  $f_0(n) = 1, f_1(n) = 0, f_2(n) = \binom{n-1}{2}, f_3(n) = 2 \binom{n-2}{3}, f_4(n) = 3 \binom{n-3}{4} + \binom{n-2}{2}^2$ .) This definition is valid if we can prove that  $f_m(n) \geq 0$  for  $n \geq m$ , because of the identity  $\sum_j \binom{n}{j} f_{m-j}(n-j) = (n+1-m)^m$ .

To prove that inequality, Schulte-Geers notes (see *CMath* (5.19)) that  $f_m(n) = \sum_{k=0}^m \binom{m-n}{k} (n-m)^{m-k} = \sum_{k=0}^m \binom{n-m-1+k}{k} (-1)^k (n-m)^{m-k}$ ; these terms pair up nicely to yield  $\sum_{k=0}^{m-1} k \binom{n-m-1+k}{k+1} (n-m)^{m-k-1} [k \text{ even}] + \binom{n-1}{m} [m \text{ even}]$ .

8. If  $0 < k < n$ , the probability that  $k$  of the variables have any particular setting is  $1/2^k$ , because the remaining variables have even parity as often as odd parity. So there's  $(n-1)$ -wise independence, but not  $n$ -wise.

9. Give probability  $1/2$  to  $0\dots 0$  and  $1\dots 1$ ; all other vectors have probability 0.

10. If  $n > p$  we have  $X_{p+1} = X_1$ , so there's no independence. Otherwise, if  $m < n \leq p$ , there's  $m$ -wise independence because any  $m$  vectors  $(1, j, \dots, j^{m-1})$  are linearly independent modulo  $p$  (they're columns of Vandermonde's matrix, exercise 1.2.3–37); but the  $X$ 's are dependent  $(m+1)$ -wise, because a polynomial of degree  $m$  cannot have  $m+1$  different roots. If  $m \geq n$  and  $n \leq p$  there is complete independence.

Instead of working mod  $p$ , we could use any finite field in this construction.

11. We can assume that  $n = 1$ , because  $(X_1 + \dots + X_n)/n$  and  $(X_{n+1} + \dots + X_{2n})/n$  are independent random variables with the same discrete distribution. Then  $\Pr(|X_1 + X_2 - 2\alpha| \leq 2|X_1 - \alpha|) \geq \Pr(|X_1 - \alpha| + |X_2 - \alpha| \leq 2|X_1 - \alpha|) = \Pr(|X_2 - \alpha| \leq |X_1 - \alpha|) = (1 + \Pr(X_1 = X_2))/2 > 1/2$ . [This exercise was suggested by T. M. Cover.]

12. Let  $w = \Pr(A \text{ and } B)$ ,  $x = \Pr(A \text{ and } \bar{B})$ ,  $y = \Pr(\bar{A} \text{ and } B)$ ,  $z = \Pr(\bar{A} \text{ and } \bar{B})$ . All five statements are equivalent to  $wz > xy$ , or to  $\left| \frac{w}{y} \frac{x}{z} \right| > 0$ , or to “ $A$  and  $B$  are strictly positively correlated” (see exercise 61). [This exercise was suggested by E. Georgiadis.]

13. False in many cases. For example, take  $\Pr(\bar{A} \text{ and } \bar{B} \text{ and } \bar{C}) = \Pr(\bar{A} \text{ and } B \text{ and } \bar{C}) = 0$ ,  $\Pr(A \text{ and } B \text{ and } C) = 2/7$ , and all other probabilities  $1/7$ .

14. Induction on  $n$ . [*Philosophical Transactions* **53** (1763), 370–418, proof of Prop. 6.]

15. If  $\Pr(C) > 0$ , this is the chain rule, conditional on  $C$ . But if  $\Pr(C) = 0$ , it's false by our conventions, unless  $A$  and  $B$  are independent.

16. If and only if  $\Pr(\bar{A} \cap B \cap C) = 0 \neq \Pr(B)$  or  $\Pr(\bar{A} \cap C) = 0$ .

17.  $4/51$ , because four of the cards other than  $\mathbf{Q}\spadesuit$  are aces.

18. Since  $(M - X)(X - m) \geq 0$ , we have  $(M \mathbb{E} X) - (\mathbb{E} X^2) + (m \mathbb{E} X) - mM \geq 0$ . [See C. Davis and R. Bhatia, *AMM* **107** (2000), 353–356, for generalizations.]

19. (a) The binary values of  $\Pr(X_n = 1) = \mathbb{E}(X_n)$  for  $n = 0, 1, 2, \dots$ , are respectively  $(.01010101010101\dots)_2$ ,  $(.0011001100110011\dots)_2$ ,  $(.0000111100001111\dots)_2$ ,  $\dots$ ; thus they're the complemented reflections of the “magic masks” 7.1.3–(47). The answer is therefore  $(2^{2^n} - 1)/(2^{2^{n+1}} - 1) = 1/(2^{2^n} + 1)$ .

(b)  $\Pr(X_0 X_1 \dots X_{n-1} = x_0 x_1 \dots x_{n-1}) = 2^{(\bar{x}_{n-1} \dots \bar{x}_1 \bar{x}_0)_2} / (2^{2^n} - 1)$  can be “read off” from the magic masks by ANDing and complementing. [See E. Lukacs, *Characteristic functions* (1960), 119, for related theory.]

(c) The infinite sum  $S$  is well defined because  $\Pr(S = \infty) = 0$ . Its expectation  $\mathbb{E} S = \sum_{n=0}^{\infty} 1/(2^{2^n} + 1) \approx 0.59606$  corresponds to the case  $z = 1/2$  in answer 7.1.3–41(c). By independence,  $\text{var } S = \sum_{n=0}^{\infty} \text{var } X_n = \sum_{n=0}^{\infty} 2^{2^n} / (2^{2^n} + 1)^2 \approx 0.44148$ .

(d) The parity number  $\mathbb{E} R = (.0110100110010110\dots)_2$  has the decimal value

$$0.41245\ 40336\ 40107\ 59778\ 33613\ 68258\ 45528\ 30895-,$$

and can be shown to equal  $\frac{1}{2} - \frac{1}{4}P$  where  $P = \prod_{k=0}^{\infty} (1 - 1/2^{2^k})$  [R. W. Gosper and R. Schroepfel, MIT AI Laboratory Memo 239 (29 February 1972), Hack 122], which is transcendental [K. Mahler, *Mathematische Annalen* **101** (1929), 342–366; **103** (1930), 532]. (Furthermore it turns out that  $1/P - 1/2 = \sum_{k=0}^{\infty} 1/\prod_{j=0}^{k-1} (2^{2^j} - 1)$ .) Since  $R$  is binary,  $\text{var}(R) = (\mathbb{E} R)(1 - \mathbb{E} R) \approx 0.242336$ .

Vandermonde's matrix  
Cover  
positively correlated  
Georgiadis  
chain rule  
Davis  
Bhatia  
magic masks  
Lukacs  
parity number  
Gosper  
Schroepfel  
HAKMEM  
transcendental  
Mahler  
Thue–Morse constant

(e) Zero (because  $\pi$  is irrational, hence  $p_0 + p_1 + \dots = \infty$ ). However, if we ask the analogous question for Euler's constant  $\gamma$  instead of  $\pi$ , nobody knows the answer.

(f)  $EY_n = 2EX_n$ ; in fact,  $\Pr(Y_0Y_1Y_2\dots = x_0x_1x_2\dots)$ , for *any* infinite string  $x_0x_1x_2\dots$ , is equal to  $2\Pr(X_0X_1X_2\dots = x_0x_1x_2\dots) \bmod 1$ , because we shift the binary representation one place to the left (and drop any carry). Thus in particular,  $EY_mY_n = 2EX_mX_n = \frac{1}{2}EY_mEY_n$  when  $m \neq n$ ;  $Y_m$  and  $Y_n$  are negatively correlated because  $\text{covar}(Y_m, Y_n) = -\frac{1}{2}EY_mEY_n$ .

(g) Clearly  $ET = 2ES$ . Also  $ET^2 = 2ES^2$ , because  $EY_mY_n = 2EX_mX_n$  for all  $m$  and  $n$ . So  $\text{var}(T) = 2(\text{var}(S) + (ES)^2) - (2ES)^2 = 2\text{var}(S) - 2(ES)^2 \approx 0.17237$ .

**20.** Let  $p_j = EX_j$ . We must prove, for example, that  $E(X_1(1-X_2)(1-X_3)X_4) = p_1(1-p_2)(1-p_3)p_4$  when  $k \geq 4$ . But this is  $E(X_1X_4 - X_1X_2X_4 - X_1X_3X_4 + X_1X_2X_3X_4) = p_1p_4 - p_1p_2p_4 - p_1p_3p_4 + p_1p_2p_3p_4$ .

**21.** From the previous exercise we know that they can't both be binary. Let  $X$  be binary and  $Y$  ternary, taking the values  $(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)$  with probabilities respectively proportional to  $(a, b, 3a+b+3d, d, 1, 1)$ . Then  $EXY = 3/D$ ,  $EX = 2/D$ , and  $EY = 3/2$ , where  $D = 4a + 2b + 4d + 2$ .

**22.** By (8) we have  $\Pr(A_1 \cup \dots \cup A_n) = E[A_1 \cup \dots \cup A_n] = E\max([A_1], \dots, [A_n]) \leq E([A_1] + \dots + [A_n]) = E[A_1] + \dots + E[A_n] = \Pr(A_1) + \dots + \Pr(A_n)$ .

**23.** The hinted probability is  $\Pr(X_s = 0 \text{ and } X_1 + \dots + X_{s-1} = s - r)$ , so it equals  $\binom{s-1}{s-r} p^{s-r} (1-p)^r$ . To get  $B_{m,n}(p)$ , sum it for  $r = n - m$  and  $n - m \leq s \leq n$ . [For an algebraic rather than probabilistic/combinatorial proof, see *CMath*, exercise 8.17.]

**24.** (a) The derivative of  $B_{m,n}(x) = \sum_{k=0}^m \binom{n}{k} x^k (1-x)^{n-k}$  is

$$\begin{aligned} B'_{m,n}(x) &= \sum_{k=0}^m \binom{n}{k} (kx^{k-1}(1-x)^{n-k} - (n-k)x^k(1-x)^{n-1-k}) \\ &= n \left( \sum_{k=0}^{m-1} \binom{n-1}{k} x^k (1-x)^{n-1-k} - \sum_{k=0}^m \binom{n-1}{k} x^k (1-x)^{n-1-k} \right) \\ &= -n \binom{n-1}{m} x^m (1-x)^{n-1-m}. \end{aligned}$$

[See Karl Pearson, *Biometrika* **16** (1924), 202–203.]

(b) The hint, which says that  $\int_0^{a/(a+b+1)} x^a(1-x)^b dx < \int_{a/(a+b+1)}^1 x^a(1-x)^b dx$  when  $0 \leq a \leq b$ , will prove that  $1 - B_{m,n}(m/n) < B_{m,n}(m/n)$ . It suffices to show that  $\int_0^{a/(a+b)} x^a(1-x)^b dx \leq \int_{a/(a+b)}^1 x^a(1-x)^b dx$ , because we have  $\int_0^{a/(a+b+1)} x^a(1-x)^b dx < \int_0^{a/(a+b)} x^a(1-x)^b dx < \int_{a/(a+b)}^1 x^a(1-x)^b dx$ . Let  $x = (a - \epsilon)/(a + b)$ , and observe that  $(a - \epsilon)^a(b + \epsilon)^b$  is less than or equal to  $(a + \epsilon)^a(b - \epsilon)^b$  for  $0 \leq \epsilon \leq a$ , because the quantity

$$\left( \frac{a - \epsilon}{a + \epsilon} \right)^a = e^{a(\ln(1-\epsilon/a) - \ln(1+\epsilon/a))} = \exp\left(-2\epsilon\left(1 + \frac{\epsilon^2}{3a^2} + \frac{\epsilon^4}{5a^4} + \dots\right)\right)$$

increases when  $a$  increases.

(c) Let  $t_k = \binom{n}{k} m^k (n-m)^{n-k}$ . When  $m \geq n/2$  we can show that  $1 - B_{m,n}(m/n) = \sum_{k>m} t_k/n^n < B_{m,n}(m/n) = \sum_{k=0}^m t_k/n^n$ , because  $t_{m+d} < t_{m+1-d}$  for  $1 \leq d \leq n-m$ . For if  $r_d = t_{m+d}/t_{m+1-d}$ , we have  $r_1 = m/(m+1) < 1$ ; also

$$\frac{r_{d+1}}{r_d} = \frac{(n-m+d)(n-m-d)m^2}{(m+1+d)(m+1-d)(n-m)^2} < 1,$$

because  $((m+1)^2 - d^2)(n-m)^2 - ((n-m)^2 - d^2)m^2 = (2m+1)(n-m)^2 + (2m-n)nd^2$ .

[Peter Neumann proved in *Wissenschaftliche Zeitschrift der Technischen Universität Dresden* **15** (1966), 223–226, that  $m$  is the median. The argument in part (c) is due to Nick Lord, in *The Mathematical Gazette* **94** (2010), 331–332.]

Neumann  
Lord  
Newton  
Sylvester  
Pitman

**25.** (a)  $\binom{n}{k} - \binom{n}{k+1}$  is  $\sum p_I q_J (q_t/(n-k) - p_t/(k+1))$ , summed over all partitions of  $\{1, \dots, n\}$  into disjoint sets  $I \cup J \cup \{t\}$ , where  $|I| = k$ ,  $|J| = n - k - 1$ ,  $p_I = \prod_{i \in I} p_i$ ,  $q_J = \prod_{j \in J} q_j$ . And  $q_t/(n-k) - p_t/(k+1) \geq 0 \iff p_t \leq (k+1)/(n+1)$ .

(b) Given  $p_1, \dots, p_{n-1}$ , the quantity  $\binom{n}{k}$  is maximized when  $p_n = p$ , by (a). The same argument applies symmetrically to all indices  $j$ .

**26.** The inequality is equivalent to  $r_{n,k}^2 \geq r_{n,k-1}r_{n,k+1}$ , which was stated without proof on pages 242–245 of Newton's *Arithmetica Universalis* (1707), then finally proved by Sylvester many years later [*Proc. London Math. Soc.* **1** (1865), 1–16]. We have  $nr_{n,k} = kp_n r_{n-1,k-1} + (n-k)q_n r_{n-1,k}$ ; hence  $n^2(r_{n,k}^2 - r_{n,k-1}r_{n,k+1}) = (p_n r_{n-1,k-1} - q_n r_{n-1,k})^2 + (k^2 - 1)p_n^2 A + (k-1)(n-1-k)p_n q_n B + ((n-k)^2 - 1)q_n^2 C$ , where  $A = r_{n-1,k-1}^2 - r_{n-1,k-2}r_{n-1,k}$ ,  $B = r_{n-1,k-1}r_{n-1,k} - r_{n-1,k-2}r_{n-1,k+1}$ , and  $C = r_{n-1,k}^2 - r_{n-1,k-1}r_{n-1,k+1}$  are nonnegative, by induction on  $n$ .

**27.**  $\sum_{k=0}^m \binom{n}{k} = \sum_{k=0}^m \binom{n-m-1+k}{k} (1 - p_{n-m+k})$ , by the same argument as before.

**28.** (a)  $\binom{n}{k} = \binom{n-2}{k}A + \binom{n-2}{k-1}B + \binom{n-2}{k-2}C$  and  $Eg(X) = \sum_{k=0}^{n-2} \binom{n-2}{k} h_k$ , where  $A = (1 - p_{n-1})(1 - p_n)$ ,  $C = p_{n-1}p_n$ ,  $B = 1 - A - C$ , and  $h_k = Ag(k) + Bg(k+1) + Cg(k+2)$ . If the  $p_j$ 's aren't all equal, we may assume that  $p_{n-1} < p < p_n$ . Setting  $p'_{n-1} = p_{n-1} + \epsilon$  and  $p'_n = p_n - \epsilon$ , where  $\epsilon = \min(p_n - p, p - p_{n-1})$ , changes  $A, B, C$  to  $A' = A + \delta$ ,  $B' = B - 2\delta$ ,  $C' = C + \delta$ , where  $\delta = (p_n - p)(p - p_{n-1})$ ; hence  $h_k$  changes to  $h'_k = h_k + \delta(g(k) - 2g(k+1) + g(k+2))$ . Convex functions satisfy  $g(k) - 2g(k+1) + g(k+2) \geq 0$ , by (19) with  $x = k$  and  $y = k+2$ ; hence we can permute the  $p$ 's and repeat this transformation until  $p_j = p$  for  $1 \leq j \leq n$ .

(b) Suppose  $Eg(X)$  is maximum, and that  $r$  of the  $p$ 's are 0 and  $s$  of them are 1. Let  $a$  satisfy  $(n-r-s)a + s = np$  and assume that  $0 < p_{n-1} < a < p_n < 1$ . As in part (a) we can write  $Eg(X) = \alpha A + \beta B + \gamma C$  for some coefficients  $\alpha, \beta, \gamma$ .

If  $\alpha - 2\beta + \gamma > 0$ , the transformation in (a) (but with  $a$  in place of  $p$ ) would increase  $Eg(X)$ . And if  $\alpha - 2\beta + \gamma < 0$ , we could increase it with a similar transformation, using  $\delta = -\min(p_{n-1}, 1 - p_n)$ . Therefore  $\alpha - 2\beta + \gamma = 0$ ; and we can repeat the transformation of (a) until every  $p_j$  is 0, 1, or  $a$ .

(c) Since  $\sum_{k=0}^m \binom{n}{k} = 0$  when  $s > m$ , we may assume that  $s \leq m$ , hence  $r+s < n$ . For this function  $g(k) = [0 \leq k \leq m]$  we have  $\alpha - 2\beta + \gamma = \binom{n-2}{m} - \binom{n-2}{m-1}$ . This difference cannot be positive if the choice of  $\{p_1, \dots, p_n\}$  is optimum; in particular we cannot have  $s = m$ . If  $r > 0$  we can make  $p_{n-1} = 0$  and  $p_n = a$ , so that  $\binom{n-2}{m} = \binom{n-r-s-1}{m-s} a^{m-s} (1-a)^{n-r-1-m}$  and  $\binom{n-2}{m-1} = \binom{n-r-s-1}{m-1-s} a^{m-1-s} (1-a)^{n-r-m}$ . But then the ratio  $\binom{n-2}{m} / \binom{n-2}{m-1} = (n-r-m)a / ((m-s)(1-a))$  exceeds 1; hence  $r = 0$ .

Similarly if  $s > 0$  we can set  $(p_{n-1}, p_n) = (a, 1)$ , getting the ratio  $\binom{n-2}{m} / \binom{n-2}{m-1} = (n-1-m)a / ((m-s+1)(1-a)) \geq 1$ . In this case  $\binom{n-2}{m} = \binom{n-2}{m-1}$  if and only if  $np = m+1$ ; we can transform without changing  $Eg(X)$ , until  $s = 0$  and each  $p_j = p$ .

[Reference: *Annals of Mathematical Statistics* **27** (1956), 713–721. The coefficients  $\binom{n}{k}$  also have many other important properties; see exercise 7.2.1.5–63, and the survey by J. Pitman in *J. Combinatorial Theory* **A77** (1997), 279–303.]

**29.** The result is obvious when  $m = 0$  or  $n$ ; and there's a direct proof when  $m = n-1$ :  $B_{n-1,n}(p) = 1 - p^n \geq (1-p)n / ((1-p)n + p)$  because  $p - np^n + (n-1)p^{n+1} = p(1-p)(1+p+\dots+p^{n-1} - p^{n-1}n) \geq 0$ . The result is also clear when  $p = 0$  or 1.

If  $p = (m + 1)/n$  we have  $R_{m,n}(p) = ((1 - p)(m + 1)/((1 - p)m + 1))^{n-m} = ((n - m - 1)/(n - m))^{n-m}$ . So if  $m > 0$  and  $\hat{p} = m/(n - 1)$ , we can apply exercise 28(c) with  $p_1 = \dots = p_{n-1} = \hat{p}$  and  $p_n = 1$ :

$$B_{m,n}(p) \geq \sum_{k=0}^m \binom{n}{k} = \sum_{k=0}^m \binom{n-1}{k-1} \hat{p}^{k-1} (1 - \hat{p})^{n-k} = B_{m-1,n-1}(\hat{p}).$$

When  $1 \leq m < n - 1$ , let  $Q_{m,n}(p) = B_{m,n}(p) - R_{m,n}(p)$ . The derivative

$$Q'_{m,n}(p) = (n - m) \binom{n}{m} (1 - p)^{n-m-1} (A - F(p)) / ((1 - p)m + 1)^{n-m+1},$$

where  $A = (m + 1)^{n-m} / \binom{n}{m} > 1$  and  $F(p) = p^m ((1 - p)m + 1)^{n-m+1}$ , begins positive at  $p = 0$ , eventually becomes negative but then is positive again at  $p = 1$ . (Notice that  $F(0) = 0$ , and  $F(p)$  increases dramatically until  $p = (m + 1)/(n + 1)$ ; then it decreases to  $F(1) = 1$ .) The facts that  $Q_{m,n}(\frac{m+1}{n}) \geq 0 = Q_{m,n}(0) = Q_{m,n}(1)$  now complete the proof, because  $Q'_{m,n}(p)$  changes sign only once in  $[0, \frac{m+1}{n}]$ . [*Annals of Mathematical Statistics* **36** (1965), 1272–1278.]

**30.** (a)  $\Pr(X_k = 0) = n/(n + 1)$ ; hence  $p = n^n/(n + 1)^n > 1/e \approx 0.368$ .

(b) (Solution by J. H. Elton.) Let  $p_{km} = \Pr(X_k = m)$ . Assume that these probabilities are fixed for  $1 \leq k < n$ , and let  $x_m = p_{nm}$ . Then  $x_0 = x_2 + 2x_3 + 3x_4 + \dots$ ; we want to minimize  $p = \sum_{m=1}^{\infty} (A_m + (m - 1)A_0)x_m$  in nonnegative variables  $x_1, x_2, \dots$ , where  $A_m = \Pr(X_1 + \dots + X_{n-1} \leq n - m)$ , subject to the condition  $\sum_{m=1}^{\infty} mx_m = 1$ . Since all coefficients of  $p$  are nonnegative, the minimum is achieved when all  $x_m$  for  $m \geq 1$  are zero except for one value  $m = m_n$ , which minimizes  $(A_m + (m - 1)A_0)/m$ . And  $m_n \leq n + 1$ , because  $A_m = 0$  whenever  $m > n$ . Similarly  $m_1, \dots, m_{n-1}$  also exist.

(c) (Solution by E. Schulte-Geers.) Letting  $m_1 = \dots = m_n = t \leq n + 1$ , we want to minimize  $B_{\lfloor n/t \rfloor, n}(1/t)$ . The inequality of Samuels in exercise 29 implies that

$$B_{m,n}(p) \geq \left(1 - \frac{1}{f(m, n, p) + 1}\right)^n \text{ for } p \leq \frac{m + 1}{n}, \text{ where } f(m, n, p) = \frac{(m + 1)(1 - p)n}{(n - m)p},$$

because we can set  $x = ((1 - p)m + 1)/((1 - p)m + 1)$  in the arithmetic–geometric mean inequality  $x^{n-m} \leq ((n - m)x + m)^n/n^n$ . Now  $1/t \leq (\lfloor n/t \rfloor + 1)/(n + 1)$  and  $f(\lfloor n/t \rfloor, n, 1/t) \geq n$ ; hence  $B_{\lfloor n/t \rfloor, n}(1/t) \geq n^n/(n + 1)^n$ .

[Peter Winkler called this the “gumball machine problem” in *CACM* **52**, 8 (August 2009), 104–105. J. H. Elton has verified that the joint distributions in (a) are optimum when  $n \leq 20$ ; see [arXiv:0908.3528](https://arxiv.org/abs/0908.3528) [math.PR] (2009), 7 pages. Do those distributions in fact minimize  $p$  for all  $n$ ? Uriel Feige has conjectured more generally that we have  $\Pr(X_1 + \dots + X_n < n + 1/(e - 1)) \geq 1/e$  whenever  $X_1, \dots, X_n$  are independent nonnegative random variables with  $E X_k \leq 1$ ; see *SICOMP* **35** (2006), 964–984.]

**31.** This result is immediate because  $\Pr(f([A_1], \dots, [A_n])) = E f([A_1], \dots, [A_n])$ . But a more detailed, lower-level proof will be helpful with respect to exercise 32.

Suppose, for example, that  $n = 4$ . The reliability polynomial is the sum of the reliability polynomials for the minterms of  $f$ ; so it suffices to show that the result is true for functions like  $x_1 \wedge \bar{x}_2 \wedge \bar{x}_3 \wedge x_4 = x_1(1 - x_2)(1 - x_3)x_4$ . And it's clear that  $\Pr(A_1 \cap \bar{A}_2 \cap \bar{A}_3 \cap A_4) = \Pr(A_1 \cap \bar{A}_2 \cap A_4) - \Pr(A_1 \cap \bar{A}_2 \cap A_3 \cap A_4) = \pi_{14} - \pi_{124} - \pi_{134} + \pi_{1234}$ . (See exercise 7.1.1–12; also recall the inclusion–exclusion principle.)

**32.** The  $2^n$  minterm probabilities in the previous answer must all be nonnegative, and they must sum to 1. We've already stipulated that  $\pi_\emptyset = 1$ , so the sum-to-1 condition is automatically satisfied. (The condition stated in the exercise when  $I \subseteq J$  is necessary but not sufficient; for example,  $\pi_{12}$  must be  $\geq \pi_1 + \pi_2 - 1$ .)

**33.** The three events  $(X, Y) = (1, 0), (0, 1), (1, 1)$  occur with probabilities  $p, q, r$ , respectively. The value of  $E(X|Y)$  is  $1, r/(q+r), r/(q+r)$  in those cases. Hence the answer is  $pz + (q+r)z^{r/(q+r)}$ . (This example demonstrates why univariate generating functions are *not* used in the study of conditional random variables such as  $E(X|Y)$ . But we do have the simple formula  $E(X|Y=k) = ([z^k] \frac{\partial}{\partial w} G(1, z)) / ([z^k] G(1, z))$ .)

martingale  
set partitions  
generating function  
binomial distribution  
Grimmett  
Stirzaker  
concave

**34.** The right-hand side is

$$\begin{aligned} \sum_{\omega} E(X|Y) \Pr(\omega) &= \sum_{\omega} \Pr(\omega) \sum_{\omega'} X(\omega') \Pr(\omega') [Y(\omega') = Y(\omega)] / \Pr(Y = Y(\omega)) \\ &= \sum_{\omega} \Pr(\omega) \sum_{\omega'} X(\omega') \Pr(\omega') [Y(\omega') = Y(\omega)] / \Pr(Y = Y(\omega')) \\ &= \sum_{\omega'} X(\omega') \Pr(\omega') \sum_{\omega} \Pr(\omega) [Y(\omega) = Y(\omega')] / \Pr(Y = Y(\omega')). \end{aligned}$$

**35.** Part (b) is false. If, for instance,  $X$  and  $Y$  are independent random bits and  $Z = X$ , we have  $E(X|Y) = \frac{1}{2}$  and  $E(\frac{1}{2}|Z) = \frac{1}{2} \neq X = E(X|Z)$ . The correct formula instead of (b) is

$$E(E(X|Y, Z)|Z) = E(X|Z). \tag{*}$$

This is (12) in the probability spaces conditioned by  $Z$ , and it is the crucial identity that underlies exercise 91. Part (a) is true because it is the case  $Y = Z$  of (\*).

**36.** (a)  $f(X)$ ; (b)  $E(f(Y)g(X))$ , generalizing (12). Proof:  $E(f(Y)E(g(X)|Y)) = \sum_y f(y)E(g(X)|Y=y)\Pr(Y=y) = \sum_{x,y} f(y)g(x)\Pr(X=x, Y=y) = E(f(Y)g(X))$ .

**37.** If we're given the values of  $X_1, \dots, X_{k-1}$ , the value of  $X_k$  is equally likely to be any of the  $n+1-k$  values in  $\{1, \dots, n\} \setminus \{X_1, \dots, X_{k-1}\}$ . Hence its average value is  $(1 + \dots + n - X_1 - \dots - X_{k-1}) / (n+1-k)$ . We conclude that  $E(X_k | X_1, \dots, X_{k-1}) = (n(n+1)/2 - X_1 - \dots - X_{k-1}) / (n+1-k)$ . [Incidentally, the sequence  $Z_0, Z_1, \dots$ , defined by  $Z_j = (n+j)X_1 + (n+j-2)X_2 + \dots + (n-j)X_{j+1} - (j+1)n(n+1)/2$  for  $0 \leq j < n$  and  $Z_j = Z_{n-1}$  for  $j \geq n$ , is therefore a martingale.]

**38.** Let  $t_{m,n}$  be the number of restricted growth strings of length  $m+n$  that begin with  $01 \dots (m-1)$ . (This is the number of set partitions of  $\{1, \dots, m+n\}$  in which each of  $\{1, \dots, m\}$  appears in a different block.) The generating function  $\sum_{n \geq 0} t_{m,n} z^n / n!$  turns out to be  $\exp(e^z - 1 + mz)$ ; hence  $t_{m,n} = \sum_k \varpi_k \binom{n}{k} m^{n-k}$ .

Suppose  $M = \max(X_1, \dots, X_{k-1}) + 1$ . Then  $\Pr(X_k = j) = t_{M, n-k} / t_{M, n+1-k}$  for  $0 \leq j < M$ , and  $t_{M+1, n-k} / t_{M, n+1-k}$  for  $j = M$ . Hence  $E(X_k | X_0, \dots, X_{k-1}) = ((\binom{M}{2} t_{M, n-k} + M t_{M+1, n-k}) / t_{M, n+1-k})$ .

**39.** (a) Since  $E(K|N=n) = pn$  we have  $E(K|N) = pN$ .

(b) Hence  $E K = E(E(K|N)) = E pN = p\mu$ .

(c) Let  $p_{nk} = \Pr(N=n, K=k) = (e^{-\mu} \mu^n / n!) \times \binom{n}{k} p^k (1-p)^{n-k} = (e^{-\mu} \mu^k p^k / k!) \times f(n-k)$ , where  $f(n) = (1-p)^n \mu^n / n!$ . Then  $E(N|K=k) = \sum_n n p_{nk} / \sum_n p_{nk}$ . Since  $n f(n-k) = k f(n-k) + (n-k) f(n-k)$  and  $n f(n) = (1-p) \mu f(n-1)$ , the answer is  $k + (1-p)\mu$ ; hence  $E(N|K) = K + (1-p)\mu$ . [G. Grimmett and D. Stirzaker, *Probability and Random Processes* (Oxford: 1982), §3.7.]

**40.** If  $p = \Pr(X > m)$ , clearly  $E X \leq (1-p)m + pM$ . [We also get this result from (15), by taking  $S = \{x | x \leq m\}$ ,  $f(x) = M - x$ ,  $s = M - m$ .]

**41.** (a) Convex when  $a \geq 1$  or  $a = 0$ ; otherwise neither convex nor concave. (However,  $x^a$  is concave when  $0 < a < 1$  and convex when  $a < 0$ , if we consider only positive values of  $x$ .) (b) Convex when  $n$  is even or  $n = 1$ ; otherwise neither convex nor concave.

(This function is  $\int_0^x t^{n-1} e^{x-t} dt / (n-1)!$ , according to 1.2.11.3-(5); so  $f''(x)/x > 0$  when  $n \geq 3$  is odd.) (c) Convex. (In fact  $f(|x|)$  is convex whenever  $f(z)$  has a power series with nonnegative coefficients, convergent for all  $z$ .) (d) Convex, provided of course that we allow  $f$  to be infinite in the definition (19).

**42.** We can show by induction on  $n$  that  $f(p_1 x_1 + \cdots + p_n x_n) \leq p_1 f(x_1) + \cdots + p_n f(x_n)$ , when  $p_1, \dots, p_n \geq 0$  and  $p_1 + \cdots + p_n = 1$ , as in exercise 6.2.2-36. The general result follows by taking limits as  $n \rightarrow \infty$ . [The quantity  $p_1 x_1 + \cdots + p_n x_n$  is called a “convex combination” of  $\{x_1, \dots, x_n\}$ ; similarly,  $EX$  is a convex combination of  $X$  values. Jensen actually began his study by assuming only the case  $p = q = \frac{1}{2}$  of (19).]

**43.**  $f(EX) = f(E(E(X|Y))) \leq E(f(E(X|Y))) \leq E(Ef(X|Y)) = Ef(X)$ . [S. M. Ross, *Probability Models for Computer Science* (2002), Lemma 3.2.1.]

**44.** The function  $f(xy)$  is convex in  $y$  for any fixed  $x$ . Therefore  $g(y) = Ef(Xy)$  is convex in  $y$ : It's a convex combination of convex functions. Also  $g(y) \geq f(EXy) = f(0) = g(0)$  by (20). Hence  $0 \leq a \leq b$  implies  $g(0) \leq g(a) \leq g(b)$  by convexity of  $g$ . [S. Boyd and L. Vandenberghe, *Convex Optimization* (2004), exercise 3.10.]

**45.**  $\Pr(X > 0) = \Pr(|X| \geq 1)$ ; set  $m = 1$  in (16).

**46.**  $EX^2 \geq (EX)^2$  in *any* probability distribution, by Jensen's inequality, because squaring is convex. We can also prove it directly, since  $EX^2 - (EX)^2 = E(X - EX)^2$ .

**47.** We always have  $Y \geq X$  and  $Y^2 \leq X^2$ . (Consequently (22) yields  $\Pr(X > 0) = \Pr(Y > 0) \geq (EY)^2 / (EY^2) \geq (EX)^2 / (EX^2)$  when  $EX \geq 0$ .)

**48.**  $\Pr(a - X_1 - \cdots - X_n > 0) \geq a^2 / (a^2 + \sigma_1^2 + \cdots + \sigma_n^2)$ , by exercise 47. [This inequality was *also* known to Chebyshev; see *J. Math. Pures et Appl.* (2) **19** (1874), 157-160. In the special case  $n = 1$  it is equivalent to “Cantelli's inequality,”

$$\Pr(X \geq EX + a) \leq \text{var}(X) / (\text{var}(X) + a^2), \quad \text{for } a \geq 0;$$

see *Atti del Congresso Internazionale dei Matematici* **6** (Bologna: 1928), 47-59, §6-§7.]

**49.**  $\Pr(X = 0) = 1 - \Pr(X > 0) \leq (EX^2 - (EX)^2) / EX^2 \leq (EX^2 - (EX)^2) / (EX)^2 = (EX^2) / (EX)^2 - 1$ . [Some authors call *this* inequality the “second moment principle,” but it is strictly weaker than (22).]

**50.** (a) Let  $Y_j = X_j/X$  if  $X_j > 0$ , otherwise  $Y_j = 0$ . Then  $Y_1 + \cdots + Y_m = [X > 0]$ . Hence  $\Pr(X > 0) = \sum_{j=1}^m EY_j$ ; and  $EY_j = E(X_j/X | X_j > 0) \cdot \Pr(X_j > 0)$ . [This identity, which requires only that  $X_j \geq 0$ , is elementary yet nonlinear, so it apparently lay undiscovered for many years. See D. Aldous, *Discrete Math.* **76** (1989), 168.]

(b) Since  $X_j \in \{0, 1\}$ , we have  $\Pr(X_j > 0) = EX_j = p_j$ ; and  $E(X_j/X | X_j > 0) = E(X_j/X | X_j = 1) = E(1/X | X_j = 1) \geq 1/E(X | X_j = 1)$ .

(c)  $\Pr(X_J = 1) = \sum_{j=1}^m \Pr(J = j \text{ and } X_j = 1) = \sum_{j=1}^m p_j/m = EX/m$ . Hence  $\Pr(J = j | X_J = 1) = \Pr(J = j \text{ and } X_j = 1) / \Pr(X_J = 1) = (p_j/m) / (EX/m) = p_j/EX$ .

(d) Since  $J$  is independent we have  $t_j = E(X | J = j \text{ and } X_j = 1) = E(X | X_j = 1)$ .

(e) The right side is  $(EX) \sum_{j=1}^m (p_j/EX) / t_j \geq (EX) / \sum_{j=1}^m (p_j/EX) t_j$ .

**51.** If  $g(q_1, \dots, q_m) = 1 - f(p_1, \dots, p_m)$  is the dual of  $f$ , where  $q_j = 1 - p_j$ , a lower bound on  $g$  gives an upper bound on  $f$ . For example, when  $f$  is  $x_1 x_2 x_3 \vee x_2 x_3 x_4 \vee x_4 x_5$ ,  $\bar{f}$  is  $\bar{x}_1 \bar{x}_4 + \bar{x}_2 \bar{x}_4 + \bar{x}_3 \bar{x}_4 + \bar{x}_2 \bar{x}_5 + \bar{x}_3 \bar{x}_5$ . So the inequality (24) gives  $g(q_1, \dots, q_5) \geq q_1 q_4 / (1 + q_2 + q_3 + q_2 q_5 + q_3 q_5) + q_2 q_4 / (q_1 + 1 + q_3 + q_5 + q_3 q_5) + q_3 q_4 / (q_1 + q_2 + 1 + q_2 q_5 + q_5) + q_2 q_5 / (q_1 q_4 + q_4 + q_3 q_4 + 1 + q_3) + q_3 q_5 / (q_1 q_4 + q_2 q_4 + q_4 + q_2 + 1)$ . In particular,  $g(.1, \dots, .1) > 0.039$  and  $f(.9, \dots, .9) < 0.961$ .

**52.**  $\binom{n}{k} p^k / \sum_{j=0}^k \binom{k}{j} \binom{n-k}{j} p^j$ .

power series  
convex combination  
Jensen  
Ross  
Boyd  
Vandenberghe  
Jensen's inequality  
convex  
Chebyshev  
Cantelli's inequality  
second moment principle  
Aldous  
dual of a Boolean function



**53.**  $f(p_1, \dots, p_6) \geq p_1 p_2 (1-p_3) / (1+p_4 p_5 (1-p_6)) + \dots + p_6 p_1 (1-p_2) / (1+p_3 p_4 (1-p_5))$ . Monotonicity is not required when applying this method, nor need the implicants be prime. The result is exact when the implicants are disjoint.

**54.** (a)  $\Pr(X > 0) \leq EX = \binom{n}{3} p^3$ , because  $EX_{uvw} = p^3$  for all  $u < v < w$ .

(b)  $\Pr(X > 0) \geq (EX)^2 / (EX^2)$ , where the numerator is the square of (a) and the denominator can be shown to be  $\binom{n}{3} p^3 + 12 \binom{n}{4} p^5 + 30 \binom{n}{5} p^6 + 20 \binom{n}{6} p^6$ . For example, the expansion of  $X^2$  contains 12 terms of the form  $X_{uvw} X_{uvw'}$  with  $u < v < w < w'$ , and each of those terms has expected value  $p^5$ .

**55.** A BDD for the corresponding Boolean function of  $\binom{10}{2} = 45$  variables has about 1.4 million nodes, and allows us to evaluate the true probability  $(1-p)^{45} G(p/(1-p))$  exactly, where  $G(z)$  is the corresponding generating function (see exercise 7.1.4-25). The results are: (a)  $30/37 \approx .811 < 35165158461687/2^{45} \approx .999 < 15$ ; (b)  $10/109 \approx .092 < 4180246784470862526910349589019919032987399/(4 \times 10^{43}) \approx .105 < .12$ .

**56.** The upper bound is  $\mu = \lambda^3/6$ ; the lower bound divides this by  $1 + \mu$ . [The exact asymptotic value can be obtained using the principle of inclusion and exclusion and its “bracketing” property, as in Eq. 7.2.1.4-(48); the result is  $1 - e^{-\mu}$ . See P. Erdős and A. Rényi, *Magyar Tudományos Akadémia Mat. Kut. Int. Közl.* **5** (1960), 17–61, §3.]

**57.** To compute  $E(X | X_{uvw} = 1)$  we sum  $\Pr(X_{u'v'w'} | X_{uvw} = 1)$  over all  $\binom{n}{3}$  choices of  $u' < v' < w'$ . If  $\{u', v', w'\} \cap \{u, v, w\}$  has  $t$  elements, this probability is  $p^{3-t(t-1)/2}$ ; and there are  $\binom{3}{t} \binom{n-3}{3-t}$  such cases. Consequently we get

$$\Pr(X > 0) \geq \binom{n}{3} p^3 / ((\binom{n-3}{3}) p^3 + 3 \binom{n-3}{2} p^3 + 3 \binom{n-3}{1} p^2 + \binom{n-3}{0} p^0).$$

[In this problem the lower bound turns out to be the same using either inequality; but the derivation here was easier.]

**58.**  $\Pr(X > 0) \leq \binom{n}{k} p^{k(k-1)/2}$ . The lower bound, using the conditional expectation inequality as in the previous answer, divides this by  $\sum_{t=0}^k \binom{k}{t} \binom{n-k}{k-t} p^{k(k-1)/2-t(t-1)/2}$ .

**59.** (a) The hypotheses imply that  $a_0 a_1 b_0 b_1 \leq c_0 c_1 d_0 d_1$ . The key observation is that

$$\begin{aligned} c_1 d_0 ((c_0 + c_1)(d_0 + d_1) - (a_0 + a_1)(b_0 + b_1)) = \\ c_1 d_0 (c_0 d_0 - a_0 b_0 + c_1 d_1 - a_1 b_1) + (c_1 d_0 - a_0 b_1)(c_1 d_0 - a_1 b_0) + c_0 c_1 d_0 d_1 - a_0 a_1 b_0 b_1. \end{aligned}$$

Thus the result holds when  $c_1 d_0 \neq 0$ . If  $c_1 = 0$  we have  $a_0 b_0 + a_0 b_1 + a_1 b_0 + a_1 b_1 = a_0 b_0 \leq c_0 d_0 \leq c_0 (d_0 + d_1)$ . And a similar argument applies to the case  $d_0 = 0$ .

All four hypotheses hold with equality when  $a_0 = b_0 = d_0 = 0$  and the other variables are 1, yet the conclusion is that  $1 \leq 2$ . Conversely, when  $b_1 = c_1 = 2$  and the other variables are 1, we have  $a_1 b_0 < c_1 d_0$  but conclude only that  $6 \leq 6$ .

(b) Let  $A_l = \sum \{a_{2j+l} \mid 0 \leq j < 2^{n-1}\}$  for  $l = 0$  and  $l = 1$ , and define  $B_l, C_l, D_l$  similarly from  $b_{2j+l}, c_{2j+l}, d_{2j+l}$ . The hypotheses for  $j \bmod 2 = l$  and  $k \bmod 2 = m$  prove that  $A_l B_m \leq C_{l|m} D_{l \& m}$ , by induction on  $n$ . Hence, by part (a), we have the desired inequality  $(A_0 + A_1)(B_0 + B_1) \leq (C_0 + C_1)(D_0 + D_1)$ . [This result is due to R. Ahlswede and D. E. Daykin, *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* **43** (1978), 183–185, who stated it in the language of the next exercise.]

(c) Now let  $A_n = a_0 + \dots + a_{2^n-1}$ , and define  $B_n, C_n, D_n$  similarly. If  $A_\infty B_\infty > C_\infty D_\infty$ , we'll have  $A_n B_n > C_n D_n$  for some  $n$ . But  $C_\infty D_\infty \geq C_n D_n$ , contra (b).

**60.** (a) We can consider each set to be a subset of the nonnegative integers. Let  $\bar{\alpha}(S) = \alpha(S)[S \in \mathcal{F}]$ ,  $\bar{\beta}(S) = \beta(S)[S \in \mathcal{G}]$ ,  $\bar{\gamma}(S) = \gamma(S)[S \in \mathcal{F} \sqcup \mathcal{G}]$ ,  $\bar{\delta}(S) = \delta(S)[S \in \mathcal{F} \cap \mathcal{G}]$ ; then  $\bar{\alpha}(\wp) = \alpha(\mathcal{F})$ ,  $\bar{\beta}(\wp) = \beta(\mathcal{G})$ ,  $\bar{\gamma}(\wp) = \gamma(\mathcal{F} \sqcup \mathcal{G})$ , and  $\bar{\delta}(\wp) = \delta(\mathcal{F} \cap \mathcal{G})$ , where  $\wp$  is the

prime implicants  
BDD  
Boolean function  
generating function  
inclusion and exclusion  
bracketing  
enveloping series  
Erdős  
Rényi  
conditional expectation inequality  
Ahlswede  
Daykin  
 $\wp$

family of all possible subsets. Since any set  $S$  of nonnegative integers can be encoded in the usual way as the binary number  $s = \sum_{j \in S} 2^j$ , the desired result follows from the four functions theorem if we let  $a_s = \bar{\alpha}(S)$ ,  $b_s = \bar{\beta}(S)$ ,  $c_s = \bar{\gamma}(S)$ ,  $d_s = \bar{\delta}(S)$ .

(b) Let  $\alpha(S) = \beta(S) = \gamma(S) = \delta(S) = 1$  for all sets  $S$ .

**61.** (a) In the hinted case we can let  $\alpha(S) = f(S)\mu(S)$ ,  $\beta(S) = g(S)\mu(S)$ ,  $\gamma(S) = f(S)g(S)\mu(S)$ ,  $\delta(S) = \mu(S)$ ; the four functions theorem yields the result. The general case follows because we have  $E(fg) - E(f)E(g) = E(\hat{f}\hat{g}) - E(\hat{f})E(\hat{g})$ , where  $\hat{f}(S) = f(S) - f(\emptyset)$  and  $\hat{g}(S) = g(S) - g(\emptyset)$ . [See *Commun. Math. Physics* **22** (1971), 89–103.]

(b) Changing  $f(S)$  to  $\theta f(S)$  and  $g(S)$  to  $\phi g(S)$  changes  $E(fg) - E(f)E(g)$  to  $\theta\phi(E(fg) - E(f)E(g))$ , for all real numbers  $\theta$  and  $\phi$ .

(c) If  $S$  and  $T$  are supported, then  $R = S \cap T$  and  $U = S \cup T$  are supported. Furthermore we can write  $S = R \cup \{s_1, \dots, s_k\}$  and  $T = R \cup \{t_1, \dots, t_l\}$  where the sets  $S_i = R \cup \{s_1, \dots, s_i\}$  and  $T_j = R \cup \{t_1, \dots, t_j\}$  are supported, as are their unions  $U_{i,j} = S_i \cup T_j$ , for  $0 \leq i \leq k$  and  $0 \leq j \leq l$ . By (iii) we know that  $\mu(U_{i+1,j})/\mu(U_{i,j}) \leq \mu(U_{i+1,j+1})/\mu(U_{i,j+1})$  when  $0 \leq i < k$  and  $0 \leq j < l$ . Multiplying these inequalities for  $0 \leq i < k$ , we obtain  $\mu(U_{k,j})/\mu(U_{0,j}) \leq \mu(U_{k,j+1})/\mu(U_{0,j+1})$ . Hence  $\mu(S)/\mu(R) = \mu(U_{k,0})/\mu(U_{0,0}) \leq \mu(U_{k,l})/\mu(U_{0,l}) = \mu(U)/\mu(T)$ .

(d) In fact, equality holds, because  $[j \in S] + [j \in T] = [j \in S \cup T] + [j \in S \cap T]$ .

[*Note:* Random variables with this distribution are often confusingly called “Poisson trials,” a term that conflicts with the (quite different) Poisson distribution of exercise 39.]

(e) Choose  $c$  in the following examples so that  $\sum_S \mu(S) = 1$ . In each case the supported sets are subsets of  $U = \{1, \dots, m\}$ . (i) Let  $\mu(S) = cr_1r_2 \dots r_{|S|}$ , where  $0 < r_1 \leq \dots \leq r_m$ . (ii) Let  $\mu(S) = cp_j$  when  $S = \{1, \dots, j\}$  and  $1 \leq j \leq m$ , otherwise  $\mu(S) = 0$ . (If  $p_1 = \dots = p_m$  in this case, the FKG inequality reduces to Chebyshev’s monotonic inequality of exercise 1.2.3–31.) (iii) Let

$$\mu(S) = c\mu_1(S \cap U_1)\mu_2(S \cap U_2) \dots \mu_k(S \cap U_k),$$

where each  $\mu_j$  is a distribution on the subsets of  $U_j \subseteq U$  that satisfies (\*\*). The subuniverses  $U_1, \dots, U_k$  needn’t be disjoint. (iv) Let  $\mu(S) = ce^{-f(S)}$ , where  $f$  is a submodular set function on the supported subsets of  $U$ :  $f(S \cup T) + f(S \cap T) \leq f(S) + f(T)$  whenever  $f(S)$  and  $f(T)$  are defined. (See Section 7.6.)

**62.** A Boolean function is essentially a set function whose values are 0 or 1. In general, under the Bernoulli distribution or any other distribution that satisfies the condition of exercise 61, the FKG inequality implies that any monotone increasing Boolean function is positively correlated with any other monotone increasing Boolean function, but negatively correlated with any monotone *decreasing* Boolean function. In this case,  $f$  is monotone increasing but  $g$  is monotone decreasing: Adding an edge doesn’t disconnect a graph; deleting an edge doesn’t invalidate a 4-coloring.

(Notice that when  $f$  is a Boolean function,  $E f$  is the probability that  $f$  is true under the given distribution. The fact that  $\text{covar}(f, g) \leq 0$  in such a case is equivalent to saying that the conditional probability  $\Pr(f | g)$  is  $\leq \Pr(f)$ .)

**63.** If  $\omega$  is the event  $(Z_0 = a, Z_1 = b)$ , we have  $Z_0(\omega) = a$  and  $E(Z_1 | Z_0)(\omega) = (p_{a1} + 2p_{a2})/(p_{a0} + p_{a1} + p_{a2})$ . Hence  $p_{01} = p_{02} = p_{20} = p_{21} = 0$ , and  $p_{10} = p_{12}$ ; these conditions are necessary and sufficient for  $E(Z_1 | Z_0) = Z_0$ .

**64.** (a) No. Consider the probability space consisting of just three events  $(Z_0, Z_1, Z_2) = (0, 0, -2), (1, 0, 2), (1, 2, 2)$ , each with probability  $1/3$ . Call those events  $a, b, c$ . Then  $E(Z_1 | Z_0)(a) = 0 = Z_0(a)$ ;  $E(Z_1 | Z_0)(b, c) = \frac{1}{2}(0 + 2) = Z_0(b, c)$ ;  $E(Z_2 | Z_1)(a, b) = \frac{1}{2}(-2 + 2) = Z_1(a, b)$ ;  $E(Z_2 | Z_1)(c) = 2 = Z_1(c)$ . But  $E(Z_2 | Z_0, Z_1)(a) = -2 \neq Z_1(a)$ .

sets, represented as integers  
Poisson trials  
Poisson distribution  
Chebyshev’s monotonic inequality  
submodular set function  
FKG  
Boolean function  
monotone  
conditional probability  
covariance  
conditional prob

(b) Yes. We have  $\sum_{z_{n+1}} (z_{n+1} - z_n) \Pr(Z_0 = z_0, \dots, Z_{n+1} = z_{n+1}) = 0$  for all fixed  $(z_0, \dots, z_n)$ . Sum these to get  $\sum_{z_{n+1}} (z_{n+1} - z_n) \Pr(Z_n = z_n, Z_{n+1} = z_{n+1}) = 0$ .

**65.** Observe first that  $E(Z_{n+1} | Z_0, \dots, Z_k) = E(E(Z_{n+1} | Z_0, \dots, Z_n) | Z_0, \dots, Z_k) = E(Z_n | Z_0, \dots, Z_k)$  whenever  $k < n$ . Thus  $E(Z_{m(n+1)} | Z_0, \dots, Z_{m(n)}) = Z_{m(n)}$  for all  $n \geq 0$ . Hence  $E(Z_{m(n+1)} | Z_{m(0)}, \dots, Z_{m(n)}) = Z_{m(n)}$ , as in the previous exercise.

**66.** We need to specify the joint distribution of  $\{Z_0, \dots, Z_n\}$ , and it's not difficult to see that there is only one solution. Let  $p(\sigma_1, \dots, \sigma_n) = \Pr(Z_1 = \sigma_1, \dots, Z_n = \sigma_n)$  when  $\sigma_1, \dots, \sigma_n$  are each  $\pm 1$ . The martingale law  $p(\sigma_1 \dots \sigma_n \bar{1})(n+1) - p(\sigma_1 \dots \sigma_n \bar{1})(n) = \sigma_n p(\sigma_1 \dots \sigma_n) n = \sigma_n (p(\sigma_1 \dots \sigma_n \bar{1}) + p(\sigma_1 \dots \sigma_n \bar{1})) n$  gives  $p(\sigma_1 \dots \sigma_{n+1})/p(\sigma_1 \dots \sigma_n) = (1 + 2n[\sigma_n \sigma_{n+1} > 0])/(2n + 2)$ . Hence we find that  $\Pr(Z_1 = z_1, \dots, Z_n = z_n) = (\prod_{k=1}^{n-1} (1 + 2k[z_k z_{k+1} > 0]))/(2^n n!)$ . When  $n = 3$ , for example, the eight possible cases  $z_1 z_2 z_3 = 123, 1\bar{2}\bar{3}, \dots, \bar{1}\bar{2}\bar{3}$  occur with probabilities  $(15, 3, 1, 5, 5, 1, 3, 15)/48$ .

**67.** (a) You “always” (with probability 1) make  $2^{n+1} - (1 + 2 + \dots + 2^n) = 1$  dollar.

(b) Your total payments are  $X = X_0 + X_1 + \dots$  dollars, where  $X_n = 2^n$  with probability  $2^{-n}$ , otherwise  $X_n = 0$ . So  $E X_n = 1$ , and  $E X = E X_0 + E X_1 + \dots = \infty$ .

(c) Let  $\langle T_n \rangle$  be a sequence of uniformly random bits; and define the fair sequence  $Y_n = (-1)^{T_n} 2^n T_0 \dots T_{n-1}$ , or  $Y_n = 0$  if there is no  $n$ th bet. Then  $Z_n = Y_0 + \dots + Y_n$ .

[The famous adventurer Casanova lost a fortune in 1754 using this strategy, which he called “the martingale” in his autobiography *Histoire de ma vie*. A similar betting scheme had been proposed by Nicolas Bernoulli (see P. R. de Montmort, *Essay d'Analyse sur les Jeux de Hazard*, second edition (1713), page 402); and the perplexities of (a) and (b) were studied by his cousin Daniel Bernoulli, whose important paper in *Commentarii Academiæ Scientiarum Imperialis Petropolitanæ* **5** (1731), 175–192, has caused this scenario to become known as the St. Petersburg paradox.]

**68.** (a) Now  $Z_n = Y_1 + \dots + Y_n$ , where  $Y_n = (-1)^{T_n} [N \geq n]$ . Again  $\Pr(Z_N = 1) = 1$ .

(b) The generating function  $g(z)$  equals  $z(1 + g(z)^2)/2$ , since he must win \$2 if the first bet loses. Hence  $g(z) = (1 - \sqrt{1 - z^2})/z$ ; and the desired probability is  $[z^n] g(z) = C_{(n-1)/2} [n \text{ odd}]/2^n$ , where  $C_k$  is the Catalan number  $\binom{2k}{k}/(k+1)$ .

(c)  $\Pr(N \geq n) = [z^n] (1 - zg(z))/(1 - z) = [z^n] (1 + z)/\sqrt{1 - z^2} = \binom{2[n/2]}{[n/2]}/2^{[n/2]}$ .

(d)  $EN = g'(1) = \infty$ . (It's also  $\sum_{n=1}^{\infty} \Pr(N \geq n)$ , where  $\Pr(N \geq n) \sim 1/\sqrt{\pi n}$ .)

(e) Let  $p_m = \Pr(Z_n \geq -m)$  for all  $n \geq 0$ . Clearly  $p_0 = 1/2$  and  $p_m = (1 + p_{m-1}p_m)/2$  for  $m > 0$ ; this recurrence has the solution  $p_m = (m+1)/(m+2)$ . So the answer is  $1/((m+1)(m+2))$ ; it's another probability distribution with infinite mean.

(f) The generating function  $g_m(z)$  for the number of times  $-m$  is hit satisfies  $g_0(z) = z/(2-z)$ ,  $g_m(z) = (1 + g_{m-1}(z)g_m(z))/2$  for  $m > 0$ . So  $g_m(z) = h_m(z)/h_{m+1}(z)$  for  $m \geq 0$ , where  $h_m(z) = 2m - (2m-1)z$ , and  $g'_m(1) = 2$ . [A distribution with *finite* mean! See W. Feller, *An Intro. to Probability Theory* **2**, second edition (1971), XII.2.]

**69.** Each permutation of  $n$  elements corresponds to a configuration of  $n+1$  balls in the urn. For Method 1, the number of corresponding “red balls” is the *position* of element 1; for Method 2, it is the *value* in position 1. For example, we'd put 3 1 2 4 into node (2, 3) with respect to Method 1 but into (3, 2) with respect to Method 2. (In fact, Methods 1 and 2 construct permutations that are inverses of each other.)

**70.** Start with the permutation  $1 2 \dots (c-1)$  at the root, and use Method 1 of the previous exercise to generate all  $n!/(c-1)!$  permutations in which these elements retain that order. A permutation with  $j$  in position  $P_j$  for  $1 \leq j < c$  stands for  $P_j - P_{j-1}$  balls of color  $j$ , where  $P_0 = 0$  and  $P_c = n+1$ ; for example, if  $c = 3$ , the permutation

Casanova  
Bernoulli  
de Montmort  
Bernoulli  
St. Petersburg paradox  
paradox  
Catalan number  
recurrence  
infinite mean  
generating function  
Feller  
inverses

3142 would correspond to node  $(2, 2, 1)$ . The resulting tuples  $(A_1, \dots, A_c)/(n+1)$  then form a martingale for  $n = c, c+1, \dots$ , uniformly distributed (for each  $n$ ) among all  $\binom{n}{c-1}$  compositions of  $n+1$  into  $c$  positive parts.

[We can also use this setup to deal with Pólya's two-color model when there are  $r$  red balls and  $b$  black balls at the beginning: Imagine  $r+b$  colors, then identify the first  $r$  of them with red. This model was first studied by D. Blackwell and D. Kendall, *J. Applied Probability* **1** (1964), 284–296.]

**71.** If  $m = r' - r$  and  $n = b' - b$  we must move  $m$  times to the right and  $n$  times to the left; there are  $\binom{m+n}{n}$  such paths. Every path occurs with the same probability, because the numerators of the fractions are  $r \cdot (r+1) \cdot \dots \cdot (r'-1) \cdot b \cdot (b+1) \cdot \dots \cdot (b'-1) = r^{\overline{m}} b^{\overline{n}}$  in some order, and the denominators are  $(r+b) \cdot (r+b+1) \cdot \dots \cdot (r'+b'-1) = (r+b)^{\overline{m+n}}$ .

The answer,  $\binom{m+n}{n} r^{\overline{m}} b^{\overline{n}} / (r+b)^{\overline{m+n}}$ , reduces to  $1/(r'+b'-1)$  when  $r = b = 1$ .

**72.** Since all paths have the same probability, this expected value is the same as  $E(X_1 X_2 \dots X_m)$ , which is obviously  $1/(m+1)$ . (Thus the  $X$ 's are *very* highly correlated: This expected value would be  $1/2^m$  if they were independent. Notice that the probability of an event such as  $(X_2 = 1, X_5 = 0, X_6 = 1)$  is  $E(X_2(1-X_5)X_6) = 1/3 - 1/4$ .)

[The far-reaching ramifications of such exchangeable random variables are surveyed in O. Kallenberg's book *Probabilistic Symmetries and Invariance Principles* (2005).]

**73.**  $f(r, n) = r \binom{n+1}{r} \sum_k \binom{r-1}{k} (-1)^k q_{n+1-r+k}$ , where  $q_k = a_k/(k+1)$ , by induction on  $r$ .

**74.** Node  $(r, n+2-r)$  on level  $n$  is reached with probability  $\langle \binom{n}{r-1} \rangle / n!$ , proportional to an Eulerian number (see Section 5.1.3). (Indeed, we can associate the permutations of  $\{1, \dots, n+1\}$  that have exactly  $r$  runs with this node, using Method 1 as in exercise 69.)

*Reference: Communications on Pure and Applied Mathematics* **2** (1949), 59–70.

**75.** As before, let  $R_n = X_0 + \dots + X_n$  be the number of red balls at level  $n$ . Now we have  $E(X_{n+1} | X_0, \dots, X_n) = 1 - R_n/(n+2)$ . Hence  $E(R_{n+1} | R_n) = (n+1)R_n/(n+2) + 1$ , and the definition  $Z_n = (n+1)R_n - (n+2)(n+1)/2$  is a natural choice.

**76.** No. For example, let  $Z_0 = X$ ,  $Z'_0 = Y$ , and  $Z_1 = Z'_1 = X + Y$ , where  $X$  and  $Y$  are independent with  $E X = E Y = 0$ . Then  $E(Z_1 | Z_0) = Z_0$  and  $E(Z'_1 | Z'_0) = Z'_0$ , but  $E(Z_1 + Z'_1 | Z_0 + Z'_0) = 2(Z_0 + Z'_0)$ . (On the other hand, if  $\langle Z_n \rangle$  and  $\langle Z'_n \rangle$  are both martingales with respect to some common sequence  $\langle X_n \rangle$ , then  $\langle Z_n + Z'_n \rangle$  is also.)

**77.**  $E(Z_{n+1} | Z_0, \dots, Z_n) = E(E(Z_{n+1} | Z_0, \dots, Z_n, X_0, \dots, X_n) | Z_0, \dots, Z_n)$ , which equals  $E(E(Z_{n+1} | X_0, \dots, X_n) | Z_0, \dots, Z_n)$  because  $Z_n$  is a function of  $X_0, \dots, X_n$ ; and that equals  $E(Z_n | Z_0, \dots, Z_n) = Z_n$ . (Furthermore  $\langle Z_n \rangle$  is a martingale with respect to, say, a constant sequence. But not with respect to *every* sequence.)

A similar proof shows that any sequence  $\langle Y_n \rangle$  that is fair with respect to  $\langle X_n \rangle$  is also fair with respect to itself.

**78.**  $E(Z_{n+1} | V_0, \dots, V_n) = E(Z_n V_{n+1} | V_0, \dots, V_n) = Z_n$ .

The converse holds with  $V_0 = Z_0$  and  $V_n = Z_n/Z_{n-1}$  for  $n > 0$ , *provided* that  $Z_{n-1} = 0$  implies  $Z_n = 0$ , and that we define  $V_n = 1$  when that happens.

**79.**  $Z_n = V_0 V_1 \dots V_n$ , where  $V_0 = 1$  and each  $V_n$  for  $n > 0$  is independently equal to  $q/p$  (with probability  $p$ ) or to  $p/q$  (with probability  $q$ ). Since  $E(V_n) = q+p = 1$ ,  $\langle V_n \rangle$  is multiplicatively fair. [See A. de Moivre, *The Doctrine of Chances* (1718), 102–154.]

**80.** (a) True; in fact  $E(f_n(Y_0 \dots Y_{n-1})Y_n) = 0$  for any function  $f_n$ .

(b) False: For example, let  $Y_5 = \pm 1$  if  $Y_3 > 0$ , otherwise  $Y_5 = 0$ . (Hence permutations of a fair sequence needn't be fair. The statement is, however, true if the  $Y$ 's are *independent* with mean zero.)

compositions  
Blackwell  
Kendall  
correlated  
exchangeable random variables  
Kallenberg  
Eulerian number  
runs  
fair with respect to  
de Moivre  
independent

(c) False if  $n_1 = 0$  and  $m = 1$  (or if  $m = 0$ ); otherwise true. (Sequences that satisfy  $E((Y_{n_1} - E Y_{n_1}) \cdots (Y_{n_m} - E Y_{n_m})) = E(Y_{n_1} - E Y_{n_1}) \cdots E(Y_{n_m} - E Y_{n_m})$  are called *totally uncorrelated*. Such sequences, with  $E Y_n = 0$  for all  $n$ , are not always fair; but fair sequences are always totally uncorrelated.)

**81.** Assuming that  $X_0, \dots, X_n$  can be deduced from  $Z_0, \dots, Z_n$ , we have  $a_n X_n + b_n X_{n-1} = Z_n = E(Z_{n+1} | Z_0, \dots, Z_n) = E(a_{n+1} X_{n+1} + b_{n+1} X_n | X_0, \dots, X_n) = a_{n+1}(X_n + X_{n-1}) + b_{n+1} X_n$  for  $n \geq 1$ . Hence  $a_{n+1} = b_n$ ,  $b_{n+1} = a_n - a_{n+1} = b_{n-1} - b_n$ ; and we have  $a_n = F_{-n-1}$ ,  $b_n = F_{-n-2}$  by induction, verifying the assumption.

[See J. B. MacQueen, *Annals of Probability* **1** (1973), 263–271.]

**82.** (a)  $Z_n = A_n/C_n$ , where  $A_n = 4 - X_1 - \cdots - X_n$  is the number of aces and  $C_n$  is the number of cards remaining after you've seen  $n$  cards. Hence  $E Z_{n+1} = (A_n/C_n)(A_n - 1)/(C_n - 1) + (1 - A_n/C_n)A_n/(C_n - 1) = A_n/C_n$ . (In every generalization of Pólya's urn for which the  $n$ th step adds  $k_n$  balls of the chosen color, the ratio red/(red + black) is always a martingale, even when  $k_n$  is negative, as long as enough balls of the chosen color remain. This exercise represents the case  $k_n = -1$ .)

(b) This is the optional stopping principle in a bounded-time martingale.

(c)  $Z_N = A_N/C_N$  is the probability that an ace will be next. ["Ace Now" is a variant of R. Connelly's game "Say Red"; see *Pallbearers Review* **9** (1974), 702.]

**83.**  $Z_n = \sum_{k=1}^n (X_k - E X_k)$  is a martingale, for which we can study the bounded stopping rules  $\min(m, N)$  for any  $m$ . But Svante Janson suggests a direct computation, beginning with the formula  $S_n = \sum_{k=1}^n X_k [N \geq k]$  where  $N$  might be  $\infty$ : We have  $E(X_n [N \geq n]) = (E X_n)(E[N \geq n])$ , because  $[N \geq n]$  is a function of  $\{X_0, \dots, X_{n-1}\}$ , hence independent of  $X_n$ . And since  $X_n \geq 0$ , we have  $E S_N = \sum_{n=1}^{\infty} E(X_n [N \geq n]) = \sum_{n=1}^{\infty} (E X_n) E[N \geq n] = \sum_{n=1}^{\infty} E((E X_n)[N \geq n]) = E \sum_{n=1}^{\infty} (E X_n)[N \geq n]$ , which is  $E \sum_{n=1}^N E X_n$ . (The equation might be ' $\infty = \infty$ '.)

[Wald's original papers, in *Annals of Mathematical Statistics* **15** (1944), 283–296, **16** (1945), 287–293, solved a somewhat different problem and proved more.]

**84.** (a) We have  $f(Z_n) = f(E(Z_{n+1} | Z_0, \dots, Z_n)) \leq E(f(Z_{n+1}) | Z_0, \dots, Z_n)$  by Jensen's inequality. And the latter is  $E(f(Z_{n+1}) | f(Z_0), \dots, f(Z_n))$  as in answer 77. [Incidentally, D. Gilat has shown that every nonnegative submartingale is  $\langle |Z_n| \rangle$  for some martingale  $\langle Z_n \rangle$ ; see *Annals of Probability* **5** (1977), 475–481.]

(b) Again we get a submartingale, *provided* that we also have  $f(x) \leq f(y)$  for  $a \leq x \leq y \leq b$ . [J. L. Doob, *Stochastic Processes* (1953), 295–296.]

**85.** Since  $\langle B_n/(R_n + B_n) \rangle = \langle 1 - R_n/(R_n + B_n) \rangle$  is a martingale by (27), and since  $f(x) = 1/x$  is convex for positive  $x$ ,  $\langle (R_n + B_n)/B_n \rangle = \langle R_n/B_n + 1 \rangle$  is a submartingale by exercise 84. (A direct proof could also be given.)

**86.** The rule  $N_{n+1}(Z_0, \dots, Z_n) = [\max(Z_0, \dots, Z_n) < x \text{ and } n + 1 < m]$  is bounded. If  $\max(Z_0, \dots, Z_{m-1}) < x$  then we have  $Z_N < x$ , where  $N$  is defined by (31); similarly, if  $\max(Z_0, \dots, Z_{m-1}) \geq x$  then  $Z_N \geq x$ . Hence  $\Pr(\max(Z_0, \dots, Z_n) \geq x) = (E Z_N)/x$  by Markov's inequality; and  $E Z_N \leq E Z_n$  in a submartingale.

**87.** This is the probability that  $Z_n$  becomes  $3/4$ , which also is  $\Pr(\max(Z_0, \dots, Z_n) \geq 3/4)$ . But  $E Z_n = 1/2$  for all  $n$ , hence (33) tells us that it is at most  $(1/2)/(3/4) = 2/3$ .

(The exact value can be calculated as in the following exercise. It turns out to be  $\sum_{k=0}^{\infty} \frac{2}{(4k+2)(4k+3)} = \frac{1}{2}H_{3/4} - \frac{1}{2}H_{1/2} + \frac{1}{3} = \frac{1}{4}\pi - \frac{1}{2}\ln 2 \approx .439$ .)

**88.** (a) We have  $S > 1/2$  if and only if there comes a time when there are more red balls than black balls. Since that happens if and only if the process passes through one

totally uncorrelated  
uncorrelated  
Fibonacci numbers  
MacQueen  
Pólya's urn  
optional stopping principle  
Connelly  
Say Red  
Janson  
Wald  
Jensen's inequality  
Gilat  
nonnegative submartingale  
Doob  
Markov's inequality  
harmonic numbers, fractional

of the nodes  $(2, 1)$ ,  $(3, 2)$ ,  $(4, 3)$ ,  $\dots$ , the desired probability is  $p_1 + p_2 + \dots$ , where  $p_k$  is the probability that node  $(k + 1, k)$  is hit before any of  $(j + 1, j)$  for  $j < k$ .

All paths from the root to  $(k + 1, k)$  are equally likely, and the paths that meet our restrictions are equivalent to the paths in 7.2.1.6–(28). Thus we can use Eq. 7.2.1.6–(23) to show that  $p_k = 1/(2k - 1) - 1/(2k)$ ; and  $1 - 1/2 + 1/3 - 1/4 + \dots = \ln 2$ .

(b, c) If  $p_k$  is the probability of hitting node  $((t - 1)k + 1, k)$  before any previous  $((t - 1)j + 1, j)$ , a similar calculation using the  $t$ -ary ballot numbers  $C_{pq}^{(t)}$  yields  $p_k = (t - 1)(1/(tk - 1) - 1/(tk))$ . Then  $\sum_{k=1}^{\infty} p_k = 1 - (1 - 1/t)H_{1-1/t}$  (see Appendix A).

*Notes:* We have  $\Pr(S = 1/2) = 1 - \ln 2$ , since  $S$  is always  $\geq 1/2$ . But we *cannot* claim that  $\Pr(S \geq 2/3)$  is the sum of cases that pass through  $(2, 1)$ ,  $(4, 2)$ ,  $(6, 3)$ , etc., because the supremum might be  $2/3$  even though the value  $2/3$  is never reached. Those cases occur with probability  $\pi/\sqrt{27}$ ; hence  $\Pr(S = 2/3) \geq 2\pi/\sqrt{27} - \ln 3 \approx .111$ . A determination of the exact value of  $\Pr(S = 2/3)$  is beyond the scope of this book, because we've avoided the complications of measure theory by defining probability only in discrete spaces; we can't consider a limiting quantity such as  $S$  to be a random variable, by our definitions! But we *can* assign a probability to the event that  $\max(Z_0, Z_1, \dots, Z_n) > x$ , for any given  $n$  and  $x$ , and we can reason about the limits of such probabilities.

With the help of deeper methods, E. Schulte-Geers and W. Stadje have proved that the supremum is reached within  $n$  steps, a.s. Hence  $\Pr(S = 2/3) = 2\pi/\sqrt{27} - \ln 3$ ; indeed,  $\Pr(S \text{ is rational}) = 1$ , since only rationals are reached; and  $\Pr(S = (t - 1)/t) = (2 - 3/t)H_{1-1/t} - (1 - 2/t)H_{1-2/t} - (t - 2)/(t - 1)$ . [*J. Applied Prob.* **52** (2015), 180–190.]

**89.** Set  $Y_n = X_n - p_n$ ,  $a_n = -p_n$ ,  $b_n = 1 - p_n$ . (Incidentally, exercise 1.2.10–22 gives an upper bound for this quantity that has quite a different form.)

**90.** (a) Apply Markov's inequality to  $\Pr(e^{Y_1 + \dots + Y_n} t \geq e^{tx})$ .

(b)  $e^{yt} \leq e^{-pt}(q - y) + e^{qt}(y + p) = e^{f(t)} + ye^{g(t)}$  because the function  $e^{yt}$  is convex.

(c) We have  $f'(t) = -p + pe^t/(q + pe^t)$  and  $f''(t) = pqe^t/(q + pe^t)^2$ ; hence  $f(0) = f'(0) = 0$ . And  $f''(t) \leq 1/4$ , because the geometric mean of  $q$  and  $pe^t$ ,  $(pqe^t)^{1/2}$ , is less than or equal to the arithmetic mean,  $(q + pe^t)/2$ .

(d) Set  $c = b - a$ ,  $p = -a/c$ ,  $q = b/c$ ,  $Y = Y/c$ ,  $t = ct$ ,  $h(t) = e^{g(ct)}/c$ .

(e) In  $E((e^{c_1^2 t^2/4} + Y_1 h_1(t)) \dots (e^{c_n^2 t^2/4} + Y_n h_n(t)))$  the terms involving  $h_k(t)$  all drop out, because  $\langle Y_n \rangle$  is fair. So we're left with the constant term,  $e^{ct^2/4}$ .

(f) Let  $t = 2x/c$ , to make  $ct^2/4 - xt = -x^2/c$ .

**91.**  $E(Z_{n+1} | X_0, \dots, X_n) = E(E(Q | X_0, \dots, X_n, X_{n+1}) | X_0, \dots, X_n)$ , and this is equal to  $E(Q | X_0, \dots, X_n)$  by formula (\*) in answer 35. Apply exercise 77.

**92.**  $Q_0 = E X_m = 1/2$ . If  $n < m$  we have  $Q_n = E(X_m | X_0, \dots, X_n)$ , which is the same as  $E(X_{n+1} | X_0, \dots, X_n)$  (see exercise 72); and this is  $(1 + X_1 + \dots + X_n)/(n + 2)$ , which is the same as  $Z_n$  in (27). If  $n \geq m$ , however, we have  $Q_n = X_m$ .

**93.** Everything goes through exactly as before, except that we must replace the quantity  $(m - 1)^t/m^{t-1}$  by the generalized expected value, which is  $\sum_{k=1}^m \prod_{n=1}^t (1 - p_{nk})$ .

**94.** If the  $X$ 's are dependent, the Doob martingale still is well defined; but when we write its fair sequence as an average of  $\Delta(x_1, \dots, x_t)$  there is no longer a nice formula such as (40). In any formula for  $\Delta$  that has the form  $\sum_x p_x(Q(\dots x_n \dots) - Q(\dots x \dots))$ ,  $\Pr(X_n = x_n, X_{n+1} = x_{n+1}, \dots)/(\Pr(X_n = x_n) \Pr(X_{n+1} = x_{n+1}, \dots))$  must equal  $\sum_x p_x$ , so it must be independent of  $x_n$ . Thus (41) can't be used.

**95.** False; the probability of only one red ball at level  $n$  is  $1/(n+1) = \Omega(n^{-1})$ . But there are a.s. more than 100 red balls, because that happens with probability  $(n - 99)/(n + 1)$ .

$t$ -ary ballot numbers  
measure theory  
Schulte-Geers  
Stadje  
Markov's inequality  
convex  
geometric mean  
arithmetic mean  
a.s.

**96.** Exercise 1.2.10–21, with  $\epsilon n$  equal to the bound on  $|X - n/2|$ , tells us that (i) is q.s. and that (i), (ii), (iii) are a.s. To prove that (iv) isn't a.s., we can use Stirling's approximation to show that  $\binom{n}{n/2 \pm k}/2^n$  is  $\Theta(n^{-1/2})$  when  $k = \sqrt{n}$ ; consequently  $\Pr(|X| < \sqrt{n}) = \Theta(1)$ . A similar calculation shows that (ii) isn't q.s.

probability generating function  
Knuth  
Motwani  
Pittel  
Karp  
Upfal  
Wigderson  
Motwani  
Raghavan  
Janson  
supermartingale  
Janson  
superpolynomially small

**97.** We need to show only that a *single* bin q.s. receives that many. The probability generating function for the number of items  $H$  that appear in any particular bin is  $G(z) = ((n - 1 + z)/n)^N$ , where  $N = \lfloor n^{1+\delta} \rfloor$ . If  $r = \frac{1}{2}n^\delta$ , we have

$$\Pr(H \leq r) \leq \left(\frac{1}{2}\right)^{-r} G\left(\frac{1}{2}\right) = 2^r \left(1 - \frac{1}{2n}\right)^{\lfloor 2nr \rfloor} \leq 2^r \left(1 - \frac{1}{2n}\right)^{2nr-1} \leq 2^{r+1} e^{-r},$$

by 1.2.10–(24). And if  $r = 2n^\delta$  we have

$$\Pr(H \geq r) \leq 2^{-r} G(2) = 2^{-r} \left(1 + \frac{1}{n}\right)^{\lfloor nr/2 \rfloor} \leq 2^{-r} \left(1 + \frac{1}{n}\right)^{nr/2} \leq 2^{-r} e^{r/2},$$

by 1.2.10–(25). Both are exponentially small. [See Knuth, Motwani, and Pittel, *Random Structures & Algorithms* 1 (1990), 1–14, Lemma 1.]

**98.** Let  $E_n = \mathbb{E} R$ , where  $R$  is the number of reduction steps; and suppose  $F(n) = k$  with probability  $p_k$ , where  $\sum_{k=1}^n p_k = 1$  and  $\sum_{k=1}^n k p_k = g \geq g_n$ . (The values of  $p_1, \dots, p_n$ , and  $g$  might be different, in general, every time we compute  $F(n)$ .)

Let  $\Sigma_a^b = \sum_{j=a}^b 1/g_j$ . Clearly  $E_0 = 0$ . And if  $n > 0$ , we have by induction

$$\begin{aligned} E_n &= 1 + \sum_{k=1}^n p_k E_{n-k} \leq 1 + \sum_{k=1}^n p_k \Sigma_1^{n-k} = 1 + \sum_{k=1}^n p_k (\Sigma_1^n - \Sigma_{n-k+1}^n) \\ &= \Sigma_1^n + 1 - \sum_{k=1}^n p_k \Sigma_{n-k+1}^n \leq \Sigma_1^n + 1 - \sum_{k=1}^n p_k \frac{k}{g_n} \leq \Sigma_1^n. \end{aligned}$$

[See R. M. Karp, E. Upfal, and A. Wigderson, *J. Comp. and Syst. Sci.* 36 (1988), 252.]

**99.** The same proof would work, provided that induction could be justified, if we were to do the sums from  $k = -\infty$  to  $n$  and define  $\Sigma_a^b = -\sum_{j=b+1}^{a-1} 1/g_j$  when  $a > b$ . (For example, that definition gives  $-\Sigma_{n+3}^n = 1/g_{n+1} + 1/g_{n+2} \leq 2/g_n$ .)

And in fact it does become a proof, by induction on  $m$ , that we have  $E_{m,n} \leq \Sigma_1^n$  for all  $m, n \geq 0$ , where  $E_{m,n} = \mathbb{E} \min(m, R)$ . Indeed, we have  $E_{0,n} = E_{m+1,0} = 0$ ; and  $E_{m+1,n} = 1 + \sum_{k=-\infty}^n p_k E_{m,n-k}$  when  $n > 0$ . [This problem is exercise 1.6 in *Randomized Algorithms* by Motwani and Raghavan (1995). Svante Janson observes that the random variable  $Z_m = \Sigma_1^{X_m} + \min(m, R)$  is a supermartingale, where  $X_m$  is the value of  $X$  after  $m$  iterations, as a consequence of this proof.]

**100.** (a)  $\sum_{k=1}^m k p_k \leq \mathbb{E} \min(m, T) = p_1 + 2p_2 + \dots + mp_m + mp_{m+1} + \dots + mp_\infty \leq \mathbb{E} T$ .

(b)  $\mathbb{E} \min(m, T) \geq mp_\infty$  for all  $m$ . (We assume that  $\infty \cdot p = (p > 0? \infty : 0)$ .)

**101.** (Solution by Svante Janson.) If  $0 < t < \min(p_1, \dots, p_m) = p$ , we have  $\mathbb{E} e^{tX} = \prod_{k=1}^m \mathbb{E} e^{tX_k} = \prod_{k=1}^m p_k / (e^{-t} - 1 + p_k) < \prod_{k=1}^m p_k / (p_k - t)$ , because  $e^{-t} - 1 > -t$ . By 1.2.10–(25), therefore, and setting  $t = \theta/\mu$ ,  $\Pr(X \geq r\mu) \leq e^{-r\theta} \prod_{k=1}^m p_k / (p_k - t) = \exp(-r\theta - \sum_{k=1}^m \ln(1 - t/p_k)) \leq \exp(-r\theta - \sum_{k=1}^m (t/p_k) \ln(1 - \theta)/\theta) = \exp(-r\theta - \ln(1 - \theta))$ . Choose  $\theta = (r - 1)/r$  to get the desired bound  $re^{1-r}$ . (The bound is nearly sharp when  $m = 1$  and  $p$  is small, since  $\Pr(X \geq r/p) = (1 - p)^{\lfloor r/p \rfloor - 1} \approx e^{-r}$ .)

**102.** Applying exercise 101 with  $\mu \leq s_1 + \dots + s_m$  and  $r = \ln n$  gives probability  $O(n^{-1} \log n)$  that  $(s_1 + \dots + s_m)r$  trials aren't enough. And if  $r = f(n) \ln n$ , where  $f(n)$  is any increasing function that is unbounded as  $n \rightarrow \infty$ , the probability that  $s_k r$  trials don't obtain coupon  $k$  is superpolynomially small. So is the probability that any one of a polynomial number of such failures will occur.

**103.** (a) The recurrence  $p_{0ij} = [i=j]$ ,  $p_{(n+1)ij} = \sum_{k=0}^2 p_{nik}([f_0(k)=j] + [f_1(k)=j])/2$  leads to generating functions  $g_{ij} = \sum_{n=0}^{\infty} p_{nij} z^n$  that satisfy  $g_{i0} = [i=0] + (g_{i0} + g_{i1})z/2$ ,  $g_{i1} = [i=1] + (g_{i0} + g_{i2})z/2$ ,  $g_{i2} = [i=2] + (g_{i1} + g_{i2})z/2$ . From the solution  $g_{i0} = A + B + C$ ,  $g_{i1} = A - 2B$ ,  $g_{i2} = A + B - C$ ,  $A = \frac{1}{3}/(1-z)$ ,  $B = \frac{1}{6}(1-3[i=1])/(1+z/2)$ , and  $C = \frac{1}{2}([i=0] - [i=2])/(1-z/2)$ , we conclude that the probability is  $\frac{1}{3} + O(2^{-n})$ ; in fact it is always either  $\lfloor 2^n/3 \rfloor/2^n$  or  $\lceil 2^n/3 \rceil/2^n$ . The former occurs if and only if  $i \neq j$  and  $n$  is even, or  $i + j = 2$  and  $n$  is odd.

(b) Letting  $g_{012} = \frac{z}{2}(g_{001} + g_{112})$ ,  $g_{001} = \frac{z}{2}([j=0] + g_{011})$ , etc., yields the generating function  $g_{012} = ([j \neq 1] + [j=1]z)z^2/(4-z^2)$ . Hence each  $j$  occurs with probability  $1/3$ , and the generating function for  $N$  is  $z^2/(2-z)$ ; mean = 3, variance = 2.

(c) Now  $g_{001} = \frac{z}{2}([j=0] + g_{112})$ , etc.; the output is never 1; 0 and 2 are equally likely; and  $N$  has the same distribution as before.

(d) Functional composition isn't commutative, so the stopping criterion is different: In the second case, 111 cannot occur unless the previous step had 000 or 222. The crucial difference is that, without stopping, process (b) becomes *fixed* at coalescence; process (c) continues to *change*  $a_0 a_1 a_2$  as  $n$  increases (although all three remain equal).

(e) If  $T$  is even,  $\text{sub}(T)$  returns  $(-1, 0, 1, 2)$  with probability  $(2, (2^T - 1)/3, (2^T - 4)/3, (2^T - 1)/3)/2^T$ . Thus the supposed alternative to (b) will output 0 with probability  $\frac{1}{4} + \frac{5}{32} + \frac{85}{4096} + \dots = \frac{1}{3} \sum_{k=1}^{\infty} 2^{k+1}(2^{2^k} - 1)/2^{2^{k+1}} \approx 0.427$ , *not*  $1/3$ .

(f) Change  $\text{sub}(T)$  to use consistent bits  $X_T, X_{T-1}, \dots, X_1$  instead of generating new random bits  $X$  each time; then the method of (b) is faithfully simulated. (The necessary consistency can be achieved by carefully resetting the seed of a suitable random number generator at appropriate times.)

[The technique of (f) is called "coupling from the past" in a monotone Monte Carlo simulation. It can be used to generate uniformly random objects of many important kinds, and it runs substantially faster than method (b) when there are thousands or millions of possible states instead of just three. See J. G. Propp and D. B. Wilson, *Random Structures & Algorithms* **9** (1996), 223–252.]

**104.** Let  $q = 1 - p$ . The probability of output  $(0, 1, 2)$  in (b) is  $(q^2, 2pq, p^2)$ ; in (c) it is  $(p^2 + pq^2, 0, q^2 + qp^2)$ . In both cases  $N$  has generating function  $(1 - pq(2 - z))z^2/(1 - pqz^2)$ , mean  $3/(1 - pq) - 1$ , variance  $(5 - 2pq)pq/(1 - pq)^2$ .

**105.** Suppose  $n = 2m$  is even. Experiments for small  $m$  suggest that there are polynomials  $t_k$  such that  $g_a = z^a t_{m-a}/t_m$  for  $0 \leq a \leq m$ ; and indeed, the polynomials defined by  $t_0 = t_1 = 1$ ,  $t_{k+1} = 2t_k - z^2 t_{k-1}$  fill the bill, because they make  $g_m = z g_{m-1}$ . The generating function  $T(w) = \sum_{m=0}^{\infty} t_m w^m = (1-w)/(1-2w+w^2 z^2)$  now shows, after differentiation by  $z$ , that we have  $t'_m(1) = -m(m-1)$  and  $t''_m(1) = (m^2 - 5m + 3)m(m-1)/3$ ; hence  $t''_m(1) + t'_m(1) - t'_m(1)^2 = \frac{2}{3}(m^2 - m^4)$ . The mean and variance, given  $a$ , are therefore  $a - (m-a)(m-a-1) + m(m-1) = a(n-a)$  and  $\frac{2}{3}(m-a)^2 - (m-a)^4 - m^2 + m^4 = \frac{1}{3}(n^2 - 2a(n-a) - 2)a(n-a)$ , respectively.

When  $n = 2m - 1$  we can write  $g_a = z^a u_{m-a}/u_m$  for  $0 \leq a \leq m$ , with  $u_{m+1} = 2u_m - z^2 u_{m-1}$ . In this case we want  $u_0 = 1$  and  $u_1 = z$ , so that  $g_m = g_{m-1}$ . From  $U(w) = \sum_{m=0}^{\infty} u_m w^m = (1 + (z-2)w)/(1-2w+w^2 z^2)$  we deduce  $u'_m(1) = -m(m-2)$  and  $u''_m(1) = m(m-1)(m^2 - 7m + 7)/3$ . It follows that, also in this case, the mean number of steps in the walk is  $a(n-a)$  and the variance is  $\frac{1}{3}(n^2 - 2a(n-a) - 2)a(n-a)$ .

[The polynomials  $t_m$  and  $u_m$  in this analysis are disguised relatives of the classical Chebyshev polynomials defined by  $T_m(\cos \theta) = \cos m\theta$ ,  $U_m(\cos \theta) = \sin(m+1)\theta/\sin \theta$ . Let us also write  $V_m(\cos \theta) = \cos(m - \frac{1}{2})\theta/\cos \frac{1}{2}\theta$ . Then  $V_m(x) = (2 - 1/x)T_m(x) + (1/x - 1)U_m(x)$ ; and we have  $t_m = z^m T_m(1/z)$ ,  $u_m = z^m V_m(1/z)$ .]

recurrence  
generating functions  
commutative  
random number generator  
coupling from the past  
monotone Monte Carlo  
Monte Carlo  
generation of random objects  
Propp  
Wilson  
Chebyshev polynomials



**106.** Before coalescing, the array  $a_0 a_1 \dots a_{d-1}$  always has the form  $a^r(a+1) \dots (b-1)b^s$  for some  $0 \leq a < b < d$ ,  $r > 0$ , and  $s > 0$ , where  $r + s + b - a = d + 1$ . Initially  $a = 0$ ,  $b = d - 1$ ,  $r = s = 1$ . The behavior of the algorithm while  $r + s = t$  is like a random walk on the  $t$ -cycle, as in the previous exercise, starting at  $a = 1$ . Let  $G_t$  be the generating function for that problem, which has mean  $t - 1$  and variance  $2\binom{t}{3}$ . Then this problem has the generating function  $G_2 G_3 \dots G_d$ ; so its mean is  $\sum_{k=2}^d (k - 1) = \binom{d}{2}$ , and the variance is  $\sum_{k=2}^d 2\binom{k}{3} = 2\binom{d+1}{4}$ .

random walk on the  $t$ -cycle  
order ideals  
Nawrotzki  
Strassen  
Doebelin  
Dobrushin  
Jaccard index  
Jaccard

**107.** (a) If the probabilities can be renumbered so that  $p_1 \leq q_1$  and  $p_2 \leq q_2$ , the five events of  $\Omega$  can have probabilities  $p_1, p_2, q_1 - p_1, q_2 - p_2$ , and  $q_3$ , because  $p_3 = (q_1 - p_1) + (q_2 - p_2) + q_3$ . But if that doesn't work, we can suppose that  $p_1 < q_1 \leq q_2 \leq p_3 < p_2 \leq p_3$ . Then  $p_1, q_1 - p_1, p_1 + p_2 - q_1, p_3 - q_3$ , and  $q_3$  are nonnegative.

(b) Give  $\Omega$ 's events the probabilities  $\frac{1}{12}, \frac{2}{12}, \frac{3}{12}, \frac{6}{12}$ .

(c) For example, let  $p_1 = \frac{1}{9}, p_2 = p_3 = \frac{4}{9}, q_1 = q_2 = q_3 = \frac{1}{3}$ .

**108.** Let  $p_k = \Pr'(X = k)$  and  $q_k = \Pr''(Y = k)$ . The set  $\bigcup_n \{\sum_{k \leq n} p_k, \sum_{k \leq n} q_k\}$  divides the unit interval  $[0, 1)$  into countably many subintervals, which we take as the set  $\Omega$  of atomic events  $\omega$ . Let  $X(\omega) = n$  if and only if  $\omega \subseteq [\sum_{k < n} p_k, \sum_{k \leq n} p_k)$ ; a similar definition works for  $Y(\omega)$ . And  $X(\omega) \leq Y(\omega)$  for all  $\omega$ .

**109.** (a) We're given that  $p_1 + p_3 \leq q_1 + q_3, p_2 + p_3 \leq q_2 + q_3$ , and  $p_3 \leq q_3$ . (Also that  $0 \leq 0$  and  $p_1 + p_2 + p_3 \leq q_1 + q_2 + q_3$ ; but those inequalities always hold.) We must find a coupling with  $p_{12} = p_{21} = p_{31} = p_{32} = 0$ , because  $1 \not\leq 2, 2 \not\leq 1, 3 \not\leq 1$ , and  $3 \not\leq 2$ . In the previous problem we were given that  $p_2 + p_3 \leq q_2 + q_3$  and  $p_3 \leq q_3$ , and we had to find a coupling with  $p_{21} = p_{31} = p_{32} = 0$ .

(b) Let  $A^\uparrow = \{x \mid x \succeq a \text{ for some } a \in A\}$  and  $B^\downarrow = \{x \mid x \preceq b \text{ for some } b \in B\}$ . We're given that  $\Pr'(X \in A^\uparrow) \leq \Pr''(Y \in A^\uparrow)$  for all  $A$ . Let  $A = \{1, \dots, n\} \setminus B^\downarrow$ , so that  $\Pr'(X \in B^\downarrow) = 1 - \Pr'(X \in A)$ . The result follows because  $A = A^\uparrow$ .

(c) Remove all arcs  $x_i \rightarrow x_j$  from the network when  $i \not\leq j$ . Then a blocking pair  $(I, J)$  has the property that  $i \preceq j$  implies  $i \in I$  or  $j \in J$ . Let  $A = \{x \mid x \preceq a \text{ for some } a \notin J\}$  and  $B = \{1, \dots, n\} \setminus A$ . Then  $A \subseteq I, B \subseteq J$ , and  $B = B^\downarrow$ . Hence  $\sum_{i \in I} p_i + \sum_{j \in J} q_j \geq \sum_{i \in A} p_i + \sum_{j \in B} q_j \geq \sum_{i \in A} q_i + \sum_{j \in B} q_j = 1$ .

[See K. Nawrotzki, *Mathematische Nachrichten* **24** (1962), 193-200; V. Strassen, *Annals of Mathematical Statistics* **36** (1965), 423-439.]

**110.** (a) The result is trivial if  $r = 1$ . Otherwise consider the probability distributions  $p'_k = (p_k - r_k)/(1 - r)$  and  $q'_k = (q_k - r_k)/(1 - r)$ ; use the coupling  $p_{ij} = (1 - r)p'_i q'_j + r_j \delta_{ij}$ . [See W. Doebelin, *Revue mathématique de l'Union Interbalkanique* **2** (1938), 77-105; R. L. Dobrushin, *Teoriya Veroyatnostei i ee Primeneniia* **15** (1970), 469-497.]

(b) Yes, because the  $(p', q')$  distribution satisfies the hypotheses of that exercise.

**111.** (a) Here are the 60 triples  $1\pi 3\pi 4\pi$ , with the minima in bold type:

134 163 123 126 142 142 153 145 163 154 245 234 534 563 623 526 632 652 534 643  
 356 645 246 234 435 463 524 423 642 532 461 351 361 641 251 231 341 531 321 421  
 512 412 415 315 316 615 216 216 415 316 623 526 652 452 564 354 465 364 256 265

(b) Both  $S_A$  and  $S_B$  lie in  $A \cup B$ . Each element of  $A \cup B$  is equally likely to have the minimum value  $a\pi$ ; exactly  $|A \cap B|$  of those elements have that value as their sketch.

(c)  $|A \cap B \cap C|/|A \cup B \cup C|$ .

*Notes:* The ratio  $|A \cap B|/|A \cup B|$  is a useful measure of similarity called the Jaccard index, because Paul Jaccard used it to compare different Swiss sites according to the sets of plant species seen at each place [*Bulletin de la Société Vaudoise des Sciences*

*Naturelles* **37** (1901), 249]. It is commonly used today to rank the similarity between web pages, based on a certain set of words in each page.

Minwise independence was introduced by Andrei Broder for that application in 1997, using  $n = 2^{64}$  and a method of identifying roughly 1000 words  $A$  on a typical web page. By calculating, say, independent sketches  $S_1(A), \dots, S_{100}(A)$  for each page, the number of  $j$  such that  $S_j(A) = S_j(B)$  gives a highly reliable and quickly computable estimate of the Jaccard index. A perfectly minwise independent family is impossible in practice when  $n$  is huge, but the associated theory has suggested approximate “minhash” algorithms that work well. See A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, *J. Computer and System Sciences* **60** (2000), 630–659.

web pages  
 Broder  
 minhash  
 Charikar  
 Frieze  
 Mitzenmacher  
 uniform probing  
 Takei  
 Itoh  
 Shinozaki

**112.** (a) Such a rule breaks ties properly, provided that the number of  $\pi$  with  $\infty$ 's in  $B$  is a multiple of  $n - m$ . Each  $B$  can have its own rule.

(b) In fact we can produce families whose permutations are all obtained from  $N/n = d$  “seeds” by cyclic shifts, as in exercise 111. Begin with  $m = 1$  and a table of  $N = \text{lcm}(1, 2, \dots, n)$  partial permutations whose entries  $\pi_{ij}$  for  $1 \leq i \leq N$  and  $1 \leq j \leq n$  are entirely blank, except that  $\pi_{ij} = 1$  for each pair  $ij$  with  $(j - 1)d < i \leq jd$  and  $1 \leq j \leq n$ . When  $n = 4$ , for instance, the initial tableau

1<sub>UUU</sub> 1<sub>UUU</sub> 1<sub>UUU</sub> 1<sub>UUU</sub> 1<sub>UUU</sub> 1<sub>UUU</sub> 1<sub>UUU</sub> 1<sub>UUU</sub> 1<sub>UUU</sub> 1<sub>UUU</sub> 1<sub>UUU</sub> 1<sub>UUU</sub>

represents  $N = 12$  truncated permutations with  $m = 1$ . We'll insert some 2s next.

Let  $A$  be a subset of size  $n - m$  that is all blank, in some  $\pi$ . Each  $A$  occurs equally often (as in uniform probing, Section 6.4); so the number of such  $\pi$  is  $N/\binom{n}{n-m}$ . Fortunately this is a multiple of  $n - m$ , because exercise 1.2.6–48 tells us that  $N/\binom{n}{n-m} = N \sum_{k=0}^m (-1)^k \binom{m}{k} / (n - m + k)$ .

Take  $n - m$  such  $\pi$  and insert  $m + 1$  into different positions within them. Then find another such  $A$ , if possible, and repeat the process until no blank subsets of size  $n - m$  remain. Then set  $m \leftarrow m + 1$ , and continue in the same way until  $m = n$ .

It's not hard to see that the insertions can be done so that  $\pi_j, \pi_{d+j}, \dots, \pi_{(n-1)d+j}$  are maintained as cyclic shifts of each other. When  $n = 4$  the 2s are essentially forced:

12<sub>UU</sub> 1<sub>U2U</sub> 1<sub>UU2</sub> 1<sub>U2U</sub> 1<sub>UU2</sub> 21<sub>UU</sub> 1<sub>UU2</sub> 2<sub>U1U</sub> 1<sub>U2U</sub> 2<sub>UU1</sub> 1<sub>U2U</sub> 1<sub>UU2</sub>

But then there are two ways to fill the two cases with  $A = \{3, 4\}$ :

123<sub>U</sub> 1<sub>U2U</sub> 13<sub>U2</sub> 1<sub>U23</sub> 1<sub>U23</sub> 21<sub>U3</sub> 3<sub>U12</sub> 2<sub>U1U</sub> 1<sub>U213</sub> 23<sub>U1</sub> 1<sub>U2U1</sub> 3<sub>U21</sub>  
 12<sub>U3</sub> 1<sub>U2U</sub> 13<sub>U2</sub> 31<sub>U2</sub> 1<sub>U23</sub> 213<sub>U</sub> 1<sub>U312</sub> 2<sub>U1U</sub> 1<sub>U213</sub> 2<sub>U31</sub> 1<sub>U2U1</sub> 3<sub>U21</sub>

Adopting the first of these leads to two ways to fill  $A = \{2, 4\}$ :

123<sub>U</sub> 132<sub>U</sub> 13<sub>U2</sub> 1<sub>U23</sub> 1<sub>U32</sub> 21<sub>U3</sub> 3<sub>U12</sub> 2<sub>U13</sub> 1<sub>U213</sub> 23<sub>U1</sub> 32<sub>U1</sub> 3<sub>U21</sub>  
 123<sub>U</sub> 1<sub>U23</sub> 13<sub>U2</sub> 1<sub>U23</sub> 31<sub>U2</sub> 21<sub>U3</sub> 3<sub>U12</sub> 231<sub>U</sub> 1<sub>U213</sub> 23<sub>U1</sub> 1<sub>U231</sub> 3<sub>U21</sub>

Here  $A$  is a cyclic shift of itself, but consistent placement is always possible.

[See Yoshinori Takei, Toshiya Itoh, and Takahiro Shinozaki, *IEICE Transactions on Fundamentals* **E83-A** (2000), 646–655, 747–755.]

**113.** (a) The probability is zero if  $l \geq k$  or  $r > n - k$ . Otherwise the result follows if we can prove it in the “complete” case when  $l = k - 1$  and  $r = n - k$ , because we can sum the probabilities of complete cases over all ways to specify which of the unconstrained elements are  $< k$  and which are  $> k$ .

To prove the complete case, we may assume that  $a_i = i$ ,  $b = k$ , and  $c_j = k + j$  for  $1 \leq i \leq l = k - 1$  and  $1 \leq j \leq r = n - k$ . The probability can be computed

via the principle of inclusion and exclusion, because we know  $\Pr(\min_{a \in A} a\pi = k\pi) = 1/(n - k + t) = P_B$  whenever  $A = \{k, \dots, n\} \cup B$  and  $B$  consists of  $t$  elements less than  $k$ . For example, if  $k = 4$  the probability that  $4\pi = 4$  and  $\{1\pi, 2\pi, 3\pi\} = \{1, 2, 3\}$  is  $P_\emptyset - P_{\{1\}} - P_{\{2\}} - P_{\{3\}} + P_{\{1,2\}} + P_{\{1,3\}} + P_{\{2,3\}} - P_{\{1,2,3\}}$ ; each of those probabilities is correct for truly random  $\pi$ .

(b) This event is the disjoint union of complete events of type (a). [See A. Z. Broder and M. Mitzenmacher, *Random Structures & Algorithms* **18** (2001), 18–30.]

*Notes:* The function  $\psi(n) = \ln(\text{lcm}(1, 2, \dots, n)) = \sum_{p^k \leq n} [p \text{ prime}] \ln p$  was introduced by P. L. Chebyshev [see *J. de mathématiques pures et appliquées* **17** (1852), 366–390], who proved that it is  $\Theta(n)$ . Refinements by C.-J. de la Vallée Poussin [*Annales de la Société Scientifique de Bruxelles* **20** (1896), 183–256] showed that in fact  $\psi(n) = n + O(ne^{-C \log n})$  for some positive constant  $C$ . Thus  $\text{lcm}(1, 2, \dots, n)$  grows roughly as  $e^n$ , and we cannot hope to generate a list of minwise independent permutations when  $n$  is large; the length of such a list is 232,792,560 already for  $19 \leq n \leq 22$ .

**114.** First assume that  $|S_j| = d_j + 1$  for all  $j$ , and let  $g_j(x) = \prod\{(x - s) \mid s \in S_j\}$ . We can replace  $x_j^{d_j+1}$  by  $g_j(x_j)$ , without changing the value of  $f(x_1, \dots, x_n)$ , when  $x_j \in S_j$ . Doing this repeatedly until every term of  $f$  has degree  $\leq d_j$  in each variable  $x_j$  will produce a polynomial that has at least one nonroot in  $S_1 \times \dots \times S_n$ , according to exercise 4.6.1–16. [See N. Alon, *Combinatorics, Probab. and Comput.* **8** (1999), 7–29.]

Now in general, if there were at most  $|S_1| + \dots + |S_n| - (d_1 + \dots + d_n + n)$  nonroots, we could find subsets  $S'_j \subseteq S_j$  with  $|S'_j| = d_j + 1$  such that  $S'_j$  differs from  $x_j$  in  $|S_j| - d_j - 1$  of the nonroots and  $S'_1 \times \dots \times S'_n$  avoids them all—a contradiction.

(This inequality also implies stronger lower bounds when the sets  $S_j$  are large. If, for example,  $d_1 = \dots = d_n = d$  and if each  $|S_j| \geq s$ , where  $s = d + 1 + \lceil d/(n-1) \rceil$ , we can decrease each  $|S_j|$  to  $s$  and increase the right-hand side. For further asymptotic improvements see Béla Bollobás, *Extremal Graph Theory* (1978), §6.2 and §6.3.)

**115.** Representing the vertex in row  $x$  and column  $y$  by  $(x, y)$ , if all points could be covered we'd have  $f(x, y) = \prod_{j=1}^p (x - a_j) \prod_{j=1}^q (y - b_j) \prod_{j=1}^r (x + y + c_j)(x - y + d_j) = 0$ , for all  $1 \leq x \leq m$  and  $1 \leq y \leq n$  and for some choices of  $a_j, b_j, c_j, d_j$ . But  $f$  has degree  $p + q + 2r = m + n - 2$ , and the coefficient of  $x^{m-1}y^{n-1}$  is  $\pm \binom{r}{\lfloor r/2 \rfloor} \neq 0$ .

**116.** Let  $g_v = \sum\{x_e \mid v \in e\}$  for each vertex  $v$ , including  $x_e$  twice if  $e$  is a loop from  $v$  to itself. Apply the nullstellensatz with  $f = \prod_v (1 - g_v^{p-1}) - \prod_e (1 - x_e)$  and with each  $S_j = \{0, 1\}$ , using mod  $p$  arithmetic. This polynomial has degree  $m$ , the number of edges and variables, because the first product has degree  $(p-1)n < m$ ; and the coefficient of  $\prod_e x_e$  is  $(-1)^m \neq 0$ . Hence there is a solution  $x$  that makes  $f(x)$  nonzero. The subgraph consisting of all edges with  $x_e = 1$  in this solution is nonempty and satisfies the desired condition, because  $g_v(x) \bmod p = 0$  for all  $v$ .

(This proof works also if we consider that a loop contributes just 1 to the degree. See N. Alon, S. Friedland, and G. Kalai, *J. Combinatorial Theory* **B37** (1984), 79–91.)

**117.** If  $\omega = e^{2\pi i/m}$ , we have  $E\omega^{jX} = \sum_{k=0}^n \binom{n}{k} p^k (1-p)^{n-k} \omega^{jk} = (\omega^j p + 1 - p)^n$ . Also  $|\omega^j p + 1 - p|^2 = p^2 + (1-p)^2 + p(1-p)(\omega^j + \omega^{-j}) = 1 - 4p(1-p)\sin^2(\pi j/m)$ . Now  $\sin \pi t \geq 2t$  for  $0 \leq t \leq 1/2$ . Hence, if  $0 \leq j \leq m/2$  we have  $|\omega^j p + 1 - p|^2 \leq 1 - 16p(1-p)j^2/m^2 \leq \exp(-16p(1-p)j^2/m^2)$ ; if  $m/2 \leq j \leq m$  we have  $\sin(\pi j/m) = \sin(\pi(m-j)/m)$ . Thus  $\sum_{j=1}^{m-1} |E\omega^{jX}| \leq 2 \sum_{j=1}^{m-1} \exp(-8p(1-p)j^2/m^2)$ .

The result follows, since  $\Pr(X \bmod m = r) = \frac{1}{m} \sum_{j=0}^{m-1} \omega^{-jr} E\omega^{jX}$ . [S. Janson and D. E. Knuth, *Random Structures & Algorithms* **10** (1997), 130–131.]

inclusion and exclusion  
Broder  
Mitzenmacher  
Chebyshev  
de la Vallée Poussin  
Alon  
Bollobás  
Alon  
Friedland  
Kalai  
Janson  
Knuth

**118.** Indeed, (22) with  $Y = X - x$  yields *more* (when we also apply exercise 47):

$$\begin{aligned} \Pr(X \geq x) &\geq \Pr(X > x) \geq \frac{(\mathbb{E}X - x)^2}{\mathbb{E}(X - x)^2} = \frac{(\mathbb{E}X - x)^2}{\mathbb{E}X^2 - x(2\mathbb{E}X - x)} \\ &\geq \frac{(\mathbb{E}X - x)^2}{\mathbb{E}X^2 - x\mathbb{E}X} \geq \frac{(\mathbb{E}X - x)^2}{\mathbb{E}X^2 - x^2}. \end{aligned}$$

(The attribution of this result to Paley and Zygmund is somewhat dubious. They did, however, write an important series of papers [*Proc. Cambridge Philosophical Society* **26** (1930), 337–357, 458–474; **28** (1932), 190–205] in which a related inequality appeared in the proof of Lemma 19.)

**119.** Let  $f(x, t) = \Pr(U \leq V \leq W \text{ and } V \leq (1-t)U + tW)$ ,  $g(x, t) = \Pr(U \leq W \leq V \text{ and } W \leq (1-t)U + tV)$ ,  $h(x, t) = \Pr(W \leq U \leq V \text{ and } U \leq (1-t)W + tV)$ . We want to prove that  $f(x, t) + g(x, t) + h(x, t) = t$ . Notice that, if  $\bar{U} = 1 - U$ ,  $\bar{V} = 1 - V$ ,  $\bar{W} = 1 - W$ , we have  $\Pr(W \leq U \leq V \text{ and } U \geq (1-t)W + tV) = \Pr(\bar{V} \leq \bar{U} \leq \bar{W} \text{ and } \bar{U} \leq t\bar{V} + (1-t)\bar{W})$ . Hence  $\frac{x}{2} - h(x, t) = f(1-x, 1-t)$ , and we may assume that  $t \leq x$ .

Clearly  $g(x, t) = \int_0^x \frac{du}{x} \int_x^1 \frac{dv}{1-x} t(v-u) = \frac{t}{2}$ . And  $t \leq x$  implies that

$$f(x, t) = \int_{(x-t)/(1-t)}^x \frac{du}{x} \int_x^{(1-t)u+t} \frac{dv}{1-x} (1 - (v - (1-t)u)/t) = t^2(1-x)^2/(6(1-t)x);$$

$$h(x, t) = \int_x^1 \frac{dv}{1-x} \left( \int_0^{vt} \frac{du}{x} u + \int_{vt}^x \frac{du}{x} \frac{t}{1-t}(v-u) \right) = \frac{t}{2} - f(x, t).$$

Instead of this elaborate calculation, Tamás Terpai has found a much simpler proof: Let  $A = \min(U, V, W)$ ,  $M = \langle UVW \rangle$ , and  $Z = \max(U, V, W)$ . Then the conditional distribution of  $M$ , given  $A$  and  $Z$ , is a mixture of three distributions: Either  $A = U$ ,  $Z = V$ , and  $M$  is uniform in  $[A..Z]$ ; or  $A = U$ ,  $Z = W$ , and  $M$  is uniform in  $[x..Z]$ ; or  $A = W$ ,  $Z = V$ , and  $M$  is uniform in  $[A..x]$ . (These three cases occur with respective probabilities  $(Z - A, Z - x, x - A)/(2Z - 2A)$ , but we don't need to know that detail.) The overall distribution of  $M$ , being an average of conditional uniform distributions over all  $A \leq x$  and  $Z \geq x$ , is therefore uniform.

[See S. Volkov, *Random Struct. & Algorithms* **43** (2013), 115–130, Theorem 5.]

**120.** See J. Jabbour-Hattab, *Random Structures & Algorithms* **19** (2001), 112–127.

**121.** (a)  $D(y||x) = \frac{1}{5} \lg \frac{6}{5} + \frac{2}{15} \lg \frac{4}{5} \approx .0097$ ;  $D(x||y) = \frac{1}{6} \lg \frac{5}{6} + \frac{1}{6} \lg \frac{5}{4} \approx .0098$ .

(b) We have  $\mathbb{E}(\rho(X) \lg \rho(X)) \geq (\mathbb{E}\rho(X)) \lg \mathbb{E}\rho(X)$  by Jensen's inequality (20); and  $\mathbb{E}\rho(X) = \sum_t y(t) = 1$ , so the logarithm evaluates to 0.

The question about zero is the hard part of this exercise. We need to observe that the function  $f(x) = x \lg x$  is *strictly convex*, in the sense that equality holds in (19) only when  $x = y$ . Thus we have  $(\mathbb{E}Z) \lg \mathbb{E}Z = \mathbb{E}(Z \lg Z)$  for a positive random variable  $Z$  only when  $Z$  is constant. Consequently  $D(y||x) = 0$  if and only if  $x(t) = y(t)$  for all  $t$ .

(c) Let  $\hat{x}(t) = x(t)/p$  and  $\hat{y}(t) = y(t)/q$  be the distributions of  $X$  and  $Y$  within  $T$ . Then  $0 \leq D(\hat{y}||\hat{x}) = \sum_{t \in T} \hat{y}(t) \lg(\hat{y}(t)/\hat{x}(t)) = \mathbb{E}(\lg \rho(Y) | Y \in T) + \lg(p/q)$ .

(d)  $D(y||x) = (\mathbb{E} \lg m) - H_Y = \lg m - H_Y$ . (Hence, by (b), the maximum entropy of any such random variable  $Y$  is  $\lg m$ , attainable only with the uniform distribution.)

(e)  $I_{X,Y} = -H_Z - \sum_{u,v} z(u,v) (\lg x(u) + \lg y(v)) = -H_Z + \sum_u x(u) \lg(1/x(u)) + \sum_v y(v) \lg(1/y(v))$ , because  $\sum_v z(u,v) = x(u)$  and  $\sum_u z(u,v) = y(v)$ . (One can also write  $I_{X,Y} = H_Y - H_{Y|X}$ , where  $H_{Y|X} = \sum_t x(t) H_{Y|t}$ .)

**122.** (a)  $D(y||x) = \sum_{t=0}^{\infty} (3^t/4^{t+1}) \lg(3^t/2^{t+1}) = \lg \frac{27}{16} \approx 0.755$ ;  $D(x||y) = \lg \frac{4}{3} \approx 0.415$ .

Paley  
Zygmund  
Terpai  
conditional distribution  
Volkov  
Jabbour-Hattab  
Jensen's inequality  
*strictly convex*  
uniform distribution  
conditional probability

(b) Let  $q = 1 - p$  and  $t = pn + u\sqrt{n}$ . Then we have

$$y(t) = \frac{e^{-u^2/(2pq)}}{\sqrt{2\pi pqn}} \exp\left(\left(\frac{u}{2q} - \frac{u}{2p} + \frac{u^3}{6p^2} - \frac{u^3}{6q^2}\right)\frac{1}{\sqrt{n}} + O\left(\frac{1}{n}\right)\right);$$

$$\ln \rho(t) = -\frac{u^2}{2q} - \frac{1}{2} \ln q + \left(\frac{u}{2q} - \frac{u^3}{6q^2}\right)\frac{1}{\sqrt{n}} + O\left(\frac{1}{n}\right).$$

trading tails  
Kullback  
Leibler  
Cauchy-Schwarz inequality  
variance  
Markov's inequality  
Chatterjee  
Diaconis

By restricting  $|u| \leq n^\epsilon$  and trading tails (see 7.2.1.5-(20)), we obtain

$$D(y||x) = \frac{1}{\sqrt{2\pi pqn}} \int_{-\infty}^{\infty} e^{-u^2/(2pq)} \left(-\frac{u^2}{2q \ln 2} - \frac{1}{2} \lg q\right) du\sqrt{n} + O\left(\frac{1}{n}\right)$$

$$= \frac{1}{2 \ln 2} \left(\ln \frac{1}{1-p} - p\right) + O\left(\frac{1}{n}\right).$$

In this case  $D(x||y)$  is trivially  $\infty$ , because  $x(n+1) > 0$  but  $y(n+1) = 0$ .

**123.** Since  $p_{k+1} = p_k y(t)/z_k(t)$  we have  $\rho(t) = (1 - p_k)p_{k+1}/(p_k(1 - p_{k+1}))$ . [This relation was the original motivation that led S. Kullback and R. E. Leibler to define  $D(y||x)$ , in *Annals of Mathematical Statistics* **22** (1951), 79-86.]

**124.** Let  $m = c^2 2^{D(y||x)}$  and  $g(t) = f(t)[\rho(t) \leq m]$ ; thus  $g(t) = f(t)$  except with probability  $\Delta_c$ . We have  $|E(f) - E_n(f)| = (E(f) - E(g)) + |E(g) - E_n(g)| + (E_n(f) - E_n(g))$ . The Cauchy-Schwarz inequality (exercise 1.2.3-30) implies that the first and last are bounded by  $\|f\| \sqrt{\Delta_c}$ , because  $f(t) - g(t) = f(t)[\rho(t) > m]$ .

Now  $\text{var}(\rho(X)g(X)) \leq E(\rho(X)^2 g(X)^2) \leq m E(\rho(X) f(X)^2) = m E(f(Y)^2) = m \|f\|^2$ . Hence  $(E(g) - E_n(g))^2 = \text{var} E_n(g) = \text{var}(\rho(X)g(X))/n \leq \|f\|^2/c^2$ .

Consider now the case  $c < 1$ . From Markov's inequality we have  $\Pr(\rho(X) > m) \leq (E \rho(X))/m = 1/m$ . Also  $E(\rho(X)[\rho(X) \leq m]) = E[\rho(Y) \leq m] = 1 - \Delta_c$ . Consequently  $\Pr(E_n(1) \geq a) \leq \Pr(\max_{1 \leq k \leq n} \rho(X_k) > m) + \Pr(\sum_{k=1}^n \rho(X_k)[\rho(X_k) \leq m] \geq na) \leq n/m + E(\sum_{k=1}^n \rho(X_k)[\rho(X_k) \leq m])/(na) = c^2 + (1 - \Delta_c)/a$ .

[S. Chatterjee and P. Diaconis, preprint (September 2015).]

# INDEX AND GLOSSARY

GOLDSMITH

*He writes indexes to perfection.*

— OLIVER GOLDSMITH, *Citizen of the World* (1762)

When an index entry refers to a page containing a relevant exercise, see also the *answer* to that exercise for further information. An answer page is not indexed here unless it refers to a topic not included in the statement of the exercise.

- $\gamma$  (Euler's constant), 29.
- $\nu(x)$  (sideways sum), 13.
- $\pi$  (circle ratio), 14.
- $\phi$  (golden ratio), 12, 26.
  
- A priori* versus *a posteriori* probabilities, 25.
- a.s.: Almost surely, 11–12, 20, 21, 39, 40.
- Ace Now, 8, 19.
- Ahlsvede, Rudolph, 34.
- Aldous, David John, 33.
- Almost sure events, 11.
- Alon, Noga (נוגה אלון), 44.
- Analysis of algorithms, 9, 21, 22.
- Arithmetic–geometric mean inequality, 31, 39.
- Asymptotic methods, 11, 12, 16.
- Atomic events, 1.
- Azuma, Kazuoki (吾妻一興), 9, 20.
  
- $B(p_1, \dots, p_m)$ , *see* Multivariate Bernoulli distribution.
- $B_n(p)$ , *see* Binomial distribution.
- $B_{m,n}(p)$ , *see* Cumulative binomial distribution.
- Backward versus forward, 21.
- Ballot numbers, 39.
- Bayes, Thomas, 14.
- BDD (binary decision diagram), 5, 34.
- Bell, Eric Temple, numbers  $\varpi_n$ , 15.
- Bernoulli, Daniel, 36.
- Bernoulli, Jacques (= Jakob = James), distribution, multivariate, 14, 18, 20.
- Bernoulli, Nicolas (= Nikolaus), 36.
- Beta distribution, 14.
- Bhatia, Rajendra (राजेन्द्र भाटिया), 28.
- Bienaymé, Irénée Jules, inequality, 4.
- Bin-packing problem, 11, 20.
- Binary notation, 14.
- Binary random variables, 2, 3, 5, 13–15, 20.
- Binary search trees, 24.
- Bingo, 12–13.
- Binomial distribution, 14, 24, 32.
  - cumulative, 14–15, 31.
- Bit vectors, 3, 9, 13–14.
- Bits of information, 24.
- Blackwell, David Harold, 37.
- Bollobás, Béla, 44.
- Boolean functions, 5, 15, 33, 35.
  - dual of, 33.
  - monotone, 5, 35.
  - symmetric, 16.
- Boolean random variables, *see* Binary random variables.
- Boolean vectors, *see* Bit vectors.
- Boyd, Stephen Poythress, 33.
- Bracket notation, 2.
- Bracketing property, 34.
- Broder, Andrei Zary, 43, 44.
- Brown, John O'Connor, 26.
  
- Cantelli, Francesco Paolo, inequality, 33.
- Casanova de Seingalt, Giacomo Girolamo, 36.
- Catalan, Eugène Charles, numbers, 36.
- Cauchy, Augustin Louis, inequality, 46.
- Chain rule for conditional probability, 14, 28.
- Charikar, Moses Samson (मोद्देस सॉमसन चरीकर), 43.
- Chatterjee, Sourav (সৌরভ চ্যাট্টোপাধ্যায়), 46.
- Chebyshev (= Tschebyscheff), Pafnutii Lvovich (Чебышев, Пафнутий Львович = Чебышев, Пафнутий Львович), 33, 44.
  - inequality, 4, 9, 16.
  - monotonic inequality, 35.
  - polynomials, 27, 41.
- Chesterton, Gilbert Keith, 26.
- Chicks, 15.
- Circle ratio ( $\pi$ ), 14.
- Cliques, 16–17.
- CMath: Concrete Mathematics*, a book by R. L. Graham, D. E. Knuth, and O. Patashnik, 27, 29.
- Coalescing random walk, 21.
- Coin tosses, 11–12, 19, 20.
- Column sums, 22.
- Combinatorial nullstellensatz, 23.
- Commutative law, 41.
- Compositions, 37.
- Concave functions, 4, 32.
- Conditional distribution, 3, 45.
- Conditional expectation, 2–3, 15–19.
  - inequality, 5, 16, 34.
- Conditional probability, 1, 13–14, 35, 45.

- Connelly, Robert, Jr., 38.  
 Convex combinations, 33.  
 Convex functions, 4, 8, 16, 20, 33, 38, 39.  
   strictly, 45.  
 Correlated random variables, 17–18, 28, 37.  
 Correlation inequalities, 17.  
 Coupling, 22.  
   from the past, 41.  
 Covariance, 2, 14, 17, 28, 35.  
 Cover, Thomas Merrill, 13, 28.  
 Covering all points, 23.  
 Cumulative binomial distribution, 14–15, 31.  
 Cycle graph ( $C_n$ ), 22, 41.
- Darwin, Charles Robert, 25.  
 Davis, Chandler, 28.  
 Daykin, David Edward, 34.  
 de La Vallée Poussin, Charles Jean  
   Gustave Nicolas, 44.  
 de Moivre, Abraham, 37.  
   martingale, 19.  
 de Montmort, Pierre Rémond, 36.  
 Density, relative, 24.  
 Degree of a multivariate polynomial, 23.  
 Diaconis, Persi Warren, iv, 46.  
 Diagonal lines, 23.  
 Dice, 12, 24.  
 Discrete probabilities, 1.  
 Döblin, Wolfgang (= Doebelin, Vincent), 42.  
 Dobrushin, Roland L'vovich (Добрушин,  
   Роланд Львович), 42.  
 Doob, Joseph Leo, 6, 9, 38.  
   martingales, 9–10, 20, 37, 39.  
 Dual of a Boolean function, 33.
- Eggenberger, Florian, 6.  
 Elton, John Hancock, 31.  
 Entropy, 24.  
   relative, 24.  
 Enveloping series, 34.  
 Erdős, Pál (= Paul), 34.  
 Etesami, Omid (امید اعتصامی), iv.  
 Euler, Leonhard (Ейлеръ, Леонардъ =  
   Эйлер, Леонард), constant  $\gamma$ , 29.  
 Eulerian numbers, 37.  
 Events, 1–3.  
 Exchangeable random variables, 37.  
 Expected value, 2–5, 14–16, *see also*  
   Conditional expectation.
- Fair sequences, 7, 10, 19, 38.  
   with respect to a sequence, 7, 37.  
 Families of sets, 17.  
 Feige, Uriel (אוריאל פייה), 31.  
 Feller, Willibald (= Vilim = Willy =  
   William), 36.  
 Fibonacci, Leonardo, of Pisa (= Leonardo  
   filio Bonacci Pisano), dice, 12.  
   martingale, 19.  
   numbers, 12, 38.
- First moment principle, 4, 16.  
 FKG inequality, 5, 17, 35.  
 Flow in a network, 22.  
 Fortuin, Cornelis Marius, 17.  
 Forward versus backward, 21.  
 Four functions theorem, 17, 34.  
 Friedland, Shmuel (שמואל פרידלנד), 44.  
 Friedman, Bernard, urn, 19.  
 Frieze, Alan Michael, 43.
- Games, 4, 8, 13.  
 Garey, Michael Randolph, 11.  
 Generating functions, 15, 22, 24, 32,  
   33, 36, 40, 41.  
 Generation of random objects, 41.  
 Geometric distribution, 21, 24.  
 Geometric mean and arithmetic mean,  
   31, 39.  
 Georgiadis, Evangelos (Γεωργιάδης,  
   Ευάγγελος), 28.  
 Gilat, David, 38.  
 Gimbre, Jean, 17.  
 Golden ratio ( $\phi$ ), 12, 26.  
 Goldsmith, Oliver, 47.  
 Gosper, Ralph William, Jr., 28.  
 Graham, Ronald Lewis (葛立恒), 47.  
 Grid, 23.  
 Grimmett, Geoffrey Richard, 32.  
 Gumball machine problem, 31.
- HAKMEM, 28.  
 Harmonic numbers, fractional, 38.  
 Hashing, 9–10, 20.  
 Hoeffding, Wassily, 9, 15.  
   inequality, 9–10, 20.
- Importance sampling, 25.  
 Inclusion and exclusion, 31, 34, 44.  
 Incomplete beta function, 14.  
 Independent events, 2.  
 Independent random variables, 1, 7, 9,  
   10, 13–15, 20, 37.  
    $k$ -wise, 1, 13.  
 Infinite mean, 36, 38.  
 Information, bits of, 24.  
 Information gained, 25.  
 Integer multilinear representation, *see*  
   Reliability polynomials.  
 Internet, ii, iii.  
 Inverses, 36.  
 Itoh, Toshiya (伊東利哉), 43.
- Jabbour-Hattab, Jean (غسان جبور حطاب), 45.  
 Jaccard, Paul, 42.  
   index, 42–43.  
 James White, Phyllis Dorothy, 11.  
 Janson, Carl Svante, iv, 38, 40, 44.  
 Jensen, Johan Ludvig William Valdemar, 33.  
   inequality, 4, 16, 33, 38, 45.

- Johnson, David Stifter, 11.  
 Joint distribution, 13, 24, 35.  
 Joint entropy, 24.
- $k$ -cliques, 17.  
 $k$ -wise independence, 1, 13.  
 Kalai, Gil (גיל קלעי), 44.  
 Kallenberg, Olav Herbert, 37.  
 Karp, Richard Manning, 40.  
 Kasteleyn, Pieter Willem, 17.  
 Kendall, David George, 37.  
 Knuth, Donald Ervin (高德纳), i, iii,  
 iv, 37, 40, 44, 47.  
 Kolmogorov, Andrei Nikolaevich  
 (Колмогоров, Андрей Николаевич), 9,  
 inequality, 9.  
 Kullback, Solomon, 46.  
 divergence,  $D(y||x)$ , 24–25.
- La Vallée Poussin, Charles Jean Gustave  
 Nicolas de, 44.  
 Lake Wobegon dice, 12.  
 Large deviations, *see* Tail inequalities.  
 Larrie, Cora Mae, 21.  
 Least common multiple, 23.  
 Leibler, Richard Arthur, 46.  
 divergence,  $D(y||x)$ , 24–25.  
 Lipschitz, Rudolph Otto Sigismund,  
 condition, 10.  
 Loaded dice, 24.  
 Loop, running time of, 21.  
 Loops from a vertex to itself, 24.  
 Lord, Nicholas John, 30.  
 Lukács, Eugene (= Jenő), 28.
- MacQueen, James Buford, 38.  
 Magic masks, 28.  
 Mahler, Kurt, 28.  
 Markov (= Markoff), Andrei Andreevich  
 (Марков, Андрей Андреевич),  
 the elder, 4.  
 inequality, 4, 5, 16, 38, 39, 46.  
 Martingale differences, *see* Fair sequences.  
 Martingales, 6–11, 18–20, 24, 32.  
 with respect to a sequence, 7, 19, 37.  
 Max-flow min-cut theorem, 22, 42.  
 Maximal inequality, 8–9, 20.  
 McDiarmid, Colin John Hunter, 10.  
 Median value of a random variable, 14, 24.  
 Mengden, Nicolai Alexandrovitch von  
 (Менгденъ, Николай Александровичъ  
 фонъ), 25.  
 Method of bounded differences, 10.  
 Minhash algorithms, 43.  
 Minterms, 31.  
 Minwise independent permutations, 23.  
 Mitzenmacher, Michael David, 43, 44.  
 Moivre, Abraham de, 19, 37.
- Monotone Boolean functions, 5, 35.  
 Monotone Monte Carlo method, 41.  
 Montmort, Pierre Rémond de, 36.  
 Monus operation, 21–22.  
 Moraleda Oliván, Jorge Alfonso, 27.  
 Morse, Harold Calvin Marston, constant, 28.  
 Motwani, Rajeev (राजीव मोटवानी), 40.  
 MPR: Mathematical Preliminaries Redux, v.  
 Multigraphs, 24.  
 Multiplicatively fair sequences, 19.  
 Multivariate Bernoulli distribution,  
 14, 18, 20.  
 Multivariate total positivity, *see* FKG  
 inequality.  
 Mutual information, 24.
- NanoBingo, 12–13.  
 Nawrotzki, Kurt, 42.  
 Negative binomial distribution,  
 cumulative, 14.  
 Negatively correlated random variables,  
 18, 29.  
 Neumann, Peter, 30.  
 Neville-Neil, George Vernon, III, 50.  
 Newton, Isaac, 30.  
 Nonnegative submartingales, 9, 38.  
 Nonnegatively correlated random  
 variables, 17.  
 Nontransitive dice, 12.  
 NP-complete problems, 11.  
 Nullstellensatz, combinatorial, 23.
- One-sided estimates, 16.  
 Optional stopping principle, 8, 38.  
 Order ideals, 42.
- $\wp$  (power set, the family of all subsets), 34.  
 Pairwise independent random variables,  
 1, 13.  
 Paley, Raymond Edward Alan Christopher,  
 24, 45.  
 Paradoxes, 12, 13, 36.  
 Parity number, 28.  
 Parity of a binary integer, 13.  
 Partial ordering, 22.  
 Patashnik, Oren, 47.  
 Pearson, Karl (= Carl), 29.  
 Pi ( $\pi$ ), 14.  
 Pitman, James William, 30.  
 Pittel, Boris Gershon (Питтель, Борис  
 Гершенович), 40.  
 Playing cards, 1, 8, 14, 19.  
 Poisson, Siméon Denis, distribution,  
 15, 24, 35.  
 trials, 35.  
 Pólya, György (= George), 6, 19.  
 urn model, iv, 6–7, 19–20, 38.  
 Polynomials, 23; *see also* Chebyshev  
 polynomials, Reliability polynomials.



- Positively correlated random variables, 17, 28.
- Power series, 33.
- Prime implicants of a Boolean function, 5, 34.
- Probability estimates, 3–5, 8–9, 16.
- Probability generating functions, 15, 40.
- Probability spaces, 1–2.
- Propp, James Gary, 41.
- q.s.: Quite surely, 12, 20, 21.
- Quick, Jonathan Horatio, 18.
- Quite sure events, 12, 27.
- Raghavan, Prabhakar (பிரபாகர் ராகவன்), 40.
- Random bits, 2, 3, 5, 9, 13–15, 36.
- Random graphs, 16, 18.
- Random number generators, 41.
- Random permutations, 15.
- Random variables, 1–21.
- Random walk, 18.
  - coalescing, 21.
  - on  $r$ -cycle, 22, 42.
- Randomized algorithms, 21.
- Recurrence relations, 36, 41.
- Regular graphs and multigraphs, 24.
- Reliability polynomials, 5, 15, 16.
- Rémond de Montmort, Pierre, 36.
- Rényi, Alfréd, 34.
- Restricted growth strings, 15.
- Ross, Sheldon Mark, iv, 5, 33.
- Row sums, 22.
- Runs of a permutation, 37.
- $S_{\geq m}$  (a symmetric threshold function), 16.
- Samuels, Stephen Mitchell, 15, 31.
- Saturating addition and subtraction, 21–22.
- Savage, Richard Preston, Jr., 26.
- Say Red, 38.
- Schroepfel, Richard Crabtree, 28.
- Schulte-Geers, Ernst Franz Fred, iv, 13, 31, 39.
- Schwarz, Karl Hermann Amandus, inequality, 46.
- Second moment principle, 4, 16, 24, 33.
- Set partitions, 30, 32.
- Sets, represented as integers, 35.
- Shinozaki, Takahiro (篠崎隆宏), 43.
- Shuffles, 1.
- Sideways sum ( $\nu x$ ), 13.
- Sketches, 23.
- St. Petersburg paradox, 36.
- Stadje, Gert Wolfgang, 39.
- Stirzaker, David Robert, 32.
- Stopping rules, 7, 8, 19–20.
- Stork, David Geoffrey, 27.
- Strassen, Volker, 42.
- Stross, Charles David George, iii.
- Subadditive law, 11.
- Submartingales, 8–9, 20.
- Submodular set functions, 35.
- Subsequence of a martingale, 18.
- Summation by parts, 38.
- Supermartingales, 8, 40.
- Superpolynomially small, 12, 40.
- Supported sets, 17.
- Sylvester, James Joseph, 30.
- Symmetric Boolean functions, 16.
- $t$ -ary ballot numbers, 39.
- Tail inequalities, 4, 8–11, 15, 20, 21.
- Takei, Yoshinori (武井由智), 43.
- Taylor, Brook, formula, 20.
- Terpai, Tamás, 45.
- Thue, Axel, constant, 28.
- Totally uncorrelated sequences, 38.
- Trading tails, 46.
- Transcendental numbers, 28.
- Triangles (3-cliques), 16.
- Trybula, Stanislaw, 27.
- Uncorrelated sequences, 38.
- Uniform distribution, 1, 13, 16, 22–24, 36, 37, 41, 45.
- Uniform probing, 43.
- Union inequality, 14.
- Upfal, Eli (אלי אבפלי), 40.
- Urn models, 6–7, 18–20, 38.
- Usiskin, Zalman Philip, 27.
- Vallée Poussin, Charles Jean Gustave Nicolas de la, 44.
- Vandenberghe, Lieven Lodewijk André, 33.
- Vandermonde, Alexandre Théophile, matrix, 28.
- Variance, 2, 4, 9, 14, 35, 46.
- Vicious, Kode (pen name of George Vernon Neville-Neil III), vi.
- Volkov, Stanislav Evgenyevich (Волков, Станислав Евгеньевич), 45.
- von Mengden, Nicolai Alexandrovitch (Фонъ Менгденъ, Николай Александровичъ), 25.
- Wald, Abraham (= *Ábrahám*), 38.
  - equation, 20.
- Web pages, 43.
- whp (with high probability), *see a.s.*
- Wigderson, Avi (אבי ויגדורן), 40.
- Wilson, David Bruce, 41.
- Winkler, Peter Mann, 31.
- Zygmund, Antoni, 24, 45.

Note to readers:  
Please ignore these  
sidenotes; they're just  
hints to myself for  
preparing the index,  
and they're often flaky!

KNUTH

# THE ART OF COMPUTER PROGRAMMING

VOLUME 4    PRE-FASCICLE 5B

## INTRODUCTION TO BACKTRACKING

DONALD E. KNUTH *Stanford University*

ADDISON-WESLEY



November 12, 2016

Internet  
Stanford GraphBase  
MMIX

Internet page <http://www-cs-faculty.stanford.edu/~knuth/taocp.html> contains current information about this book and related books.

See also <http://www-cs-faculty.stanford.edu/~knuth/sgb.html> for information about *The Stanford GraphBase*, including downloadable software for dealing with the graphs used in many of the examples in Chapter 7.

See also <http://www-cs-faculty.stanford.edu/~knuth/mmixware.html> for downloadable software to simulate the MMIX computer.

Copyright © 2016 by Addison–Wesley

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher, except that the official electronic file may be used to print single copies for personal (not commercial) use.

Zeroth printing (revision -79), 12 November 2016

November 12, 2016

## PREFACE

*Begin at the beginning, and do not allow yourself to gratify  
a mere idle curiosity by dipping into the book, here and there.  
This would very likely lead to your throwing it aside,  
with the remark "This is much too hard for me!"  
and thus losing the chance of adding a very large item  
to your stock of mental delights.*

— LEWIS CARROLL, in *Symbolic Logic* (1896)

THIS BOOKLET contains draft material that I'm circulating to experts in the field, in hopes that they can help remove its most egregious errors before too many other people see it. I am also, however, posting it on the Internet for courageous and/or random readers who don't mind the risk of reading a few pages that have not yet reached a very mature state. *Beware:* This material has not yet been proofread as thoroughly as the manuscripts of Volumes 1, 2, 3, and 4A were at the time of their first printings. And alas, those carefully-checked volumes were subsequently found to contain thousands of mistakes.

Given this caveat, I hope that my errors this time will not be so numerous and/or obtrusive that you will be discouraged from reading the material carefully. I did try to make the text both interesting and authoritative, as far as it goes. But the field is vast; I cannot hope to have surrounded it enough to corral it completely. So I beg you to let me know about any deficiencies that you discover.

To put the material in context, this portion of fascicle 5 previews the opening pages of Section 7.2.2 of *The Art of Computer Programming*, entitled "Backtrack programming." The preceding section, 7.2.1, was about "Generating basic combinatorial patterns" —namely tuples, permutations, combinations, partitions, and trees. Now it's time to consider the non-basic patterns, the ones that have a much less uniform structure. For these we generally need to make tentative choices and then we need to back up when those choices need revision. Several subsections (7.2.2.1, 7.2.2.2, etc.) will follow this introductory material.

\* \* \*

The explosion of research in combinatorial algorithms since the 1970s has meant that I cannot hope to be aware of all the important ideas in this field. I've tried my best to get the story right, yet I fear that in many respects I'm woefully ignorant. So I beg expert readers to steer me in appropriate directions.

Please look, for example, at the exercises that I've classed as research problems (rated with difficulty level 46 or higher), namely exercises 14, . . . ; I've also implicitly mentioned or posed additional unsolved questions in the answers to exercises 6, 8, 42, 45, . . . . Are those problems still open? Please inform me if you know of a solution to any of these intriguing questions. And of course if no solution is known today but you do make progress on any of them in the future, I hope you'll let me know.

stamping  
Knuth

I urgently need your help also with respect to some exercises that I made up as I was preparing this material. I certainly don't like to receive credit for things that have already been published by others, and most of these results are quite natural "fruits" that were just waiting to be "plucked." Therefore please tell me if you know who deserves to be credited, with respect to the ideas found in exercises 31(b), 33, 44, 50, 51, 62, 66, 67, 75, 100, . . . . Furthermore I've credited exercises . . . to unpublished work of . . . . Have any of those results ever appeared in print, to your knowledge?

I've got a historical question too: Have you any idea who originated the idea of "stamping" in data structures? (See 7.2.2–(26). This concept is quite different from the so-called time stamps in persistent data structures, and quite different from the so-called time stamps in depth-first search algorithms, and quite different from the so-called time stamps in cryptology, although many programmers do use the name "time stamp" for those kinds of stamp.) It's a technique that I've seen often, in programs that have come to my attention during recent decades, but I wonder if it ever appeared in a book or paper that was published before, say, 1980.

\* \* \*

Special thanks are due to . . . for their detailed comments on my early attempts at exposition, as well as to numerous other correspondents who have contributed crucial corrections.

\* \* \*

I happily offer a "finder's fee" of \$2.56 for each error in this draft when it is first reported to me, whether that error be typographical, technical, or historical. The same reward holds for items that I forgot to put in the index. And valuable suggestions for improvements to the text are worth 32¢ each. (Furthermore, if you find a better solution to an exercise, I'll actually do my best to give you immortal glory, by publishing your name in the eventual book:—)

Cross references to yet-unwritten material sometimes appear as '00'; this impossible value is a placeholder for the actual numbers to be supplied later.

Happy reading!

*Stanford, California*  
*99 Umbruary 2016*

D. E. K.

MPR

## Part of the Preface to Volume 4B

During the years that I've been preparing Volume 4, I've often run across basic techniques of probability theory that I would have put into Section 1.2 of Volume 1 if I'd been clairvoyant enough to anticipate them in the 1960s. Finally I realized that I ought to collect most of them together in one place, near the beginning of Volume 4B, because the story of these developments is too interesting to be broken up into little pieces scattered here and there.


Therefore this volume begins with a special section entitled "Mathematical Preliminaries Redux," and future sections use the abbreviation 'MPR' to refer to its equations and its exercises.

## MATHEMATICAL PRELIMINARIES REDUX

MANY PARTS of this book deal with *discrete probabilities*, namely with a finite or countably infinite set  $\Omega$  of atomic events  $\omega$ , each of which has a given probability  $\Pr(\omega)$ , where

$$0 \leq \Pr(\omega) \leq 1 \quad \text{and} \quad \sum_{\omega \in \Omega} \Pr(\omega) = 1. \quad (1)$$

...

 For the complete text of the special MPR section, please see Pre-Fascicle 5a. Incidentally, Section 7.2.2 intentionally begins on a left-hand page, and its illustrations are numbered beginning with Fig. 68, because Section 7.2.1 ended on a right-hand page and its final illustration was Fig. 67. The editor has decided to treat Chapter 7 as a single unit, even though it will be split across several physical volumes.

|   |  |
|---|--|
| <p><i>Nowhere to go but out,<br/>Nowhere to come but back.</i><br/>— BEN KING, in <i>The Sum of Life</i> (c. 1893)<br/><i>When you come to one legal road that's blocked,<br/>you back up and try another.</i><br/>— PERRY MASON, in <i>The Case of the Black-Eyed Blonde</i> (1944)<br/><i>No one I think is in my tree.</i><br/>— JOHN LENNON, in <i>Strawberry Fields Forever</i> (1967)</p> | <p>KING<br/>MASON<br/>Gardner ES<br/>LENNON<br/>backtrack<br/>Walker<br/>domain<br/>cutoff<br/>properties: logical propositions (relations)<br/><math>P_0()</math><br/>lexicographically</p> |
|---|--|

### 7.2.2. Backtrack Programming

Now that we know how to generate simple combinatorial patterns such as tuples, permutations, combinations, partitions, and trees, we're ready to tackle more exotic patterns that have subtler and less uniform structure. Instances of almost *any* desired pattern can be generated systematically, at least in principle, if we organize the search carefully. Such a method was christened "backtrack" by R. J. Walker in the 1950s, because it is basically a way to examine all fruitful possibilities while exiting gracefully from situations that have been fully explored.

Most of the patterns we shall deal with can be cast in a simple, general framework: We seek all sequences  $x_1x_2 \dots x_n$  for which some property  $P_n(x_1, x_2, \dots, x_n)$  holds, where each item  $x_k$  belongs to some given domain  $D_k$  of integers. The backtrack method, in its most elementary form, consists of inventing intermediate "cutoff" properties  $P_l(x_1, \dots, x_l)$  for  $1 \leq l < n$ , such that

$$P_l(x_1, \dots, x_l) \text{ is true whenever } P_{l+1}(x_1, \dots, x_{l+1}) \text{ is true;} \quad (1)$$

$$P_l(x_1, \dots, x_l) \text{ is fairly easy to test, if } P_{l-1}(x_1, \dots, x_{l-1}) \text{ holds.} \quad (2)$$

(We assume that  $P_0()$  is always true. Exercise 1 shows that all of the basic patterns studied in Section 7.2.1 can easily be formulated in terms of domains  $D_k$  and cutoff properties  $P_l$ .) Then we can proceed lexicographically as follows:

**Algorithm B** (*Basic backtrack*). Given domains  $D_k$  and properties  $P_l$  as above, this algorithm visits all sequences  $x_1x_2 \dots x_n$  that satisfy  $P_n(x_1, x_2, \dots, x_n)$ .

- B1.** [Initialize.] Set  $l \leftarrow 1$ , and initialize the data structures needed later.
- B2.** [Enter level  $l$ .] (Now  $P_{l-1}(x_1, \dots, x_{l-1})$  holds.) If  $l > n$ , visit  $x_1x_2 \dots x_n$  and go to B5. Otherwise set  $x_l \leftarrow \min D_l$ , the smallest element of  $D_l$ .
- B3.** [Try  $x_l$ .] If  $P_l(x_1, \dots, x_l)$  holds, update the data structures to facilitate testing  $P_{l+1}$ , set  $l \leftarrow l + 1$ , and go to B2.
- B4.** [Try again.] If  $x_l \neq \max D_l$ , set  $x_l$  to the next larger element of  $D_l$  and return to B3.
- B5.** [Backtrack.] Set  $l \leftarrow l - 1$ . If  $l > 0$ , downdate the data structures by undoing the changes recently made in step B3, and return to B4. (Otherwise stop.) ■

The main point is that if  $P_l(x_1, \dots, x_l)$  is false in step B3, we needn't waste time trying to append any further values  $x_{l+1} \dots x_n$ . Thus we can often rule out huge regions of the space of all potential solutions. A second important point is that very little memory is needed, although there may be many, many solutions.



For example, let's consider the classic *problem of  $n$  queens*: In how many ways can  $n$  queens be placed on an  $n \times n$  board so that no two are in the same row, column, or diagonal? We can suppose that one queen is in each row, and that the queen in row  $k$  is in column  $x_k$ , for  $1 \leq k \leq n$ . Then each domain  $D_k$  is  $\{1, 2, \dots, n\}$ ; and  $P_n(x_1, \dots, x_n)$  is the condition that

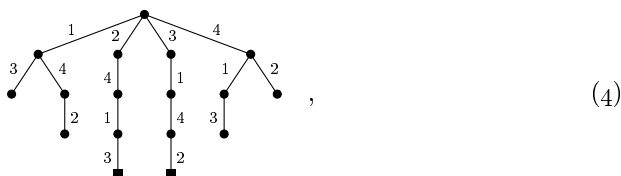
$$x_j \neq x_k \quad \text{and} \quad |x_k - x_j| \neq k - j, \quad \text{for } 1 \leq j < k \leq n. \quad (3)$$

(If  $x_j = x_k$  and  $j < k$ , two queens are in the same column; if  $|x_k - x_j| = k - j$ , they're in the same diagonal.)

This problem is easy to set up for Algorithm B, because we can let property  $P_l(x_1, \dots, x_l)$  be the same as (3) but restricted to  $1 \leq j < k \leq l$ . Condition (1) is clear; and so is condition (2), because  $P_l$  requires testing (3) only for  $k = l$  when  $P_{l-1}$  is known. Notice that  $P_1(x_1)$  is always true in this example.

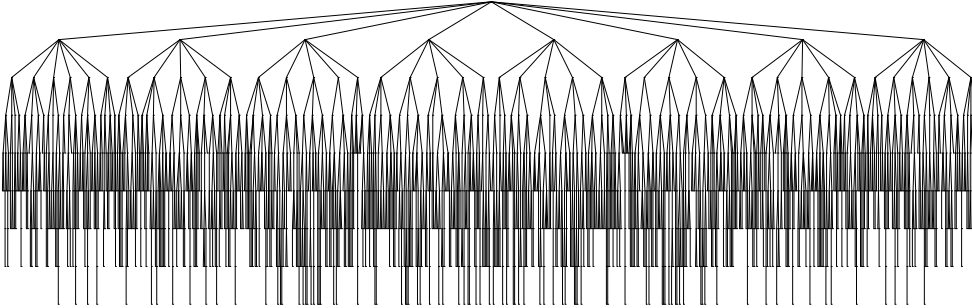
One of the best ways to learn about backtracking is to execute Algorithm B by hand in the special case  $n = 4$  of the  $n$  queens problem: First we set  $x_1 \leftarrow 1$ . Then when  $l = 2$  we find  $P_2(1, 1)$  and  $P_2(1, 2)$  false; hence we don't get to  $l = 3$  until trying  $x_2 \leftarrow 3$ . Then, however, we're stuck, because  $P_3(1, 3, x)$  is false for  $1 \leq x \leq 4$ . Backtracking to level 2, we now try  $x_2 \leftarrow 4$ ; and this allows us to set  $x_3 \leftarrow 2$ . However, we're stuck again, at level 4; and this time we must back up all the way to level 1, because there are no further valid choices at levels 3 and 2. The next choice  $x_1 \leftarrow 2$  does, happily, lead to a solution without much further ado, namely  $x_1 x_2 x_3 x_4 = 2413$ . And one more solution (3142) turns up before the algorithm terminates.

The behavior of Algorithm B is nicely visualized as a tree structure, called a search tree or *backtrack tree*. For example, the backtrack tree for the four queens problem has just 17 nodes,



corresponding to the 17 times step B2 is performed. Here  $x_l$  is shown as the label of an edge from level  $l - 1$  to level  $l$  of the tree. (Level  $l$  of the algorithm actually corresponds to the tree's level  $l - 1$ , because we've chosen to represent patterns using subscripts from 1 to  $n$  instead of from 0 to  $n - 1$  in this discussion.) The *profile*  $(p_0, p_1, \dots, p_n)$  of this particular tree—the number of nodes at each level—is  $(1, 4, 6, 4, 2)$ ; and we see that the number of solutions,  $p_n = p_4$ , is 2.

Figure 68 shows the corresponding tree when  $n = 8$ . This tree has 2057 nodes, distributed according to the profile  $(1, 8, 42, 140, 344, 568, 550, 312, 92)$ . Thus the early cutoffs facilitated by backtracking have allowed us to find all 92 solutions by examining only 0.01% of the  $8^8 = 16,777,216$  possible sequences  $x_1 \dots x_8$ . (And  $8^8$  is only 0.38% of the  $\binom{64}{8} = 4,426,165,368$  ways to put eight queens on the board.)



data structures-  
mems  
downdating vs updating+  
undoes

**Fig. 68.** The problem of placing eight nonattacking queens has this backtrack tree.

Notice that, in this case, Algorithm B spends most of its time in the vicinity of level 5. Such behavior is typical: The backtrack tree for  $n = 16$  queens has 1,141,190,303 nodes, and its profile is (1, 16, 210, 2236, 19688, 141812, 838816, 3998456, 15324708, 46358876, 108478966, 193892860, 260303408, 253897632, 171158018, 72002088, 14772512), concentrated near level 12.

**Data structures.** Backtrack programming is often used when a *huge* tree of possibilities needs to be examined. Thus we want to be able to test property  $P_l$  as quickly as possible in step B3.

One way to implement Algorithm B for the  $n$  queens problem is to avoid auxiliary data structures and simply to make a bunch of sequential comparisons in that step: “Is  $x_l - x_j \in \{j - l, 0, l - j\}$  for some  $j < l$ ?” Assuming that we access memory whenever referring to  $x_j$ , given a trial value  $x_l$  in a register, such an implementation performs approximately 112 billion memory accesses when  $n = 16$ ; that’s about 98 mems per node.

We can do better by introducing three simple arrays. Property  $P_l$  in (3) says essentially that the numbers  $x_k$  are distinct, and so are the numbers  $x_k + k$ , and so are the numbers  $x_k - k$ . Therefore we can use auxiliary Boolean arrays  $a_1 \dots a_n$ ,  $b_1 \dots b_{2n-1}$ , and  $c_1 \dots c_{2n-1}$ , where  $a_j$  means ‘some  $x_k = j$ ’,  $b_j$  means ‘some  $x_k + k - 1 = j$ ’, and  $c_j$  means ‘some  $x_k - k + n = j$ ’. Those arrays are readily updated and downdated if we customize Algorithm B as follows:

**B1\*.** [Initialize.] Set  $a_1 \dots a_n \leftarrow 0 \dots 0$ ,  $b_1 \dots b_{2n-1} \leftarrow 0 \dots 0$ ,  $c_1 \dots c_{2n-1} \leftarrow 0 \dots 0$ , and  $l \leftarrow 1$ .

**B2\*.** [Enter level  $l$ .] (Now  $P_{l-1}(x_1, \dots, x_{l-1})$  holds.) If  $l > n$ , visit  $x_1 x_2 \dots x_n$  and go to B5\*. Otherwise set  $t \leftarrow 1$ .

**B3\*.** [Try  $t$ .] If  $a_t = 1$  or  $b_{t+l-1} = 1$  or  $c_{t-l+n} = 1$ , go to B4\*. Otherwise set  $a_t \leftarrow 1$ ,  $b_{t+l-1} \leftarrow 1$ ,  $c_{t-l+n} \leftarrow 1$ ,  $x_l \leftarrow t$ ,  $l \leftarrow l + 1$ , and go to B2\*.

**B4\*.** [Try again.] If  $t < n$ , set  $t \leftarrow t + 1$  and return to B3\*.

**B5\*.** [Backtrack.] Set  $l \leftarrow l - 1$ . If  $l > 0$ , set  $t \leftarrow x_l$ ,  $c_{t-l+n} \leftarrow 0$ ,  $b_{t+l-1} \leftarrow 0$ ,  $a_t \leftarrow 0$ , and return to B4\*. (Otherwise stop.) ■

Notice how step B5\* neatly undoes the updates that step B3\* had made, in the reverse order. Reverse order for downdating is typical of backtrack algorithms,

although there is some flexibility; we could, for example, have restored  $a_t$  before  $b_{t+l-1}$  and  $c_{t-l+n}$ , because those arrays are independent.

The auxiliary arrays  $a, b, c$  make it easy to test property  $P_l$  at the beginning of step B3\*, but we must also access memory when we update them and downgrade them. Does that cost us more than it saves? Fortunately, no: The running time for  $n = 16$  goes down to about 34 billion mems, roughly 30 mems per node.

Furthermore we could keep the bit vectors  $a, b, c$  entirely in registers, on a machine with 64-bit registers, assuming that  $n \leq 32$ . Then there would be just two memory accesses per node, namely to store  $x_l \leftarrow t$  and later to fetch  $t \leftarrow x_l$ ; however, quite a lot of in-register computation would become necessary.

**Walker's method.** The 1950s-era programs of R. J. Walker organized backtracking in a somewhat different way. Instead of letting  $x_l$  run through all elements of  $D_l$ , he calculated and stored the set

$$S_l \leftarrow \{x \in D_l \mid P_l(x_1, \dots, x_{l-1}, x) \text{ holds}\} \quad (5)$$

upon entry to each node at level  $l$ . This computation can often be done efficiently all at once, instead of piecemeal, because some cutoff properties make it possible to combine steps that would otherwise have to be repeated for each  $x \in D_l$ . In essence, he used the following variant of Algorithm B:

**Algorithm W** (*Walker's backtrack*). Given domains  $D_k$  and cutoffs  $P_l$  as above, this algorithm visits all sequences  $x_1 x_2 \dots x_n$  that satisfy  $P_n(x_1, x_2, \dots, x_n)$ .

**W1.** [Initialize.] Set  $l \leftarrow 1$ , and initialize the data structures needed later.

**W2.** [Enter level  $l$ .] (Now  $P_{l-1}(x_1, \dots, x_{l-1})$  holds.) If  $l > n$ , visit  $x_1 x_2 \dots x_n$  and go to W4. Otherwise determine the set  $S_l$  as in (5).

**W3.** [Try to advance.] If  $S_l$  is nonempty, set  $x_l \leftarrow \min S_l$ , update the data structures to facilitate computing  $S_{l+1}$ , set  $l \leftarrow l + 1$ , and go to W2.

**W4.** [Backtrack.] Set  $l \leftarrow l - 1$ . If  $l > 0$ , downgrade the data structures by undoing changes made in step W3, set  $S_l \leftarrow S_l \setminus x_l$ , and retreat to W3. ■

Walker applied this method to the  $n$  queens problem by computing  $S_l = U \setminus A_l \setminus B_l \setminus C_l$ , where  $U = D_l = \{1, \dots, n\}$  and

$$A_l = \{x_j \mid 1 \leq j < l\}, B_l = \{x_j + j - l \mid 1 \leq j < l\}, C_l = \{x_j - j + l \mid 1 \leq j < l\}. \quad (6)$$

He represented these auxiliary sets by bit vectors  $a, b, c$ , analogous to (but different from) the bit vectors of Algorithm B\* above. Exercise 9 shows that the updating in step W3 is easy, using bitwise operations on  $n$ -bit numbers; furthermore, no downdating is needed in step W4. The corresponding run time when  $n = 16$  turns out to be just 9.1 gigamems, or 8 mems per node.

Let  $Q(n)$  be the number of solutions to the  $n$  queens problem. Then we have

$$\begin{array}{cccccccccccccccc} n & = & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ Q(n) & = & 1 & 1 & 0 & 0 & 2 & 10 & 4 & 40 & 92 & 352 & 724 & 2680 & 14200 & 73712 & 365596 & 2279184 & 14772512 \end{array}$$

and the values for  $n \leq 11$  were computed independently by several people during the nineteenth century. Small cases were relatively easy; but when T. B. Sprague

registers  
Walker  
cutoff properties  
visits  
bitwise operations  
historical notes+  
Sprague

had finished computing  $Q(11)$  he remarked that “This was a very heavy piece of work, and occupied most of my leisure time for several months. . . . It will, I imagine, be scarcely possible to obtain results for larger boards, unless a number of persons co-operate in the work.” [See *Proc. Edinburgh Math. Soc.* **17** (1899), 43–68; Sprague was the leading actuary of his day.] Nevertheless, H. Onnen went on to evaluate  $Q(12) = 14,200$ —an astonishing feat of hand calculation—in 1910. [See W. Ahrens, *Math. Unterhaltungen und Spiele* **2**, second edition (1918), 344.]

All of these hard-won results were confirmed in 1960 by R. J. Walker, using the SWAC computer at UCLA and the method of exercise 9. Walker also computed  $Q(13)$ ; but he couldn’t go any further with the machine available to him at the time. The next step,  $Q(14)$ , was computed by Michael D. Kennedy at the University of Tennessee in 1963, commandeering an IBM 1620 for 120 hours. S. R. Bunch evaluated  $Q(15)$  in 1974 at the University of Illinois, using about two hours on an IBM System 360-75; then J. R. Bitner found  $Q(16)$  after about three hours on the same computer, but with an improved method.

Computers and algorithms have continued to get better, of course, and such results are now obtained almost instantly. Hence larger and larger values of  $n$  lie at the frontier. The whopping value  $Q(27) = 234,907,967,154,122,528$ , found in 2016 by Thomas B. Preußer and Matthias R. Engelhardt, probably won’t be exceeded for awhile! [See *J. Signal Processing Systems* **89** (2017), to appear. This distributed computation occupied a dynamic cluster of diverse FPGA devices for 383 days; those devices provided a total peak of more than 7000 custom-designed hardware solvers to handle 2,024,110,796 independent subproblems.]

**Permutations and Langford pairs.** Every solution  $x_1 \dots x_n$  to the  $n$  queens problem is a permutation of  $\{1, \dots, n\}$ , and many other problems are permutation-based. Indeed, we’ve already seen Algorithm 7.2.1.2X, which is an elegant backtrack procedure specifically designed for special kinds of permutations. When that algorithm begins to choose the value of  $x_l$ , it makes all of the appropriate elements  $\{1, 2, \dots, n\} \setminus \{x_1, \dots, x_{l-1}\}$  conveniently accessible in a linked list.

We can get further insight into such data structures by returning to the problem of Langford pairs, which was discussed at the very beginning of Chapter 7. That problem can be reformulated as the task of finding all permutations of  $\{1, 2, \dots, n\} \cup \{-1, -2, \dots, -n\}$  with the property that

$$x_j = k \text{ implies } x_{j+k+1} = -k, \quad \text{for } 1 \leq j \leq 2n \text{ and } 1 \leq k \leq n. \quad (7)$$

For example, when  $n = 4$  there are two solutions, namely  $234\bar{2}1\bar{3}\bar{1}\bar{4}$  and  $413\bar{1}2\bar{4}\bar{3}\bar{2}$ . (As usual we find it convenient to write  $\bar{1}$  for  $-1$ ,  $\bar{2}$  for  $-2$ , etc.) Notice that if  $x = x_1 x_2 \dots x_{2n}$  is a solution, so is its “dual”  $-x^R = (-x_{2n}) \dots (-x_2)(-x_1)$ .

Here’s a Langford-inspired adaptation of Algorithm 7.2.1.2X, with the former notation modified slightly to match Algorithms B and W: We want to maintain pointers  $p_0 p_1 \dots p_n$  such that, if the positive integers not already present in  $x_1 \dots x_{l-1}$  are  $k_1 < k_2 < \dots < k_t$  when we’re choosing  $x_l$ , we have the linked list

$$p_0 = k_1, p_{k_1} = k_2, \dots, p_{k_{t-1}} = k_t, p_{k_t} = 0. \quad (8)$$

Such a condition turns out to be easy to maintain.

Sprague  
Onnen  
Ahrens  
Walker  
SWAC computer  
UCLA  
Kennedy  
University of Tennessee  
IBM 1620  
Bunch  
University of Illinois  
IBM System 360-75  
Bitner  
Preußer  
Engelhardt  
distributed computation  
FPGA  
permutation  
data structures  
Langford pairs  
dual  
linked list

**Algorithm L** (*Langford pairs*). This algorithm visits all solutions  $x_1 \dots x_{2n}$  to (7) in lexicographic order, using pointers  $p_0 p_1 \dots p_n$  that satisfy (8), and also using an auxiliary array  $y_1 \dots y_{2n}$  for backtracking.

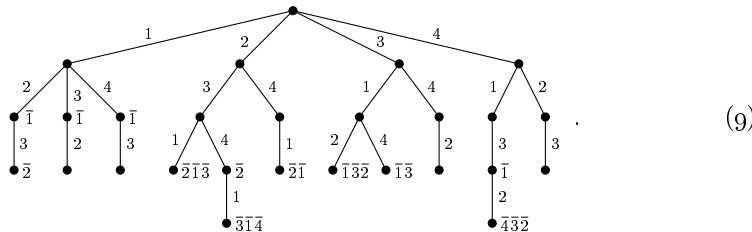
undoes  
deletion operation  
dancing links  
search tree  
cutoff principle

- L1.** [Initialize.] Set  $x_1 \dots x_{2n} \leftarrow 0 \dots 0$ ,  $p_k \leftarrow k+1$  for  $0 \leq k < n$ ,  $p_n \leftarrow 0$ ,  $l \leftarrow 1$ .
- L2.** [Enter level  $l$ .] Set  $k \leftarrow p_0$ . If  $k = 0$ , visit  $x_1 x_2 \dots x_{2n}$  and go to L5. Otherwise set  $j \leftarrow 0$ , and while  $x_l < 0$  set  $l \leftarrow l+1$ .
- L3.** [Try  $x_l = k$ .] (At this point we have  $k = p_j$ .) If  $l+k+1 > 2n$ , go to L5. Otherwise, if  $x_{l+k+1} = 0$ , set  $x_l \leftarrow k$ ,  $x_{l+k+1} \leftarrow -k$ ,  $y_l \leftarrow j$ ,  $p_j \leftarrow p_k$ ,  $l \leftarrow l+1$ , and return to L2.
- L4.** [Try again.] (We've found all solutions that begin with  $x_1 \dots x_{l-1} k$  or something smaller.) Set  $j \leftarrow k$  and  $k \leftarrow p_j$ , then go to L3 if  $k \neq 0$ .
- L5.** [Backtrack.] Set  $l \leftarrow l-1$ . If  $l > 0$  do the following: While  $x_l < 0$ , set  $l \leftarrow l-1$ . Then set  $k \leftarrow x_l$ ,  $x_l \leftarrow 0$ ,  $x_{l+k+1} \leftarrow 0$ ,  $j \leftarrow y_l$ ,  $p_j \leftarrow k$ , and go back to L4. Otherwise terminate the algorithm. ■

Careful study of these steps will reveal how everything fits together nicely. Notice that, for example, step L3 removes  $k$  from the linked list (8) by simply setting  $p_j \leftarrow p_k$ . That step also sets  $x_{l+k+1} \leftarrow -k$ , in accordance with (7), so that we can skip over position  $l+k+1$  when we encounter it later in step L2.

The main point of Algorithm L is the somewhat subtle way in which step L5 undoes the deletion operation by setting  $p_j \leftarrow k$ . The pointer  $p_k$  still retains the appropriate link to the next element in the list, *because  $p_k$  has not been changed by any of the intervening updates*. (Think about it.) This is the germ of an idea called "dancing links" that we will explore in Section 7.2.2.1.

To draw the search tree corresponding to a run of Algorithm L, we can label the edges with the positive choices of  $x_l$  as we did in (4), while labeling the nodes with any previously set negative values that are passed over in step L2. For instance the tree for  $n = 4$  is



Solutions appear at depth  $n$  in this tree, even though they involve  $2n$  values  $x_1 x_2 \dots x_{2n}$ .

Algorithm L sometimes makes false starts and doesn't realize the problem until probing further than necessary. Notice that the value  $x_l = k$  can appear only when  $l+k+1 \leq 2n$ ; hence if we haven't seen  $k$  by the time  $l$  reaches  $2n-k-1$ , we're *forced* to choose  $x_l = k$ . For example, the branch 12 $\bar{1}$  in (9) needn't be pursued, because 4 must appear in  $\{x_1, x_2, x_3\}$ . Exercise 20 explains how to incorporate this cutoff principle into Algorithm L. When  $n = 17$ , it reduces the number of nodes in the search tree from 1.29 trillion to 330 billion,

and reduces the running time from 25.0 teramems to 8.1 teramems. (The amount of work has gone from 19.4 mems per node to 24.4 mems per node, because of the extra tests for cutoffs, yet there’s a significant overall reduction.)

Furthermore, we can “break the symmetry” by ensuring that we don’t consider both a solution and its dual. This idea, exploited in exercise 21, reduces the search tree to just 160 billion nodes and costs just 3.94 teramems—that’s 24.6 mems per node.

**Word rectangles.** Let’s look next at a problem where the search domains  $D_l$  are much larger. An  $m \times n$  *word rectangle* is an array of  $n$ -letter words\* whose columns are  $m$ -letter words. For example,

$$\begin{array}{l} \text{status} \\ \text{lowest} \\ \text{utopia} \\ \text{making} \\ \text{sledge} \end{array} \quad (10)$$

is a  $5 \times 6$  word rectangle whose columns all belong to WORDS(5757), the collection of 5-letter words in the Stanford GraphBase. To find such patterns, we can suppose that column  $l$  contains the  $x_l$ th most common 5-letter word, where  $1 \leq x_l \leq 5757$  for  $1 \leq l \leq 6$ ; hence there are  $5757^6 = 36,406,369,848,837,732,146,649$  ways to choose the columns. In (10) we have  $x_1 \dots x_6 = 1446 \ 185 \ 1021 \ 2537 \ 66 \ 255$ . Of course very few of those choices will yield suitable rows; but backtracking will hopefully help us to find all solutions in a reasonable amount of time.

We can set this problem up for Algorithm B by storing the  $n$ -letter words in a trie (see Section 6.3), with one trie node of size 26 for each  $l$ -letter prefix of a legitimate word,  $0 \leq l \leq n$ .

For example, such a trie for  $n = 6$  represents 15727 words with 23667 nodes. The prefix *st* corresponds to node number 260, whose 26 entries are

$$(484,0,0,0,1589,0,0,0,2609,0,0,0,0,1280,0,0,251,0,0,563,0,0,0,1621,0); \quad (11)$$

this means that *sta* is node 484, *ste* is node 1589, . . . , *sty* is node 1621, and there are no 6-letter words beginning with *stb*, *stc*, . . . , *stx*, *stz*. A slightly different convention is used for prefixes of length  $n - 1$ ; for example, the entries for node 580, ‘*corne*’, are

$$(3879,0,0,3878,0,0,0,0,0,0,9602,0,0,0,0,0,171,0,5013,0,0,0,0,0,0), \quad (12)$$

meaning that *cornea*, *corned*, *cornel*, *corner*, and *cornet* are ranked 3879, 3878, 9602, 171, and 5013 in the list of 6-letter words.

\* Whenever five-letter words are used in the examples of this book, they’re taken from the 5757 Stanford GraphBase words as explained at the beginning of Chapter 7. Words of other lengths are taken from the *The Official SCRABBLE® Players Dictionary*, fourth edition (Hasbro, 2005), because those words have been incorporated into many widely available computer games. Such words have been ranked according to the British National Corpus of 2007—where ‘*the*’ occurs 5,405,633 times and the next-most common word, ‘*of*’, occurs roughly half as often (3,021,525). The OSPD4 list includes respectively (101, 1004, 4002, 8887, 15727, 23958, 29718, 29130, 22314, 16161, 11412) words of lengths (2, 3, . . . , 12), of which (97, 771, 2451, 4474, 6910, 8852, 9205, 8225, 6626, 4642, 3061) occur at least six times in the British National Corpus.

break the symmetry  
dual  
word rectangles—  
footnote doesn’t show up here  
 $n$ -letter words  
WORDS  
5-letter words  
Stanford GraphBase  
trie

Suppose  $x_1$  and  $x_2$  specify the 5-letter column-words `slums` and `total` as in (10). Then the trie tells us that the next column-word  $x_3$  must have the form  $c_1c_2c_3c_4c_5$  where  $c_1 \in \{\mathbf{a}, \mathbf{e}, \mathbf{i}, \mathbf{o}, \mathbf{r}, \mathbf{u}, \mathbf{y}\}$ ,  $c_2 \notin \{\mathbf{e}, \mathbf{h}, \mathbf{j}, \mathbf{k}, \mathbf{y}, \mathbf{z}\}$ ,  $c_3 \in \{\mathbf{e}, \mathbf{m}, \mathbf{o}, \mathbf{t}\}$ ,  $c_4 \notin \{\mathbf{a}, \mathbf{b}, \mathbf{o}\}$ , and  $c_5 \in \{\mathbf{a}, \mathbf{e}, \mathbf{i}, \mathbf{o}, \mathbf{u}, \mathbf{y}\}$ . (There are 221 such words.)

Let  $a_{l1} \dots a_{lm}$  be the trie nodes corresponding to the prefixes of the first  $l$  columns of a partial solution to the word rectangle problem. This auxiliary array enables Algorithm B to find all solutions, as explained in exercise 24. It turns out that there are exactly 625,415 valid  $5 \times 6$  word rectangles, according to our conventions; and the method of exercise 24 needs about 19 teramems of computation to find them all. In fact, the profile of the search tree is

$$(1, 5757, 2458830, 360728099, 579940198, 29621728, 625415), \quad (13)$$

indicating for example that just 360,728,099 of the  $5757^3 = 190,804,533,093$  choices for  $x_1x_2x_3$  will lead to valid prefixes of 6-letter words.

With care, exercise 24's running time can be significantly decreased, once we realize that every node of the search tree for  $1 \leq l < n$  requires testing 5757 possibilities for  $x_l$  in step B3. If we build a more elaborate data structure for the 5-letter words, so that it becomes easy to run through all words that have a specific letter in a specific position, we can refine the algorithm so that the average number of possibilities per level that need to be investigated becomes only

$$(5757.0, 1697.9, 844.1, 273.5, 153.5, 100.8); \quad (14)$$

the total running time then drops to 1.15 teramems. Exercise 25 has the details. And exercise 28 discusses a method that's faster yet.

**Commafree codes.** Our next example deals entirely with *four*-letter words. But it's not obscene; it's an intriguing question of coding theory. The problem is to find a set of four-letter words that can be decoded even if we don't put spaces or other delimiters between them. If we take any message that's formed from words of the set by simply concatenating them together, `likethis`, and if we look at any seven consecutive letters  $\dots x_1x_2x_3x_4x_5x_6x_7 \dots$ , exactly one of the four-letter substrings  $x_1x_2x_3x_4$ ,  $x_2x_3x_4x_5$ ,  $x_3x_4x_5x_6$ ,  $x_4x_5x_6x_7$  will be a codeword. Equivalently, if  $x_1x_2x_3x_4$  and  $x_5x_6x_7x_8$  are codewords, then  $x_2x_3x_4x_5$  and  $x_3x_4x_5x_6$  and  $x_4x_5x_6x_7$  aren't. (For example, `iket` isn't.) Such a set is called a "commafree code" or a "self-synchronizing block code" of length four.

Commafree codes were introduced by F. H. C. Crick, J. S. Griffith, and L. E. Orgel [*Proc. National Acad. Sci.* **43** (1957), 416–421], and studied further by S. W. Golomb, B. Gordon, and L. R. Welch [*Canadian Journal of Mathematics* **10** (1958), 202–209], who considered the general case of  $m$ -letter alphabets and  $n$ -letter words. They constructed optimum commafree codes for all  $m$  when  $n = 2, 3, 5, 7, 9, 11, 13$ , and 15; and optimum codes for all  $m$  were subsequently found also for  $n = 17, 19, 21, \dots$  (see exercise 32). We will focus our attention on the four-letter case here ( $n = 4$ ), partly because that case is still very far from being resolved, but mostly because the task of finding such codes is especially instructive. Indeed, our discussion will lead us naturally to an understanding of several significant techniques that are important for backtrack programming in general.

teramems  
profile  
search tree  
data structure  
commafree codes–  
4-letter codewords  
four-letter words  
coding theory  
concatenating  
codewords  
self-synchronizing block code  
block code  
Crick  
Griffith  
Orgel  
Golomb  
Gordon  
Welch

To begin, we can see immediately that a commafree codeword cannot be “periodic,” like `dodo` or `gaga`. Such a word already appears within two adjacent copies of itself. Thus we’re restricted to *aperiodic* words like `item`, of which there are  $m^4 - m^2$ . Notice further that if `item` has been chosen, we aren’t allowed to include any of its cyclic shifts `temi`, `emit`, or `mite`, because they all appear within `itemitem`. Hence the maximum number of codewords in our commafree code cannot exceed  $(m^4 - m^2)/4$ .

For example, consider the binary case,  $m = 2$ , when this maximum is 3. Can we choose three four-bit “words,” one from each of the cyclic classes

$$\begin{aligned} [0001] &= \{0001, 0010, 0100, 1000\}, \\ [0011] &= \{0011, 0110, 1100, 1001\}, \\ [0111] &= \{0111, 1110, 1101, 1011\}, \end{aligned} \tag{15}$$

so that the resulting code is commafree? Yes: One solution in this case is simply to choose the smallest word in each class, namely 0001, 0011, and 0111. (Alert readers will recall that we studied the smallest word in the cyclic class of *any* aperiodic string in Section 7.2.1.1, where such words were called *prime strings* and where some of the remarkable properties of prime strings were proved.)

That trick doesn’t work when  $m = 3$ , however, when there are  $(81 - 9)/4 = 18$  cyclic classes. Then we cannot include 1112 after we’ve chosen 0001 and 0011. Indeed, a code that contains 0001 and 1112 can’t contain either 0011 or 0111.

We could systematically backtrack through 18 levels, choosing  $x_1$  in [0001] and  $x_2$  in [0011], etc., and rejecting each  $x_l$  as in Algorithm B whenever we discover that  $\{x_1, x_2, \dots, x_l\}$  isn’t commafree. For example, if  $x_1 = 0010$  and we try  $x_2 = 1001$ , this approach would backtrack because  $x_1$  occurs inside  $x_2x_1$ .

But a naïve strategy of that kind, which recognizes failure only after a bad choice has been made, can be vastly improved. If we had been clever enough, we could have looked a little bit ahead, and never even considered the choice  $x_2 = 1001$  in the first place. Indeed, after choosing  $x_1 = 0010$ , we can automatically exclude *all* further words of the form  $*001$ , such as 2001 when  $m \geq 3$  and 3001 when  $m \geq 4$ .

Even better pruning occurs if, for example, we’ve chosen  $x_1 = 0001$  and  $x_2 = 0011$ . Then we can immediately rule out all words of the forms  $1***$  or  $***0$ , because  $x_11***$  includes  $x_2$  and  $***0x_2$  includes  $x_1$ . Already we could then deduce, in the case  $m \geq 3$ , that classes [0002], [0021], [0111], [0211], and [1112] *must* be represented by 0002, 0021, 0111, 0211, and 2111, respectively; each of the other three possibilities in those classes has been wiped out!

Thus we see the desirability of a lookahead mechanism.

**Dynamic ordering of choices.** Furthermore, we can see from this example that it’s not always good to choose  $x_1$ , then  $x_2$ , then  $x_3$ , and so on when trying to satisfy a general property  $P_n(x_1, x_2, \dots, x_n)$  in the setting of Algorithm B. Maybe the search tree will be much smaller if we first choose  $x_5$ , say, and then turn next to some other  $x_j$ , depending on the particular value of  $x_5$  that was selected. Some orderings might have much better cutoff properties than others, and every branch of the tree is free to choose its variables in any desired order.

periodic  
aperiodic  
cyclic shifts  
prime strings  
lookahead  
dynamic ordering–  
search rearrangement, see dynamic ordering  
cutoff properties



Indeed, our commafree coding problem for ternary 4-tuples doesn't dictate any particular ordering of the 18 classes that would be likely to keep the search tree small. Therefore, instead of calling those choices  $x_1, x_2, \dots, x_{18}$ , it's better to identify them by the various class names, namely  $x_{0001}, x_{0002}, x_{0011}, x_{0012}, x_{0021}, x_{0022}, x_{0102}, x_{0111}, x_{0112}, x_{0121}, x_{0122}, x_{0211}, x_{0212}, x_{0221}, x_{0222}, x_{1112}, x_{1122}, x_{1222}$ . (Algorithm 7.2.1.1F is a good way to generate those names.) At every node of the search tree we then can choose a convenient variable on which to branch, based on previous choices. After beginning with  $x_{0001} \leftarrow 0001$  at level 1 we might decide to try  $x_{0011} \leftarrow 0011$  at level 2; and then, as we've seen, the choices  $x_{0002} \leftarrow 0002, x_{0021} \leftarrow 0021, x_{0111} \leftarrow 0111, x_{0211} \leftarrow 0211$ , and  $x_{1112} \leftarrow 2111$  are forced, so we should make them at levels 3 through 7.

Furthermore, after those forced moves are made, it turns out that they don't force any others. But only two choices for  $x_{0012}$  will remain, while  $x_{0122}$  will have three. Therefore it will probably be wiser to branch on  $x_{0012}$  rather than on  $x_{0122}$  at level 8. (Incidentally, it also turns out that there *is* no commafree code with  $x_{0001} = 0001$  and  $x_{0011} = 0011$ , *except* when  $m = 2$ .)

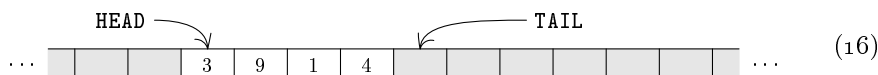
It's easy to adapt Algorithms B and W to allow dynamic ordering. Every node of the search tree can be given a "frame" in which we record the variable being set and the choice that was made. This choice of variable and value can be called a "move" made by the backtrack procedure.

Dynamic ordering can be helpful also after backtracking has taken place. If we continue the example above, where  $x_{0001} = 0001$  and we've explored all cases in which  $x_{0011} = 0011$ , we aren't obliged to continue by trying another value for  $x_{0011}$ . We do want to remember that 0011 should no longer be considered legal, until  $x_{0001}$  changes; but we could decide to explore next a case such as  $x_{0002} = 2000$  at level 2. In fact,  $x_{0002} = 2000$  is quickly seen to be impossible in the presence of 0001 (see exercise 34). An even more efficient choice at level 2, however, is  $x_{0012} = 0012$ , because that branch immediately forces  $x_{0002} = 0002, x_{0022} = 0022, x_{0122} = 0122, x_{0222} = 0222, x_{1222} = 1222$ , and  $x_{0011} = 1001$ .

**Sequential allocation redux.** The choice of a variable and value on which to branch is a delicate tradeoff. We don't want to devote more time to planning than we'll save by having a good plan.

If we're going to benefit from dynamic ordering, we'll need efficient data structures that will lead to good decisions without much deliberation. On the other hand, elaborate data structures need to be updated whenever we branch to a new level, and they need to be downdated whenever we return from that level. Algorithm L illustrates an efficient mechanism based on linked lists; but sequentially allocated lists are often even more appealing, because they are cache-friendly and they involve fewer accesses to memory.

Assume then that we wish to represent a set of items as an unordered sequential list. The list begins in a cell of memory pointed to by HEAD, and TAIL points just beyond the end of the list. For example,



frame  
move  
sequential lists—  
downdating vs updating  
cache-friendly  
unordered sequential list  
cells of memory  
MEM, an array of “cells”–

is one way to represent the set  $\{1, 3, 4, 9\}$ . The number of items currently in the set is  $\text{TAIL} - \text{HEAD}$ ; thus  $\text{TAIL} = \text{HEAD}$  if and only if the list is empty. If we wish to insert a new item  $x$ , knowing that  $x$  isn't already present, we simply set

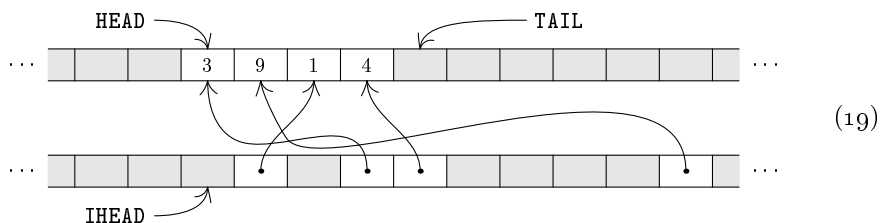
$$\text{MEM}[\text{TAIL}] \leftarrow x, \quad \text{TAIL} \leftarrow \text{TAIL} + 1. \tag{17}$$

Conversely, if  $\text{HEAD} \leq P < \text{TAIL}$ , we can easily delete  $\text{MEM}[P]$ :

$$\text{TAIL} \leftarrow \text{TAIL} - 1; \quad \text{if } P \neq \text{TAIL}, \text{ set } \text{MEM}[P] \leftarrow \text{MEM}[\text{TAIL}]. \tag{18}$$

(We've tacitly assumed in (17) that  $\text{MEM}[\text{TAIL}]$  is available for use whenever a new item is inserted. Otherwise we would have had to test for memory overflow.)

We can't delete an item from a list without knowing its MEM location. Thus we will often want to maintain an "inverse list," assuming that all items  $x$  lie in the range  $0 \leq x < M$ . For example, (16) becomes the following, if  $M = 10$ :



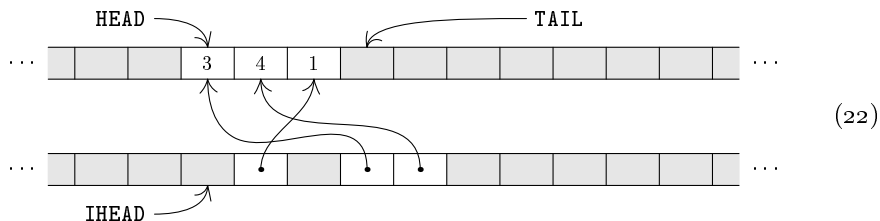
(Shaded cells have undefined contents.) With this setup, insertion (17) becomes

$$\text{MEM}[\text{TAIL}] \leftarrow x, \quad \text{MEM}[\text{IHEAD} + x] \leftarrow \text{TAIL}, \quad \text{TAIL} \leftarrow \text{TAIL} + 1, \tag{20}$$

and  $\text{TAIL}$  will never exceed  $\text{HEAD} + M$ . Similarly, deletion of  $x$  becomes

$$P \leftarrow \text{MEM}[\text{IHEAD} + x], \quad \text{TAIL} \leftarrow \text{TAIL} - 1; \\ \text{if } P \neq \text{TAIL}, \text{ set } y \leftarrow \text{MEM}[\text{TAIL}], \text{MEM}[P] \leftarrow y, \text{MEM}[\text{IHEAD} + y] \leftarrow P. \tag{21}$$

For example, after deleting '9' from (19) we would obtain this:



In more elaborate situations we also want to test whether or not a given item  $x$  is present. If so, we can keep more information in the inverse list. A particularly useful variation arises when the list that begins at  $\text{IHEAD}$  contains a *complete* permutation of the values  $\{\text{HEAD}, \text{HEAD} + 1, \dots, \text{HEAD} + M - 1\}$ , and the memory cells beginning at  $\text{HEAD}$  contain the inverse permutation — although only the first  $\text{TAIL} - \text{HEAD}$  elements of that list are considered to be "active."

For example, in our comma-free code problem with  $m = 3$ , we can begin by putting items representing the  $M = 18$  cycle classes  $[0001], [0002], \dots, [1222]$  into memory cells  $\text{HEAD}$  through  $\text{HEAD} + 17$ . Initially they're all active, with

empty  
insert  
overflow  
inverse list  
inverse permutation  
active

TAIL = HEAD + 18 and MEM[IHEAD +  $c$ ] = HEAD +  $c$  for  $0 \leq c < 18$ . Then whenever we decide to choose a codeword for class  $c$ , we delete  $c$  from the active list by using a souped-up version of (21) that maintains full permutations:

$$\begin{aligned} P &\leftarrow \text{MEM}[\text{IHEAD} + c], \quad \text{TAIL} \leftarrow \text{TAIL} - 1; \\ \text{if } P &\neq \text{TAIL}, \text{ set } y \leftarrow \text{MEM}[\text{TAIL}], \quad \text{MEM}[\text{TAIL}] \leftarrow c, \quad \text{MEM}[P] \leftarrow y, \\ &\quad \text{MEM}[\text{IHEAD} + c] \leftarrow \text{TAIL}, \quad \text{MEM}[\text{IHEAD} + y] \leftarrow P. \end{aligned} \quad (23)$$

delete  
data structures  
periodic  
radix  $m$

Later on, after backtracking to a state where we once again want  $c$  to be considered active, we simply set TAIL  $\leftarrow$  TAIL + 1, because  $c$  will already be in place!

**Lists for the commafree problem.** The task of finding all four-letter comma-free codes is not difficult when  $m = 3$  and only 18 cycle classes are involved. But it already becomes challenging when  $m = 4$ , because we must then deal with  $(4^4 - 4^2)/4 = 60$  classes. Therefore we'll want to give it some careful thought as we try to set it up for backtracking.

The example scenarios for  $m = 3$  considered above suggest that we'll repeatedly want to know the answers to questions such as, "How many words of the form 02\*\* are still available for selection as codewords?" Redundant data structures, oriented to queries of that kind, appear to be needed. Fortunately, we shall see that there's a nice way to provide them, using sequential lists as in (19)–(23).

In Algorithm C below, each of the  $m^4$  four-letter words is given one of three possible states during the search for comma-free codes. A word is *green* if it's part of the current set of tentative codewords. It is *red* if it's not currently a candidate for such status, either because it is incompatible with the existing green words or because the algorithm has already examined all scenarios in which it is green in their presence. Every other word is *blue*, and sort of in limbo; the algorithm might or might not decide to make it red or green. All words are initially blue—except for the  $m^2$  periodic words, which are permanently red.

We'll use the Greek letter  $\alpha$  to stand for the integer value of a four-letter word  $x$  in radix  $m$ . For example, if  $m = 3$  and if  $x$  is the word 0102, then  $\alpha = (0102)_3 = 11$ . The current state of word  $x$  is kept in MEM[ $\alpha$ ], using one of the arbitrary internal codes 2 (GREEN), 0 (RED), or 1 (BLUE).

The most important feature of the algorithm is that every blue word  $x = x_1x_2x_3x_4$  is potentially present in seven different lists, called P1( $x$ ), P2( $x$ ), P3( $x$ ), S1( $x$ ), S2( $x$ ), S3( $x$ ), and CL( $x$ ), where

- P1( $x$ ), P2( $x$ ), P3( $x$ ) are the blue words matching  $x_1***$ ,  $x_1x_2**$ ,  $x_1x_2x_3*$ ;
- S1( $x$ ), S2( $x$ ), S3( $x$ ) are the blue words matching  $***x_4$ ,  $**x_3x_4$ ,  $*x_2x_3x_4$ ;
- CL( $x$ ) hosts the blue words in  $\{x_1x_2x_3x_4, x_2x_3x_4x_1, x_3x_4x_1x_2, x_4x_1x_2x_3\}$ .

These seven lists begin respectively in MEM locations P1OFF +  $p_1(\alpha)$ , P2OFF +  $p_2(\alpha)$ , P3OFF +  $p_3(\alpha)$ , S1OFF +  $s_1(\alpha)$ , S2OFF +  $s_2(\alpha)$ , S3OFF +  $s_3(\alpha)$ , and CLOFF +  $4cl(\alpha)$ ; here (P1OFF, P2OFF, P3OFF, S1OFF, S2OFF, S3OFF, CLOFF) are respectively  $(2m^4, 5m^4, 8m^4, 11m^4, 14m^4, 17m^4, 20m^4)$ . We define  $p_1((x_1x_2x_3x_4)_m) = (x_1)_m$ ,  $p_2((x_1x_2x_3x_4)_m) = (x_1x_2)_m$ ,  $p_3((x_1x_2x_3x_4)_m) = (x_1x_2x_3)_m$ ,  $s_1((x_1x_2x_3x_4)_m) = (x_4)_m$ ,  $s_2((x_1x_2x_3x_4)_m) = (x_3x_4)_m$ ,  $s_3((x_1x_2x_3x_4)_m) = (x_2x_3x_4)_m$ ; and finally  $cl((x_1x_2x_3x_4)_m)$  is an internal number between 0 and  $(m^4 - m^2)/4 - 1$  assigned

**Table 1**

LISTS USED BY ALGORITHM C ( $m = 2$ ), ENTERING LEVEL 1

reflection  
symmetry breaking  
closed

|     | 0    | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | a    | b    | c    | d    | e    | f   |    |
|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|-----|----|
|     | RED  | BLUE | BLUE | BLUE | RED  | RED  | BLUE | BLUE | RED  | BLUE | RED  | BLUE | BLUE | BLUE | BLUE | RED |    |
| 0   |      | 20   | 21   | 22   |      |      | 23   | 24   |      | 29   |      | 2c   | 28   | 2b   | 2a   |     |    |
| 10  |      |      |      |      |      |      |      |      | 1100 | 1001 | 1110 | 1101 | 1011 |      |      |     | P1 |
| 20  | 0001 | 0010 | 0011 | 0110 | 0111 |      |      |      | 2d   |      |      |      |      |      |      |     |    |
| 30  | 25   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |     |    |
| 40  |      | 50   | 51   | 52   |      |      | 54   | 55   |      | 58   |      | 59   | 5c   | 5e   | 5d   |     |    |
| 50  | 0001 | 0010 | 0011 |      | 0110 | 0111 |      |      | 1001 | 1011 |      |      | 1100 | 1110 | 1101 |     | P2 |
| 60  | 53   |      |      |      | 56   |      |      |      | 5a   |      |      |      | 5f   |      |      |     |    |
| 70  |      | 80   | 82   | 83   |      |      | 86   | 87   |      | 88   |      | 8a   | 8c   | 8d   | 8e   |     |    |
| 80  | 0001 |      | 0010 | 0011 |      |      | 0110 | 0111 | 1001 |      | 1011 |      | 1100 | 1101 | 1110 |     | P3 |
| 90  | 81   |      | 84   |      | 84   |      | 88   |      | 89   |      | 8b   |      | 8e   |      | 8f   |     |    |
| a0  |      | b8   | b0   | b9   |      |      | b1   | bb   |      | ba   |      | bd   | b2   | bc   | b3   |     |    |
| b0  | 0010 | 0110 | 1100 | 1110 |      |      |      |      | 0001 | 0011 | 1001 | 0111 | 1101 | 1011 |      |     | S1 |
| c0  | b4   |      |      |      |      |      |      |      | be   |      |      |      |      |      |      |     |    |
| d0  |      | e4   | e8   | ec   |      |      | e9   | ed   |      | e5   |      | ee   | e0   | e6   | ea   |     |    |
| e0  | 1100 |      |      |      | 0001 | 1001 | 1101 |      | 0010 | 0110 | 1110 |      | 0011 | 0111 | 1011 |     | S2 |
| f0  | e1   |      |      |      | e7   |      |      |      | eb   |      |      |      | ef   |      |      |     |    |
| 100 |      | 112  | 114  | 116  |      |      | 11c  | 11e  |      | 113  |      | 117  | 118  | 11a  | 11d  |     |    |
| 110 |      |      | 0001 | 1001 | 0010 |      | 0011 | 1011 | 1100 |      | 1101 |      | 0110 | 1110 | 0111 |     | S3 |
| 120 | 110  |      | 114  |      | 115  |      | 118  |      | 119  |      | 11b  |      | 11e  |      | 11f  |     |    |
| 130 |      | 140  | 141  | 144  |      |      | 145  | 148  |      | 147  |      | 14b  | 146  | 14a  | 149  |     |    |
| 140 | 0001 | 0010 |      |      | 0011 | 0110 | 1100 | 1001 | 0111 | 1110 | 1101 | 1011 |      |      |      |     | CL |
| 150 | 142  |      |      |      | 148  |      |      |      | 14c  |      |      |      |      |      |      |     |    |

This table shows MEM locations 0000 through 150f, using hexadecimal notation. (For example, MEM[40d]=5e; see exercise 36.) Blank entries are unused by the algorithm.

to each class. The seven MEM locations where  $x$  appears in these seven lists are respectively kept in inverse lists that begin in MEM locations P10FF -  $m^4 + \alpha$ , P20FF -  $m^4 + \alpha, \dots, \text{CLOFF} - m^4 + \alpha$ . And the TAIL pointers, which indicate the current list sizes as in (19)–(23), are respectively kept in MEM locations P10FF +  $m^4 + \alpha, \text{P20FF} + m^4 + \alpha, \dots, \text{CLOFF} + m^4 + \alpha$ . (Whew; got that?)

This vast apparatus, which occupies  $22m^4$  cells of MEM, is illustrated in Table 1, at the beginning of the computation for the case  $m = 2$ . Fortunately it's not really as complicated as it may seem at first. Nor is it especially vast: After all,  $22m^4$  is only 13,750 when  $m = 5$ .

(A close inspection of Table 1 reveals incidentally that the words 0100 and 1000 have been colored red, not blue. That's because we can assume without loss of generality that class [0001] is represented either by 0001 or by 0010. The other two cases are covered by left-right reflection of all codewords.)

Algorithm C finds these lists invaluable when it is deciding where next to branch. But it has no further use for a list in which one of the items has become green. Therefore it declares such lists "closed"; and it saves most of the work of list maintenance by updating *only* the lists that remain open. A closed list is represented internally by setting its TAIL pointer to HEAD - 1.

For example, Table 2 shows how the lists in MEM will have changed just after  $x = 0010$  has been chosen to be a tentative codeword. The elements {0001, 0010, 0011, 0110, 0111} of P1( $x$ ) are effectively hidden, because the tail pointer MEM[30] = 1f = 20 - 1 marks that list as closed. (Those list elements ac-

Table 2

LISTS USED BY ALGORITHM C ( $m = 2$ ), ENTERING LEVEL 2undoing-  
Floyd  
compiler

|     | 0    | 1   | 2     | 3    | 4    | 5    | 6    | 7    | 8    | 9    | a    | b    | c    | d    | e    | f   |    |
|-----|------|-----|-------|------|------|------|------|------|------|------|------|------|------|------|------|-----|----|
| 0   | RED  | RED | GREEN | BLUE | RED  | RED  | BLUE | BLUE | RED  | RED  | RED  | BLUE | BLUE | BLUE | BLUE | RED |    |
| 10  |      |     |       |      |      |      |      |      | 1100 | 1011 | 1110 | 1101 | 29   | 28   | 2b   | 2a  |    |
| 20  |      |     |       |      |      |      |      |      | 2c   |      |      |      |      |      |      |     | P1 |
| 30  | 1f   |     |       |      |      |      |      |      |      |      |      |      |      |      |      |     |    |
| 40  |      |     |       |      |      |      | 54   | 55   |      |      |      | 58   | 5c   | 5e   | 5d   |     |    |
| 50  |      |     |       |      | 0110 | 0111 |      |      | 1011 |      |      |      | 1100 | 1110 | 1101 |     | P2 |
| 60  | 4f   |     |       |      | 56   |      |      |      | 59   |      |      |      | 5f   |      |      |     |    |
| 70  |      |     |       |      |      |      | 86   | 87   |      |      |      | 8a   | 8c   | 8d   | 8e   |     |    |
| 80  |      |     |       |      |      |      | 0110 | 0111 |      |      | 1011 |      | 1100 | 1101 | 1110 |     | P3 |
| 90  | 80   |     | 81    |      | 84   |      | 88   |      | 88   |      | 8b   |      | 8e   |      | 8f   |     |    |
| a0  |      |     |       | b9   |      |      |      |      | bb   |      |      | b8   |      |      | ba   |     |    |
| b0  |      |     |       |      |      |      |      |      | 1011 | 0011 | 1101 | 0111 |      |      |      |     | S1 |
| c0  | af   |     |       |      |      |      |      |      | bc   |      |      |      |      |      |      |     |    |
| d0  |      |     |       | ec   |      |      |      |      | ed   |      |      |      | ee   | e0   | e4   |     |    |
| e0  | 1100 |     |       |      | 1101 |      |      |      |      |      |      |      | 0011 | 0111 | 1011 |     | S2 |
| f0  | e1   |     |       |      | e5   |      |      |      | e7   |      |      |      | ef   |      |      |     |    |
| 100 |      |     |       | 116  |      |      | 11c  | 11e  |      |      |      | 117  | 118  | 11a  | 11d  |     |    |
| 110 |      |     |       |      |      |      | 0011 | 1011 | 1100 |      | 1101 |      | 0110 | 1110 | 0111 |     | S3 |
| 120 | 110  |     | 112   |      | 113  |      | 118  |      | 119  |      | 11b  |      | 11e  |      | 11f  |     |    |
| 130 |      |     |       | 144  |      |      | 145  | 148  |      |      |      | 14b  | 146  | 14a  | 149  |     |    |
| 140 |      |     |       |      | 0011 | 0110 | 1100 |      |      | 0111 | 1110 | 1101 | 1011 |      |      |     | CL |
| 150 | 13f  |     |       |      | 147  |      |      |      | 14c  |      |      |      |      |      |      |     |    |

The word 0010 has become green, thus closing its seven lists and making 0001 red. The logic of Algorithm C has also made 1001 red. Hence 0001 and 1001 have been deleted from the open lists in which they formerly appeared (see exercise 37).

ually do still appear in MEM locations 200 through 204, just as they did in Table 1. But there's no need to look at that list while any word of the form 0\*\*\* is green.)

**A general mechanism for doing and undoing.** We're almost ready to finalize the details of Algorithm C and to get on with the search for comma-free codes, but a big problem still remains: The state of computation at every level of the search involves all of the marvelous lists that we've just specified, and those lists aren't tiny. They occupy more than 5000 cells of MEM when  $m = 4$ , and they can change substantially from level to level.

We could make a new copy of the entire state, whenever we advance to a new node of the search tree. But that's a bad idea, because we don't want to perform thousands of memory accesses per node. A much better strategy would be to stick with a single instance of MEM, and to update and downgrade the lists as the search progresses, if we could only think of a simple way to do that.

And we're in luck: There *is* such a way, first formulated by R. W. Floyd in his classic paper "Nondeterministic algorithms" [*JACM* 14 (1967), 636–644]. Floyd's original idea, which required a special compiler to generate forward and backward versions of every program step, can in fact be greatly simplified when all of the changes in state are confined to a single MEM array. All we need to do is to replace every assignment operation of the form 'MEM[ $a$ ] ←  $v$ ' by the

slightly more cumbersome operation

$$\text{store}(a, v) : \text{Set } \text{UNDO}[u] \leftarrow (a, \text{MEM}[a]), \text{MEM}[a] \leftarrow v, \text{ and } u \leftarrow u + 1. \quad (24)$$

Here `UNDO` is a sequential stack that holds (address, value) pairs; in our application we could say ‘`UNDO[u] ← (a ≪ 16) + MEM[a]`’, because the cell addresses and values never exceed 16 bits. Of course we’ll also need to check that the stack pointer  $u$  doesn’t get too large, if the number of assignments has no a priori limit.

Later on, when we want to undo all changes to `MEM` since the time when  $u$  had reached a particular value  $u_0$ , we simply do this:

$$\begin{aligned} \text{unstore}(u_0) : \text{ While } u > u_0, \text{ set } u &\leftarrow u - 1, \\ &(a, v) \leftarrow \text{UNDO}[u], \text{ and } \text{MEM}[a] \leftarrow v. \end{aligned} \quad (25)$$

In our application the unstacking operation ‘ $(a, v) \leftarrow \text{UNDO}[u]$ ’ here could be implemented by saying ‘ $a \leftarrow \text{UNDO}[u] \gg 16, v \leftarrow \text{UNDO}[u] \& \#ffff$ ’.

A useful refinement of this reversible-memory technique is often advantageous, based on the idea of “stamping” that is part of the folklore of programming. It puts only one item on the `UNDO` stack when the same memory address is updated more than once in the same round.

$$\begin{aligned} \text{store}(a, v) : \text{ If } \text{STAMP}[a] \neq \sigma, \text{ set } \text{STAMP}[a] &\leftarrow \sigma, \\ &\text{UNDO}[u] \leftarrow (a, \text{MEM}[a]), \text{ and } u \leftarrow u + 1. \\ \text{Then set } \text{MEM}[a] &\leftarrow v. \end{aligned} \quad (26)$$

Here `STAMP` is an array with one entry for each address in `MEM`. It’s initially all zero, and  $\sigma$  is initially 1. Whenever we come to a fallback point, where the current stack pointer will be remembered as the value  $u_0$  for some future undoing, we “bump” the current stamp by setting  $\sigma \leftarrow \sigma + 1$ . Then (26) will continue to do the right thing. (In programs that run for a long time, we must be careful when integer overflow causes  $\sigma$  to be bumped to zero; see exercise 38.)

Notice that the combination of (24) and (25) will perform five memory accesses for each assignment and its undoing. The combination of (26) and (25) will cost seven mems for the first assignment to `MEM[a]`, but only two mems for every subsequent assignment to the same address. So (26) wins, if multiple assignments exceed one-time-only assignments.

**Backtracking through commafree codes.** OK, we’re now equipped with enough basic knowhow to write a pretty good backtrack program for the problem of generating all commafree four-letter codes.

Algorithm C below incorporates one more key idea, which is a lookahead mechanism that is specific to commafree backtracking; we’ll call it the “poison list.” Every item on the poison list is a pair, consisting of a suffix and a prefix that the commafree rule forbids from occurring together. Every green word  $x_1x_2x_3x_4$ —that is, every word that will be a final codeword in the current branch of our backtrack search—contributes three items to the poison list, namely

$$(*x_1x_2x_3, x_4***), \quad (**x_1x_2, x_3x_4**), \quad \text{and} \quad (**x_1, x_2x_3x_4*). \quad (27)$$

`UNDO`  
stack  
reversible-memory  
stamping  
fallback point  
bump  
overflow  
lookahead  
poison list

If there's a green word on both sides of a poison list entry, we're dead: The commafree condition fails, and we must backtrack. If there's a green word on one side but not the other, we can kill off all blue words on the other side by making them red. And if either side of a poison list entry corresponds to an *empty* list, we can remove this entry from the poison list because it will never affect the outcome. (Blue words become red or green, but red words stay red.)

For example, consider the transition from Table 1 to Table 2. When word 0010 becomes green, the poison list receives its first three items:

$$(*001, 0***), \quad (**00, 10**), \quad (**0, 010*).$$

The first of these kills off the \*001 list, because 0\*\*\* contains the green word 0010. That makes 1001 red. The last of these, similarly, kills off the 010\* list; but that list is empty when  $m = 2$ . The poison list now reduces to a single item, (\*\*00, 10\*\*), which remains poisonous because list \*\*00 contains the blue word 1100 and 10\*\* contains the blue word 1011.

We'll maintain the poison list at the end of MEM, following the CL lists. It obviously will contain at most  $3(m^4 - m^2)/4$  entries, and in fact it usually turns out to be quite small. No inverse list is required; so we shall adopt the simple method of (17) and (18), but with two cells per entry so that TAIL will change by  $\pm 2$  instead of by  $\pm 1$ . The value of TAIL will be stored in MEM at key times so that temporary changes to it can be undone.

The case  $m = 4$ , in which each codeword consists of four *quaternary* digits  $\{0, 1, 2, 3\}$ , is particularly interesting, because an early backtrack program by Lee Laxdal found that no such commafree code can make use of all 60 of the cycle classes [0001], [0002], ..., [2333]. [See B. H. Jiggs, *Canadian Journal of Math.* **15** (1963), 178–187.] Laxdal's program also reportedly showed that at least three of those classes must be omitted; and it found several valid 57-word sets. Further details were never published, because the proof that 58 codewords are impossible depended on what Jiggs called a “quite time-consuming” computation.

Because size 60 is impossible, our algorithm cannot simply assume that a move such as 1001 is forced when the other words 0011, 0110, 1100 of its class have been ruled out. We must also consider the possibility that class [0011] is entirely absent from the code. Such considerations add an interesting further twist to the problem, and Algorithm C describes one way to cope with it.

**Algorithm C** (*Four-letter commafree codes*). Given an alphabet size  $m \leq 7$  and a goal  $g$  in the range  $L - m(m - 1) \leq g \leq L$ , where  $L = (m^4 - m^2)/4$ , this algorithm finds all sets of  $g$  four-letter words that are commafree and include either 0001 or 0010. It uses an array MEM of  $M = \lfloor 23.5m^4 \rfloor$  16-bit numbers, as well as several more auxiliary arrays: ALF of size  $16^3m$ ; STAMP of size  $M$ ; X, C, S, and U of size  $L + 1$ ; FREE and IFREE of size  $L$ ; and a sufficiently large array called UNDO whose maximum size is difficult to guess.

**C1.** [Initialize.] Set  $\text{ALF}[(abcd)_{16}] \leftarrow (abcd)_m$  for  $0 \leq a, b, c, d < m$ . Set  $\text{STAMP}[k] \leftarrow 0$  for  $0 \leq k < M$  and  $\sigma \leftarrow 0$ . Put the initial prefix, suffix, and class lists into MEM, as in Table 1. Also create an empty poison list by

inverse list  
Laxdal  
Jiggs  
stamping++

setting  $\text{MEM}[\text{PP}] \leftarrow \text{POISON}$ , where  $\text{POISON} = 22m^4$  and  $\text{PP} = \text{POISON} - 1$ . Set  $\text{FREE}[k] \leftarrow \text{IFREE}[k] \leftarrow k$  for  $0 \leq k < L$ . Then set  $l \leftarrow 1$ ,  $x \leftarrow \#0001$ ,  $c \leftarrow 0$ ,  $s \leftarrow L - g$ ,  $f \leftarrow L$ ,  $u \leftarrow 0$ , and go to step C3. (Variable  $l$  is the level,  $x$  is a trial word,  $c$  is its class,  $s$  is the “slack,”  $f$  is the number of free classes, and  $u$  is the size of the UNDO stack.)

- C2.** [Enter level  $l$ .] If  $l > L$ , visit the solution  $x_1 \dots x_L$  and go to C6. Otherwise choose a candidate word  $x$  and class  $c$  as described in exercise 39.
- C3.** [Try the candidate.] Set  $U[l] \leftarrow u$  and  $\sigma \leftarrow \sigma + 1$ . If  $x < 0$ , go to C6 if  $s = 0$  or  $l = 1$ , otherwise set  $s \leftarrow s - 1$ . If  $x \geq 0$ , update the data structures to make  $x$  green, as described in exercise 40, escaping to C5 if trouble arises.
- C4.** [Make the move.] Set  $X[l] \leftarrow x$ ,  $C[l] \leftarrow c$ ,  $S[l] \leftarrow s$ ,  $p \leftarrow \text{IFREE}[c]$ ,  $f \leftarrow f - 1$ . If  $p \neq f$ , set  $y \leftarrow \text{FREE}[f]$ ,  $\text{FREE}[p] \leftarrow y$ ,  $\text{IFREE}[y] \leftarrow p$ ,  $\text{FREE}[f] \leftarrow c$ ,  $\text{IFREE}[c] \leftarrow f$ . (This is (23).) Then set  $l \leftarrow l + 1$  and go to C2.
- C5.** [Try again.] While  $u > U[l]$ , set  $u \leftarrow u - 1$  and  $\text{MEM}[\text{UNDO}[u] \gg 16] \leftarrow \text{UNDO}[u] \& \#ffff$ . (Those operations restore the previous state, as in (25).) Then  $\sigma \leftarrow \sigma + 1$  and redden  $x$  (see exercise 40). Go to C2.
- C6.** [Backtrack.] Set  $l \leftarrow l - 1$ , and terminate if  $l = 0$ . Otherwise set  $x \leftarrow X[l]$ ,  $c \leftarrow C[l]$ ,  $f \leftarrow f - 1$ . If  $x < 0$ , repeat this step (class  $c$  was omitted from the code). Otherwise set  $s \leftarrow S[l]$  and go back to C5. ■

Exercises 39 and 40 provide the instructive details that flesh out this skeleton.

Algorithm C needs just 13, 177, and 2380 megamems to prove that no solutions exist for  $m = 4$  when  $g$  is 60, 59, and 58. It needs about 22800 megamems to find the 1152 solutions for  $g = 57$ ; see exercise 44. There are roughly (14, 240, 3700, 38000) thousand nodes in the respective search trees, with most of the activity taking place on levels  $30 \pm 10$ . The height of the UNDO stack never exceeds 2804, and the poison list never contains more than 12 entries at a time.

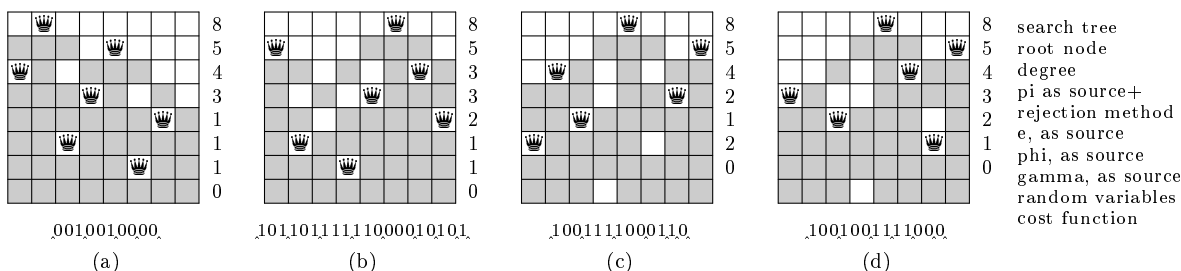
**Running time estimates.** Backtrack programs are full of surprises. Sometimes they produce instant answers to a supposedly difficult problem. But sometimes they spin their wheels endlessly, trying to traverse an astronomically large search tree. And sometimes they deliver results just about as fast as we might expect.

Fortunately, we needn't sit in the dark. There's a simple Monte Carlo algorithm by which we can often tell in advance whether or not a given backtrack strategy will be feasible. This method, based on random sampling, can actually be worked out by hand *before* writing a program, in order to help decide whether to invest further time while following a particular approach. In fact, the very act of carrying out this pleasant pencil-and-paper method often suggests useful cutoff strategies and/or data structures that will be valuable later when a program is being written. For example, the author developed Algorithm C above after first doing some armchair experiments with random choices of potential commafree codewords, and noticing that a family of lists such as those in Tables 1 and 2 would be quite helpful when making further choices.

To illustrate the method, let's consider the  $n$  queens problem again, as represented in Algorithm B\* above. When  $n = 8$ , we can obtain a decent “ballpark

Running time estimates of run time—  
Monte Carlo algorithm  
random sampling  
pencil-and-paper method  
cutoff strategies  
data structures  
author





**Fig. 69.** Four random attempts to solve the 8 queens problem. Such experiments help to estimate the size of the backtrack tree in Fig. 68. The branching degrees are shown at the right of each diagram, while the random bits used for sampling appear below. Cells have been shaded in gray if they are attacked by one or more queens in earlier rows.

estimate” of the size of Fig. 68 by examining only a few random paths in that search tree. We start by writing down the number  $D_1 \leftarrow 8$ , because there are eight ways to place the queen in row 1. (In other words, the root node of the search tree has degree 8.) Then we use a source of random numbers—say the binary digits of  $\pi \bmod 1 = (.001001000011\dots)_2$ —to select one of those placements. Eight choices are possible, so we look at three of those bits; we shall set  $X_1 \leftarrow 2$ , because 001 is the second of the eight possibilities (000, 001,  $\dots$ , 111).

Given  $X_1 = 2$ , the queen in row 2 can’t go into columns 1, 2, or 3. Hence five possibilities remain for  $X_2$ , and we write down  $D_2 \leftarrow 5$ . The next three bits of  $\pi$  lead us to set  $X_2 \leftarrow 5$ , since 5 is the second of the available columns (4, 5, 6, 7, 8) and 001 is the second value of (000, 001,  $\dots$ , 100). If  $\pi$  had continued with 101 or 110 or 111 instead of 001, we would incidentally have used the “rejection method” of Section 3.4.1 and moved to the next three bits; see exercise 47.

Continuing in this way leads to  $D_3 \leftarrow 4$ ,  $X_3 \leftarrow 1$ ; then  $D_4 \leftarrow 3$ ,  $X_4 \leftarrow 4$ . (Here we used the two bits 00 to select  $X_3$ , and the next two bits 00 to select  $X_4$ .) The remaining branches are forced:  $D_5 \leftarrow 1$ ,  $X_5 \leftarrow 7$ ;  $D_6 \leftarrow 1$ ,  $X_6 \leftarrow 3$ ;  $D_7 \leftarrow 1$ ,  $X_7 \leftarrow 6$ ; and we’re stuck when we reach level 8 and find  $D_8 \leftarrow 0$ .

These sequential random choices are depicted in Fig. 69(a), where we’ve used them to place each queen successively into an unshaded cell. Parts (b), (c), and (d) of Fig. 69 correspond in the same way to choices based on the binary digits of  $e \bmod 1$ ,  $\phi \bmod 1$ , and  $\gamma \bmod 1$ . Exactly 10 bits of  $\pi$ , 20 bits of  $e$ , 13 bits of  $\phi$ , and 13 bits of  $\gamma$  were used to generate these examples.

In this discussion the notation  $D_k$  stands for a branching degree, not for a domain of values. We’ve used uppercase letters for the numbers  $D_1$ ,  $X_1$ ,  $D_2$ , etc., because those quantities are random variables. Once we’ve reached  $D_l = 0$  at some level, we’re ready to estimate the overall cost, by implicitly assuming that the path we’ve taken is representative of *all* root-to-leaf paths in the tree.

The cost of a backtrack program can be assessed by summing the individual amounts of time spent at each node of the search tree. Notice that every node on level  $l$  of that tree can be labeled uniquely by a sequence  $x_1 \dots x_{l-1}$ , which defines the path from the root to that node. Thus our goal is to estimate the sum of all  $c(x_1 \dots x_{l-1})$ , where  $c(x_1 \dots x_{l-1})$  is the cost associated with node  $x_1 \dots x_{l-1}$ .

For example, the four queens problem is represented by the search tree (4), and its cost is the sum of 17 individual costs

$$c() + c(1) + c(13) + c(14) + c(142) + c(2) + c(24) + \cdots + c(413) + c(42). \quad (28)$$

If  $C(x_1 \dots x_l)$  denotes the total cost of the subtree rooted at  $x_1 \dots x_l$ , then

$$C(x_1 \dots x_l) = c(x_1 \dots x_l) + C(x_1 \dots x_l x_{l+1}^{(1)}) + \cdots + C(x_1 \dots x_l x_{l+1}^{(d)}) \quad (29)$$

when the choices for  $x_{l+1}$  at node  $x_1 \dots x_l$  are  $\{x_{l+1}^{(1)}, \dots, x_{l+1}^{(d)}\}$ . For instance in (4) we have  $C(1) = c(1) + C(13) + C(14)$ ;  $C(13) = c(13)$ ; and  $C() = c() + C(1) + C(2) + C(3) + C(4)$  is the overall cost (28).

In these terms a Monte Carlo estimate for  $C()$  is extremely easy to compute:

**Theorem E.** *Given  $D_1, X_1, D_2, X_2, \dots$  as above, the cost of backtracking is*

$$C() = E(c() + D_1(c(X_1) + D_2(c(X_1 X_2) + D_3(c(X_1 X_2 X_3) + \cdots))). \quad (30)$$

*Proof.* Node  $x_1 \dots x_l$ , with branch degrees  $d_1, \dots, d_l$  above it, is reached with probability  $1/d_1 \dots d_l$ ; so it contributes  $d_1 \dots d_l c(x_1 \dots x_l) / d_1 \dots d_l = c(x_1 \dots x_l)$  to the expected value in this formula. ■

For example, the tree (4) has six root-to-leaf paths, and they occur with respective probabilities  $1/8, 1/8, 1/4, 1/4, 1/8, 1/8$ . The first one contributes  $1/8$  times  $c() + 4(c(1) + 2(c(13)))$ , namely  $c()/8 + c(1)/2 + c(13)$ , to the expected value. The second contributes  $c()/8 + c(1)/2 + c(14) + c(142)$ ; and so on.

A special case of Theorem E, with all  $c(x_1 \dots x_l) = 1$ , tells us how to estimate the total size of the tree, which is often a crucial quantity:

**Corollary E.** *The number of nodes in the search tree, given  $D_1, D_2, \dots$ , is*

$$E(1 + D_1 + D_1 D_2 + \cdots) = E(1 + D_1(1 + D_2(1 + D_3(1 + \cdots))). \quad (31)$$

For example, Fig. 69 gives us four estimates for the size of the tree in Fig. 68, using the numbers  $D_j$  at the right of each  $8 \times 8$  diagram. The estimate from Fig. 69(a) is  $1 + 8(1 + 5(1 + 4(1 + 3(1 + 1(1 + 1(1 + 1)))))) = 2129$ ; and the other three are respectively 2689, 1489, 2609. None of them is extremely far from the true number, 2057, although we can't expect to be so lucky all the time.

The detailed study in exercise 51 shows that the estimate (31) in the case of 8 queens turns out to be quite well behaved:

$$(\min 489, \quad \text{ave } 2057, \quad \max 7409, \quad \text{dev } \sqrt{1146640} \approx 1071). \quad (32)$$

The analogous problem for 16 queens has a much less homogeneous search tree:

$$(\min 2597105, \quad \text{ave } 1141190303, \quad \max 131048318769, \quad \text{dev } \approx 1234000000). \quad (33)$$

Still, this standard deviation is roughly the same as the mean, so we'll usually guess the correct order of magnitude. (For example, ten independent experiments predicted .632, .866, .237, 1.027, 4.006, .982, .143, .140, 3.402, and .510 billion nodes, respectively. The mean of these is 1.195.) A thousand trials with  $n = 64$  suggest that the problem of 64 queens will have about  $3 \times 10^{65}$  nodes in its tree.

Let's formulate this estimation procedure precisely, so that it can be performed conveniently by machine as well as by hand:

binomial tree  
estimating solutions

**Algorithm E** (*Estimated cost of backtrack*). Given domains  $D_k$  and properties  $P_l$  as in Algorithm B, together with node costs  $c(x_1 \dots x_l)$  as above, this algorithm computes the quantity  $S$  whose expected value is the total cost  $C()$  in (30). It uses an auxiliary array  $y_1 y_2 \dots$  whose size should be  $\geq \max(|D_1|, \dots, |D_n|)$ .

- E1.** [Initialize.] Set  $l \leftarrow D \leftarrow 1$ ,  $S \leftarrow 0$ , and initialize any data structures needed.
- E2.** [Enter level  $l$ .] (At this point  $P_{l-1}(X_1, \dots, X_{l-1})$  holds.) Set  $S \leftarrow S + D \cdot c(X_1 \dots X_{l-1})$ . If  $l > n$ , terminate the algorithm. Otherwise set  $d \leftarrow 0$  and set  $x \leftarrow \min D_l$ , the smallest element of  $D_l$ .
- E3.** [Test  $x$ .] If  $P_l(X_1, \dots, X_{l-1}, x)$  holds, set  $y_d \leftarrow x$  and  $d \leftarrow d + 1$ .
- E4.** [Try again.] If  $x \neq \max D_l$ , set  $x$  to the next larger element of  $D_l$  and return to step E3.
- E5.** [Choose and try.] If  $d = 0$ , terminate. Otherwise set  $D \leftarrow D \cdot d$  and  $X_l \leftarrow y_I$ , where  $I$  is a uniformly random integer in  $\{0, \dots, d - 1\}$ . Update the data structures to facilitate testing  $P_{l+1}$ , set  $l \leftarrow l + 1$ , and go back to E2. ■

Although Algorithm E looks rather like Algorithm B, it never backtracks.

Of course we can't expect this algorithm to give decent estimates in cases where the backtrack tree is wildly erratic. The *expected* value of  $S$ , namely  $ES$ , is indeed the true cost; but the *probable* values of  $S$  might be quite different.

An extreme example of bad behavior occurs if property  $P_l$  is the simple condition ' $x_1 > \dots > x_l$ ' and all domains are  $\{1, \dots, n\}$ . Then there's only one solution,  $x_1 \dots x_n = n \dots 1$ ; and backtracking is a particularly stupid way to find it!

The search tree for this somewhat ridiculous problem is, nevertheless, quite interesting. It is none other than the binomial tree  $T_n$  of Eq. 7.2.1.3–(21), which has  $\binom{n}{l}$  nodes on level  $l + 1$  and  $2^n$  nodes in total. If we set all costs to 1, the expected value of  $S$  is therefore  $2^n = e^{n \ln 2}$ . But exercise 50 proves that  $S$  will almost always be much smaller, less than  $e^{(\ln n)^2 \ln \ln n}$ . Furthermore the average value of  $l$  when Algorithm E terminates with respect to  $T_n$  is only  $H_n + 1$ . When  $n = 100$ , for example, the probability that  $l \geq 20$  on termination is only 0.0000000027, while the vast majority of the nodes are near level 51.

Many refinements of Algorithm E are possible. For example, exercise 52 shows that the choices in step E5 need not be uniform. We shall discuss improved estimation techniques in Section 7.2.2.9, after having seen numerous examples of backtracking in practice.

**\*Estimating the number of solutions.** Sometimes we know that a problem has more solutions than we could ever hope to generate, yet we still want to know roughly how many there are. Algorithm E will tell us the approximate number, in cases where the backtrack process never reaches a dead end—that is, if it never terminates with  $d = 0$  in step E5. There may be another criterion for successful termination in step E2 even though  $l$  might still be  $\leq n$ . The expected final value of  $D$  is exactly the total number of solutions, because every solution  $X_1 \dots X_l$  constructed by the algorithm is obtained with probability  $1/D$ .

For example, suppose we want to know the number of different paths by which a king can go from one corner of a chessboard to the opposite corner, without revisiting any square. One such path, chosen at random using the bits of  $\pi$  for guidance as we did in Fig. 69(a), is shown here. Starting in the upper left corner, we have 3 choices for the first move. Then, after moving to the right, there are 4 choices for the second move. And so on. We never make a move that would disconnect us from the goal; in particular, two of the moves are actually forced. (Exercise 58 explains one way to avoid fatal mistakes.)

The probability of obtaining this particular path is exactly  $\frac{1}{3} \frac{1}{4} \frac{1}{6} \frac{1}{6} \frac{1}{2} \frac{1}{6} \frac{1}{7} \dots \frac{1}{2} = 1/D$ , where  $D = 3 \times 4 \times 6 \times 6 \times 2 \times 6 \times 7 \times \dots \times 2 = 1^2 \cdot 2^4 \cdot 3^4 \cdot 4^{10} \cdot 5^9 \cdot 6^6 \cdot 7^1 \approx 8.7 \times 10^{20}$ . Thus we can reasonably guess, at least tentatively, that there are  $10^{21}$  such paths, more or less.

Of course that guess, based on a single random sample, rests on very shaky grounds. But we know that the average value  $M_N = (D^{(1)} + \dots + D^{(N)})/N$  of  $N$  guesses, in  $N$  independent experiments, will almost surely approach the correct number.

How large should  $N$  be, before we can have any confidence in the results? The actual values of  $D$  obtained from random king paths tend to vary all over the map. Figure 70 plots typical results, as  $N$  varies from 1 to 10000. For each value of  $N$  we can follow the advice of statistics textbooks and calculate the sample variance  $V_N = S_N/(N - 1)$  as in Eq. 4.2.2–(16); then  $M_N \pm \sqrt{V_N/N}$  is the textbook estimate. The top diagram in Fig. 70 shows these “error bars” in gray, surrounding black dots for  $M_N$ . This sequence  $M_N$  does appear to settle down after  $N$  reaches 3000 or so, and to approach a value near  $5 \times 10^{25}$ . That’s much higher than our first guess, but it has lots of evidence to back it up.

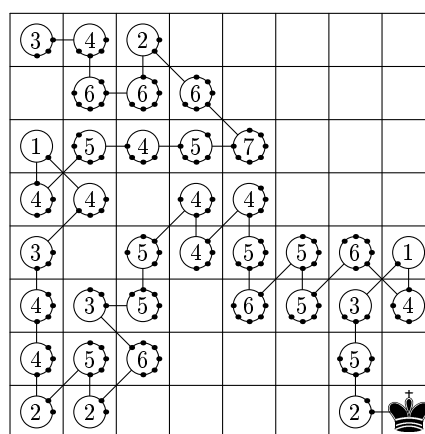
On the other hand, the bottom chart in Fig. 70 shows the distribution of the *logarithms* of the 10000 values of  $D$  that were used to make the top chart. Almost half of those values were totally negligible—less than  $10^{20}$ . About 75% of them were less than  $10^{24}$ . But some of them\* exceeded  $10^{28}$ . Can we really rely on a result that’s based on such chaotic behavior? Is it really right to throw away most of our data and to trust almost entirely on observations that were obtained from comparatively few rare events?

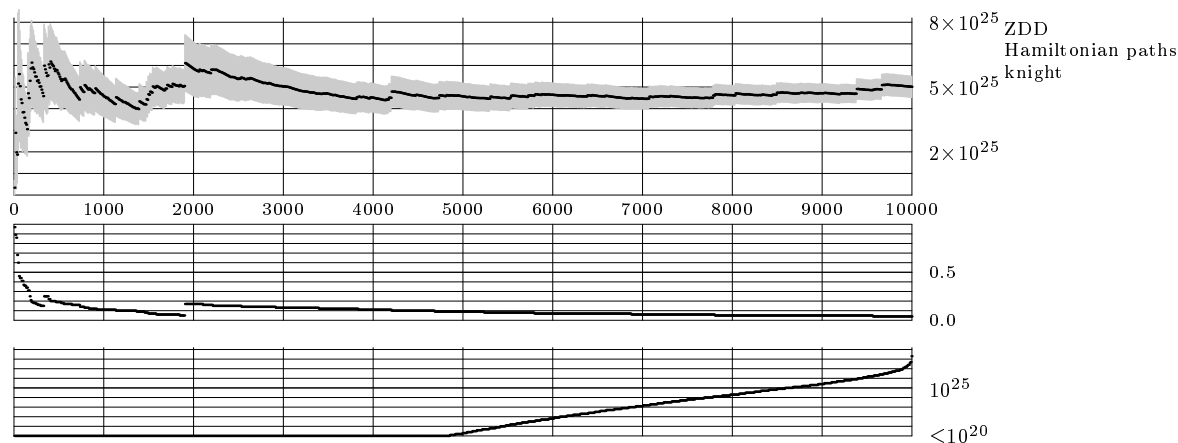
Yes, we’re okay! Some of the justification appears in exercise MPR–124, which is based on theoretical work by P. Diaconis and S. Chatterjee. In the paper cited with that exercise, they defend a simple measure of quality,

$$Q_N = \max(D^{(1)}, \dots, D^{(N)}) / (NM_N) = \frac{\max(D^{(1)}, \dots, D^{(N)})}{D^{(1)} + \dots + D^{(N)}}, \quad (34)$$

\* Four of the actual values that led to Fig. 70 were larger than  $10^{28}$ ; the largest,  $\approx 2.1 \times 10^{28}$ , came from a path of length 57. The smallest estimate, 19361664, came from a path of length 10.

king  
chessboard  
simple paths  
paths, simple  
 $\pi$  for guidance  
statistics  
sample variance  
variance  
error bars  
discarded data  
Diaconis  
Chatterjee





**Fig. 70.** Estimates of the number of king paths, based on up to 10000 random trials. The middle graph shows the corresponding quality measures of Eq. (34). The lower graph shows the *logarithms* of the individual estimates  $D^{(k)}$ , after they've been sorted.

arguing that a reasonable policy in most experiments such as these is to stop sampling when  $Q_N$  gets small. (Values of this statistic  $Q_N$  have been plotted in the middle of Fig. 70.)

Furthermore we can estimate other properties of the solutions to a backtrack problem, instead of merely counting those solutions. For example, the expected value of  $lD$  on termination of the random king's path algorithm is the total *length* of such paths. The data underlying Fig. 70 suggests that this total is  $(2.66 \pm .14) \times 10^{27}$ ; hence the average path length appears to be about 53. The samples also indicate that about 34% of the paths pass through the center; about 46% touch the upper right corner; about 22% touch both corners; and about 7% pass through the center and both corners.

For this particular problem we don't actually need to rely on estimates, because the ZDD technology of Section 7.1.4 allows us to compute the *true* values. (See exercise 59.) The total number of simple corner-to-corner king paths on a chessboard is exactly 50,819,542,770,311,581,606,906,543; this value lies almost within the error bars of Fig. 70 for all  $N \geq 250$ , except for a brief interval near  $N = 1400$ . And the total length of all these paths turns out to be exactly 2,700,911,171,651,251,701,712,099,831, which is a little higher than our estimate. The true average length is therefore  $\approx 53.15$ . The true probabilities of hitting the center, a given corner, both corners, and all three of those spots are respectively about 38.96%, 50.32%, 25.32%, and 9.86%.

The total number of corner-to-corner king paths of the maximum length, 63, is 2,811,002,302,704,446,996,926. This is a number that can *not* be estimated well by a method such as Algorithm E without additional heuristics.

The analogous problem for corner-to-corner *knight* paths, of any length, lies a bit beyond ZDD technology because many more ZDD nodes are needed. Using Algorithm E we can estimate that there are about  $(8.6 \pm 1.2) \times 10^{19}$  such paths.

**Factoring the problem.** Imagine an instance of backtracking that is equivalent to solving two *independent* subproblems. For example, we might be looking for all sequences  $x = x_1 x_2 \dots x_n$  that satisfy  $P_n(x_1, x_2, \dots, x_n) = F(x_1, x_2, \dots, x_n)$ , where

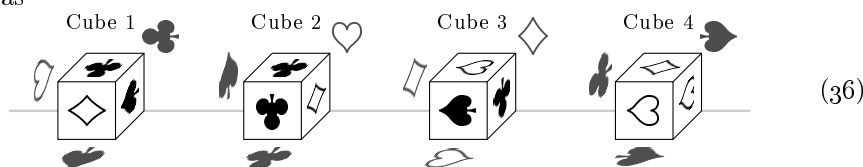
$$F(x_1, x_2, \dots, x_n) = G(x_1, \dots, x_k) \wedge H(x_{k+1}, \dots, x_n). \quad (35)$$

Then the size of the backtrack tree is essentially the *product* of the tree sizes for  $G$  and for  $H$ , even if we use dynamic ordering. Hence it's obviously foolish to apply the general setup of (1) and (2). We can do much better by finding all solutions to  $G$  first, then finding all solutions to  $H$ , thereby reducing the amount of computation to the *sum* of the tree sizes. Again we've divided and conquered, by factoring the compound problem (35) into separate subproblems.

We discussed a less obvious application of problem factorization near the beginning of Chapter 7, in connection with latin squares: Recall that E. T. Parker sped up the solution of 7-(6) by more than a dozen orders of magnitude, when he discovered 7-(7) by essentially factoring 7-(6) into ten subproblems whose solutions could readily be combined.

In general, each solution  $x$  to some problem  $F$  often implies the existence of solutions  $x^{(p)} = \phi_p(x)$  to various simpler problems  $F_p$  that are "homomorphic images" of  $F$ . And if we're lucky, the solutions to those simpler problems can be combined and "lifted" to a solution of the overall problem. Thus it pays to be on the lookout for such simplifications.

Let's look at another example. F. A. Schossow invented a tantalizing puzzle [U.S. Patent 646463 (3 April 1900)] that "went viral" in 1967 when a marketing genius decided to rename it *Instant Insanity*®. The problem is to take four cubes such as



where each face has been marked in one of four ways, and to arrange them in a row so that all four markings appear on the top, bottom, front, and back sides. The placement in (36) is incorrect, because there are two ♣s (and no ♠) on top. But we get a solution if we rotate each cube by 90°.

There are 24 ways to place each cube, because any of the six faces can be on top and we can rotate four ways while keeping the top unchanged. So the total number of placements is  $24^4 = 331776$ . But this problem can be factored in an ingenious way, so that all solutions can be found quickly by hand! [See F. de Carteblanche, *Eureka* 9 (1947), 9–11.] The idea is that any solution to the puzzle gives us two each of {♣, ♠, ♠, ♣}, if we look only at the top and bottom or only at the front and back. That's a much easier problem to solve.

For this purpose a cube can be characterized by its three pairs of markings on opposite faces; in (36) these face-pairs are respectively

$$\{\clubsuit\heartsuit, \clubsuit\spadesuit, \heartsuit\spadesuit\}, \quad \{\clubsuit\clubsuit, \clubsuit\heartsuit, \spadesuit\heartsuit\}, \quad \{\heartsuit\heartsuit, \clubsuit\heartsuit, \spadesuit\heartsuit\}, \quad \{\clubsuit\heartsuit, \spadesuit\heartsuit, \spadesuit\heartsuit\}. \quad (37)$$

factorization of problems-  
*independent* subproblems  
 backtrack tree  
 tree sizes  
 dynamic ordering  
 divide and conquer paradigm  
 latin squares  
 Parker  
 homomorphic images  
 lifted  
 Schossow  
 Patent  
 Tantalizer, see Instant InsanityTM  
 Armbruster  
 Instant InsanityTM  
 Carteblanche

Which of the  $3^4 = 81$  ways to choose one face-pair from each cube will give us  $\{\clubsuit, \clubsuit, \diamond, \diamond, \heartsuit, \heartsuit, \spadesuit, \spadesuit\}$ ? They can all be discovered in a minute or two, by listing the nine possibilities for cubes (1, 2) and the nine for (3, 4). We get just three,

$$(\clubsuit\diamond, \clubsuit\heartsuit, \spadesuit\diamond, \spadesuit\heartsuit), \quad (\spadesuit\heartsuit, \clubsuit\heartsuit, \clubsuit\diamond, \spadesuit\diamond), \quad (\spadesuit\heartsuit, \spadesuit\diamond, \clubsuit\diamond, \clubsuit\heartsuit). \quad (38)$$

Notice furthermore that each solution can be “halved” so that one each of  $\{\clubsuit, \diamond, \heartsuit, \spadesuit\}$  appears on both sides, by swapping face-pairs; we can change (38) to


$$(\diamond\clubsuit, \clubsuit\heartsuit, \spadesuit\diamond, \heartsuit\spadesuit), \quad (\heartsuit\spadesuit, \clubsuit\heartsuit, \diamond\clubsuit, \spadesuit\diamond), \quad (\heartsuit\spadesuit, \spadesuit\diamond, \diamond\clubsuit, \clubsuit\heartsuit). \quad (39)$$

Each of these solutions to the opposite-face subproblem can be regarded as a 2-regular graph, because every vertex of the multigraph whose edges are (say)  $\diamond - \clubsuit, \clubsuit - \heartsuit, \spadesuit - \diamond, \heartsuit - \spadesuit$  has exactly two neighbors.

A solution to Instant Insanity<sup>©</sup> will give us *two* such 2-regular factors, one for top-and-bottom and one for front-and-back. Furthermore those two factors will have disjoint edges: We can’t use the same face-pair in both. Therefore problem (36) can be solved only by using the first and third factor in (39).

Conversely, whenever we have two disjoint 2-regular graphs, we can always use them to position the cubes as desired, thus “lifting” the factors to a solution of the full problem.

Exercise 75 illustrates another kind of problem factorization.

 I may decide to insert a (small?) amount of additional material here, as I prepare sections 7.2.2.1–7.2.2.9.

**Historical notes.** The origins of backtrack programming are obscure. Equivalent ideas must have occurred to many people, yet there was hardly any reason to write them down until computers existed. We can be reasonably sure that James Bernoulli used such principles in the 17th century, when he successfully solved the “Tot tibi sunt dotes” problem that had eluded so many others (see Section 7.2.1.7), because traces of the method exist in his exhaustive list of solutions.

Backtrack programs typically traverse the tree of possibilities by using what is now called depth-first search, a general graph exploration procedure that Édouard Lucas credited to a student named Trémaux [*Récréations Mathématiques* **1** (Paris: Gauthier-Villars, 1882), 47–50].

The eight queens problem was first proposed by Max Bezzel [*Schachzeitung* **3** (1848), 363; **4** (1849), 40] and by Franz Nauck [*Illustrierte Zeitung* **14**, 361 (1 June 1850), 352; **15**, 377 (21 September 1850), 182], perhaps independently. C. F. Gauss saw the latter publication, and wrote several letters about it to his friend H. C. Schumacher. Gauss’s letter of 27 September 1850 is especially interesting, because it explained how to find all the solutions by backtracking—which he called ‘Tatonniren’, from a French term meaning “to feel one’s way.” He also listed the lexicographically first solutions of each equivalence class under reflection and rotation: 15863724, 16837425, 24683175, 25713864, 25741863, 26174835, 26831475, 27368514, 27581463, 35281746, 35841726, and 36258174.

2-regular graph  
disjoint  
lifting  
Bernoulli  
Tot tibi  
depth-first search  
Lucas  
Trémaux  
eight queens problem  
Bezzel  
Nauck  
Gauss  
Schumacher  
lexicographically

Computers arrived a hundred years later, and people began to use them for combinatorial problems. The time was therefore ripe for backtracking to be described as a general technique, and Robert J. Walker rose to the occasion [*Proc. Symposia in Applied Math.* **10** (1960), 91–94]. His brief note introduced Algorithm W in machine-oriented form, and mentioned that the procedure could readily be extended to find variable-length patterns  $x_1 \dots x_n$  where  $n$  is not fixed.

The next milestone was a paper by Solomon W. Golomb and Leonard D. Baumert [*JACM* **12** (1965), 516–524], who formulated the general problem carefully and presented a variety of examples. In particular, they discussed the search for maximum comma-free codes, and noted that backtracking can be used to find successively better and better solutions to combinatorial optimization problems. They introduced certain kinds of lookahead, as well as the important idea of dynamic ordering by branching on variables with the fewest remaining choices.

Other noteworthy early discussions of backtrack programming appear in Mark Wells’s book *Elements of Combinatorial Computing* (1971), Chapter 4; in a survey by J. R. Bitner and E. M. Reingold, *CACM* **18** (1975), 651–656; and in the Ph.D. thesis of John Gaschnig [Report CMU-CS-79-124 (Carnegie Mellon University, 1979), Chapter 4]. Gaschnig introduced techniques of “backmarking” and “backjumping” that we shall discuss later.

Monte Carlo estimates of the cost of backtracking were first described briefly by M. Hall, Jr., and D. E. Knuth in *Computers and Computing*, *AMM* **72**, 2, part 2, Slaughter Memorial Papers No. 10 (February 1965), 21–28. Knuth gave a much more detailed exposition a decade later, in *Math. Comp.* **29** (1975), 121–136. Such methods can be considered as special cases of so-called “importance sampling”; see J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods* (London: Methuen, 1964), 57–59. Studies of random self-avoiding walks such as the king paths discussed above were inaugurated by M. N. Rosenbluth and A. W. Rosenbluth, *J. Chemical Physics* **23** (1955), 356–359.

Backtrack applications are nicely adaptable to parallel programming, because different parts of the search tree are often completely independent of each other; thus disjoint subtrees can be explored on different machines, with a minimum of interprocess communication. Already in 1964, D. H. Lehmer explained how to subdivide a problem so that two computers of different speeds could work on it simultaneously and finish at the same time. The problem that he considered had a search tree of known shape (see Theorem 7.2.1.3L); but we can do essentially similar load balancing even in much more complicated situations, by using Monte Carlo estimates of the subtree sizes. Although many ideas for parallelizing combinatorial searches have been developed over the years, such techniques are beyond the scope of this book. Readers can find a nice introduction to a fairly general approach in the paper by R. Finkel and U. Manber, *ACM Transactions on Programming Languages and Systems* **9** (1987), 235–256.

M. Alekhovich, A. Borodin, J. Buresh-Oppenheimer, R. Impagliazzo, A. Magen, and T. Pitassi have defined *priority branching trees*, a general model of computation with which they were able to prove rigorous bounds on what backtrack programs can do, in *Computational Complexity* **20** (2011), 679–740.

Walker  
Golomb  
Baumert  
comma-free codes  
optimization  
lookahead  
dynamic ordering  
minimum remaining values  
Wells  
Bitner  
Reingold  
Gaschnig  
backmarking  
backjumping  
Monte Carlo estimates  
Hall  
Knuth  
importance sampling  
Hammersley  
Handscomb  
self-avoiding walks  
king paths  
Rosenbluth  
Rosenbluth  
parallel programming  
search tree  
Lehmer  
load balancing  
Finkel  
Manber  
Alekhovich  
Borodin  
Buresh-Oppenheimer  
Impagliazzo  
Magen  
Pitassi  
priority branching trees



## EXERCISES

- ▶ 1. [22] Explain how the tasks of generating (i)  $n$ -tuples, (ii) permutations of distinct items, (iii) combinations, (iv) integer partitions, (v) set partitions, and (vi) nested parentheses can all be regarded as special cases of backtrack programming, by presenting suitable domains  $D_k$  and cutoff properties  $P_i(x_1, \dots, x_i)$  that satisfy (1) and (2).
  - 2. [10] True or false: We can choose  $D_1$  so that  $P_1(x_1)$  is always true.
  - 3. [16] Using a chessboard and eight coins to represent queens, one can follow the steps of Algorithm B and essentially traverse the tree of Fig. 68 by hand in about three hours. Invent a trick to save half of the work.
- ▶ 4. [20] Reformulate Algorithm B as a *recursive* procedure called *try(l)*, having global variables  $n$  and  $x_1 \dots x_n$ , to be invoked by saying ‘*try(1)*’. Can you imagine why the author of this book decided *not* to present the algorithm in such a recursive form?
  - 5. [20] Given  $r$ , with  $1 \leq r \leq n$ , in how many ways can 7 nonattacking queens be placed on an  $8 \times 8$  chessboard, if no queen is placed in row  $r$ ?
  - 6. [20] (T. B. Sprague, 1890.) Are there any values  $n > 5$  for which the  $n$  queens problem has a “framed” solution with  $x_1 = 2$ ,  $x_2 = n$ ,  $x_{n-1} = 1$ , and  $x_n = n - 1$ ?
  - 7. [20] Are there two 8-queen placements with the same  $x_1 x_2 x_3 x_4 x_5 x_6$ ?
  - 8. [21] Can a  $4m$ -queen placement have  $3m$  queens on “white” squares?
- ▶ 9. [22] Adapt Algorithm W to the  $n$  queens problem, using bitwise operations on  $n$ -bit numbers as suggested in the text.

10. [M25] (W. Ahrens, 1910.) Both solutions of the  $n$  queens problem when  $n = 4$  have *chiral symmetry*: Rotation by  $90^\circ$  leaves them unchanged, but reflection doesn’t.

- a) Can the  $n$  queens problem have a solution with reflection symmetry?
- b) Show that chiral symmetry is impossible when  $n \bmod 4 \in \{2, 3\}$ .
- c) Sometimes the solution to an  $n$  queens problem contains four queens that form the corners of a tilted square, as shown here. Prove that we can always get another solution by tilting the square the other way (but leaving the other  $n - 4$  queens in place).

d) Let  $C_n$  be the number of chirally symmetric solutions, and suppose  $c_n$  of them have  $x_k > k$  for  $1 \leq k \leq n/2$ . Prove that  $C_n = 2^{\lfloor n/4 \rfloor} c_n$ .

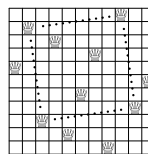
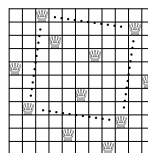
11. [M28] (*Wraparound queens.*) Replace (3) by the stronger conditions ‘ $x_j \neq x_k$ ,  $(x_k - x_j) \bmod n \neq k - j$ ,  $(x_j - x_k) \bmod n \neq k - j$ ’. (The  $n \times n$  grid becomes a torus.) Prove that the resulting problem is solvable if and only if  $n$  is not divisible by 2 or 3.

12. [M30] For which  $n \geq 0$  does the  $n$  queens problem have at least one solution?

13. [M25] If exercise 11 has  $T(n)$  toroidal solutions, show that  $Q(mn) \geq Q(m)^n T(n)$ .

14. [HM47] Does  $(\ln Q(n))/(n \ln n)$  approach a positive constant as  $n \rightarrow \infty$ ?

15. [21] Let  $H(n)$  be the number of ways that  $n$  queen bees can occupy an  $n \times n$  honeycomb so that no two are in the same line. (For example, one of the  $H(4) = 7$  ways is shown here.) Compute  $H(n)$  for small  $n$ .



16. [15] J. H. Quick (a student) noticed that the loop in step L2 of Algorithm L can be changed from ‘while  $x_i < 0$ ’ to ‘while  $x_i \neq 0$ ’, because  $x_i$  cannot be positive at that point of the algorithm. So he decided to eliminate the minus signs and just set  $x_{i+k+1} \leftarrow k$  in step L3. Was it a good idea?

$n$ -tuples  
 tuples  
 permutations  
 combinations  
 integer partitions  
 partitions  
 set partitions  
 nested parentheses  
 parentheses  
 domains  
 cutoff  
 properties  
 recursive  
 global variables  
 author  
 recursion versus iteration  
 Sprague  
 bitwise operations  
 Ahrens  
 chiral symmetry  
 Rotation by  $90^\circ$   
 Wraparound queens  
 broken diagonal, see wraparound  
 torus  
 $n$  queens problem  
 queen bees  
 bees  
 honeycomb  
 hexagons  
 Quick  
 Langford pairs+

17. [17] Suppose that  $n = 4$  and Algorithm L has reached step L2 with  $l = 4$  and  $x_1x_2x_3 = 241$ . What are the current values of  $x_4x_5x_6x_7x_8$ ,  $p_0p_1p_2p_3p_4$ , and  $y_1y_2y_3$ ?
19. [M19] What are the domains  $D_l$  in Langford's problem (7)?
- 20. [21] Extend Algorithm L so that it forces  $x_l \leftarrow k$  whenever  $k \notin \{x_1, \dots, x_{l-1}\}$ .
- 21. [M25] If  $x = x_1x_2 \dots x_{2n}$ , let  $x^D = (-x_{2n}) \dots (-x_2)(-x_1) = -x^R$  be its dual.
- Show that if  $n$  is odd and  $x$  solves Langford's problem (7), we have  $x_k = n$  for some  $k \leq \lfloor n/2 \rfloor$  if and only if  $x_k^D = n$  for some  $k > \lfloor n/2 \rfloor$ .
  - Find a similar rule that distinguishes  $x$  from  $x^D$  when  $n$  is even.
  - Consequently the algorithm of exercise 20 can be modified so that exactly one of each dual pair of solutions  $\{x, x^D\}$  is visited.
22. [M26] Explore "loose Langford pairs": Replace ' $j + k + 1$ ' in (7) by ' $j + \lfloor 3k/2 \rfloor$ '.
23. [17] We can often obtain one word rectangle from another by changing only a letter or two. Can you think of any  $5 \times 6$  rectangles that almost match (10)?
24. [20] Customize Algorithm B so that it will find all  $5 \times 6$  word rectangles.
- 25. [25] Explain how to use *orthogonal lists*, as in Fig. 13 of Section 2.2.6, so that it's easy to visit all 5-letter words whose  $k$ th character is  $c$ , given  $1 \leq k \leq 5$  and  $\mathbf{a} \leq c \leq \mathbf{z}$ . Use those sublists to speed up the algorithm of exercise 24.
26. [21] Can you find nice word rectangles of sizes  $5 \times 7$ ,  $5 \times 8$ ,  $5 \times 9$ ,  $5 \times 10$ ?
27. [22] What profile and average node costs replace (13) and (14) when we ask the algorithm of exercise 25 for  $6 \times 5$  word rectangles instead of  $5 \times 6$ ?
- 28. [23] The method of exercises 24 and 25 does  $n$  levels of backtracking to fill the cells of an  $m \times n$  rectangle one column at a time, using a trie to detect illegal prefixes in the rows. Devise a method that does  $mn$  levels of backtracking and fills just *one* cell per level, using tries for *both* rows and columns.
29. [15] What's the largest commafree subset of the following words?

aced babe bade bead beef cafe cede dada dead deaf face fade feed

- 30. [22] Let  $w_1, w_2, \dots, w_m$  be four-letter words on an  $m$ -letter alphabet. Design an algorithm that accepts or rejects each  $w_j$ , according as  $w_j$  is commafree or not with respect to the accepted words of  $\{w_1, \dots, w_{j-1}\}$ .
31. [M22] A two-letter block code on an  $m$ -letter alphabet can be represented as a digraph  $D$  on  $m$  vertices, with  $a \rightarrow b$  if and only if  $ab$  is a codeword.
- Prove that the code is commafree  $\iff D$  has no oriented paths of length 3.
  - How many arcs can be in a digraph with no oriented paths of length  $r$ ?
- 32. [M30] (W. L. Eastman, 1965.) The following elegant construction yields a commafree code of maximum size for any *odd* block length  $n$ , over any alphabet. Given a sequence of  $x = x_0x_1 \dots x_{n-1}$  of nonnegative integers, where  $x$  differs from each of its other cyclic shifts  $x_k \dots x_{n-1}x_0 \dots x_{k-1}$  for  $0 < k < n$ , the procedure outputs a cyclic shift  $\sigma x$  with the property that the set of all such  $\sigma x$  is commafree.

We regard  $x$  as an infinite periodic sequence  $\langle x_n \rangle$  with  $x_k = x_{k-n}$  for all  $k \geq n$ . Each cyclic shift then has the form  $x_kx_{k+1} \dots x_{k+n-1}$ . The simplest nontrivial example occurs when  $n = 3$ , where  $x = x_0x_1x_2x_0x_1x_2x_0 \dots$  and we don't have  $x_0 = x_1 = x_2$ . In this case the algorithm outputs  $x_kx_{k+1}x_{k+2}$  where  $x_k > x_{k+1} \leq x_{k+2}$ ; and the set of all such triples clearly satisfies the commafree condition.

domains  
dual  
loose Langford pairs  
word rectangle  
orthogonal lists  
trie  
commafree  
two-letter block code  
digraph  
commafree  
Eastman  
aperiodic words  
periodic sequence

One key idea is to think of  $x$  as partitioned into  $t$  substrings by boundary markers  $b_j$ , where  $0 \leq b_0 < b_1 < \dots < b_{t-1} < n$  and  $b_j = b_{j-t} + n$  for  $j \geq t$ . Then substring  $y_j$  is  $x_{b_j}x_{b_{j+1}} \dots x_{b_{j+1}-1}$ . The number  $t$  of substrings is always odd. Initially  $t = n$  and  $b_j = j$  for all  $j$ ; ultimately  $t = 1$ , and  $\sigma x = y_0$  is the desired output.

Eastman's algorithm is based on comparison of adjacent substrings  $y_{j-1}$  and  $y_j$ . If those substrings have the same length, we use lexicographic comparison; otherwise we declare that the longer substring is bigger.

The second key idea is the notion of "dips," which are substrings of the form  $z = z_1 \dots z_k$  where  $k \geq 2$  and  $z_1 \geq \dots \geq z_{k-1} < z_k$ . It's easy to see that any string  $y = y_0 y_1 \dots$  in which we have  $y_i < y_{i+1}$  for infinitely many  $i$  can be factored into a sequence of dips,  $y = z^{(0)} z^{(1)} \dots$ , and this factorization is unique. For example,

3141592653589793238462643383 ... = 314 15 926 535 89 79 323 846 26 4338 3 ...

Furthermore, if  $y$  is a periodic sequence, its factorization into dips is also ultimately periodic, although some of the initial factors may not occur in the period. For example,

123443550123443550123443550 ... = 12 34 435 501 23 4435 501 23 4435 ...

Given a periodic, nonconstant sequence  $y$  described by boundary markers as above, where the period length  $t$  is odd, its periodic factorization will contain an odd number of odd-length dips. Each round of Eastman's algorithm simply retains the boundary points at the left of those odd-length dips. Then  $t$  is reset to the number of retained boundary points, and another round begins if  $t > 1$ .

- Play through the algorithm by hand when  $n = 19$  and  $x = 3141592653589793238$ .
- Show that the number of rounds is at most  $\lfloor \log_3 n \rfloor$ .
- Exhibit a binary  $x$  that achieves this worst-case bound when  $n = 3^e$ .
- Implement the algorithm with full details. (It's surprisingly short!)
- Explain why the algorithm yields a commafree code.

**33.** [HM28] What is the probability that Eastman's algorithm finishes in one round? (Assume that  $x$  is a random  $m$ -ary string of odd length  $n > 1$ , unequal to any of its other cyclic shifts. Use a generating function to express the answer.)

**34.** [18] Why can't a commafree code of length  $(m^4 - m^2)/4$  contain 0001 and 2000?

► **35.** [15] Why do you think sequential data structures such as (16)–(23) weren't featured in Section 2.2.2 of this series of books (entitled "Sequential Allocation")?

**36.** [17] What's the significance of (a) MEM[40d] = 5e and (b) MEM[904] = 84 in Table 1?

**37.** [18] Why is (a) MEM[f8] = e7 and (b) MEM[a0d] = ba in Table 2?

**38.** [20] Suppose you're using the undoing scheme (26) and the operation  $\sigma \leftarrow \sigma + 1$  has just bumped the current stamp  $\sigma$  to zero. What should you do?

► **39.** [25] Spell out the low-level implementation details of the candidate selection process in step C2 of Algorithm C. Use the routine store( $a, v$ ) of (26) whenever changing the contents of MEM, and use the following selection strategy:

- Find a class  $c$  with the least number  $r$  of blue words.
- If  $r = 0$ , set  $x \leftarrow -1$ ; otherwise set  $x$  to a word in class  $c$ .
- If  $r > 1$ , use the poison list to find an  $x$  that maximizes the number of blue words that could be killed on the other side of the prefix or suffix list that contains  $x$ .

► **40.** [28] Continuing exercise 39, spell out the details of step C3 when  $x \geq 0$ .

- What updates should be done to MEM when a blue word  $x$  becomes red?
- What updates should be done to MEM when a blue word  $x$  becomes green?

substrings  
 boundary markers  
 lexicographic comparison  
 dips  
 pi, as "random" data  
 worst-case bound  
 analysis of algs  
 Eastman  
 generating function  
 analysis of algs  
 data structures  
 Sequential Allocation  
 undoing scheme  
 bumped  
 stamp  
 poison list

- c) Step C3 begins its job by making  $x$  green as in part (b). Explain how it should finish its job by updating the poison list.
42. [M30] Is there a *binary* ( $m = 2$ ) commafree code with one codeword in each of the  $(\sum_{d \mid n} \mu(d)2^{n/d})/n$  cycle classes, for every word length  $n$ ?
44. [HM29] A commafree code on  $m$  letters is equivalent to  $2m!$  such codes if we permute the letters and/or replace each codeword by its left-right reflection.  
Determine all of the nonisomorphic commafree codes of length 4 on  $m$  letters when  $m$  is (a) 2 (b) 3 (c) 4 and there are (a) 3 (b) 18 (c) 57 codewords.
45. [M42] Find a maximum-size commafree code of length 4 on  $m = 5$  letters.
47. [20] Explain how the choices in Fig. 69 were determined from the “random” bits that are displayed. For instance, why was  $X_2$  set to 1 in Fig. 69(b)?
48. [M15] Interpret the value  $E(D_1 \dots D_l)$ , in the text’s Monte Carlo algorithm.
49. [M22] What’s a simple martingale that corresponds to Theorem E?
- 50. [HM25] Elmo uses Algorithm E with  $D_k = \{1, \dots, n\}$ ,  $P_l = [x_1 > \dots > x_l]$ ,  $c = 1$ .  
a) Alice flips  $n$  coins independently, where coin  $k$  yields “heads” with probability  $1/k$ . True or false: She obtains exactly  $l$  heads with probability  $\binom{n}{l}/n!$ .  
b) Let  $Y_1, Y_2, \dots, Y_l$  be the numbers on the coins that come up heads. (Thus  $Y_1 = 1$ , and  $Y_2 = 2$  with probability  $1/2$ .) Show that  $\Pr(\text{Alice obtains } Y_1, Y_2, \dots, Y_l) = \Pr(\text{Elmo obtains } X_1 = Y_l, X_2 = Y_{l-1}, \dots, X_l = Y_1)$ .  
c) Prove that Alice q.s. obtains at most  $(\ln n)(\ln \ln n)$  heads.  
d) Consequently Elmo’s  $S$  is q.s. less than  $\exp((\ln n)^2(\ln \ln n))$ .
- 51. [M30] Extend Algorithm B so that it also computes the minimum, maximum, mean, and variance of the Monte Carlo estimates  $S$  produced by Algorithm E.
52. [M21] Instead of choosing each  $y_i$  in step E5 with probability  $1/d$ , we could use a biased distribution where  $\Pr(I = i | X_1, \dots, X_{l-1}) = p_{X_1 \dots X_{l-1}}(y_i) > 0$ . How should the estimate  $S$  be modified so that its expected value in this general scheme is still  $C()$ ?
53. [M20] If all costs  $c(x_1, \dots, x_l)$  are positive, show that the biased probabilities of exercise 52 can be chosen in such a way that the estimate  $S$  is always exact.
- 55. [M25] The commafree code search procedure in Algorithm C doesn’t actually fit the mold of Algorithm E, because it incorporates lookahead, dynamic ordering, reversible memory, and other enhancements to the basic backtrack paradigms. How could its running time be reliably estimated with Monte Carlo methods?
57. [M20] Algorithm E can potentially follow  $M$  different paths  $X_1 \dots X_{l-1}$  before it terminates, where  $M$  is the number of leaves of the backtrack tree. Suppose the final values of  $D$  at those leaves are  $D^{(1)}, \dots, D^{(M)}$ . Prove that  $(D^{(1)} \dots D^{(M)})^{1/M} \geq M$ .
58. [27] The text’s king path problem is a special case of the general problem of counting simple paths from vertex  $s$  to vertex  $t$  in a given graph.  
We can generate such paths by random walks from  $s$  that don’t get stuck, if we maintain a table of values  $\text{DIST}(v)$  for all vertices  $v$  not yet in the path, representing the shortest distance from  $v$  to  $t$  through unused vertices. For with such a table we can simply move at each step to a vertex for which  $\text{DIST}(v) < \infty$ .  
Devise a way to update the  $\text{DIST}$  table dynamically without unnecessary work.
59. [26] A ZDD with 3,174,197 nodes can be constructed for the family of all simple corner-to-corner king paths on a chessboard, using the method of exercise 7.1.4–225. Explain how to use this ZDD to compute (a) the total length of all paths; (b) the number of paths that touch any given subset of the center and/or corner points.

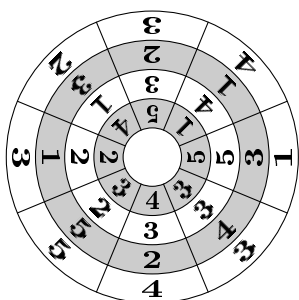
nonisomorphic symmetries  
“random” bits  
Monte Carlo martingale coins  
Stirling cycle numbers  
q.s.  
variance  
biased distribution  
commafree code  
lookahead  
dynamic ordering  
reversible memory  
simple paths  
random walks  
dynamic shortest distances  
shortest distances, dynamic  
ZDD  
corner-to-corner  
king paths  
chessboard

- ▶ **60.** [20] Experiment with biased random walks (see exercise 52), weighting each non-dead-end king move to a new vertex  $v$  by  $1 + \text{DIST}(v)^2$  instead of choosing every such move with the same probability. Does this strategy improve on Fig. 70?
- 61.** [HM26] Let  $P_n$  be the number of integer sequences  $x_1 \dots x_n$  such that  $x_1 = 1$  and  $1 \leq x_{k+1} \leq 2x_k$  for  $1 \leq k < n$ . (The first few values are 1, 2, 6, 26, 166, 1626, ...; this sequence was introduced by A. Cayley in *Philosophical Magazine* (4) **13** (1857), 245–248, who showed that  $P_n$  enumerates the partitions of  $2^n - 1$  into powers of 2.)
  - a) Show that  $P_n$  is the number of different profiles that are possible for a binary tree of height  $n$ .
  - b) Find an efficient way to compute  $P_n$  for large  $n$ . *Hint:* Consider the more general sequence  $P_n^{(m)}$ , defined similarly but with  $x_1 = m$ .
  - c) Use the estimation procedure of Theorem E to prove that  $P_n \geq 2^{\binom{n}{2}} / (n-1)!$ .
- ▶ **66.** [22] When the faces of four cubes are colored randomly with four colors, estimate the probability that the corresponding “Instant Insanity” puzzle has a unique solution. How many 2-regular graphs tend to appear during the “factored” solution process?
- 67.** [20] Find *five* cubes, each of whose faces has one of *five* colors, and where every color occurs at least five times, such that the corresponding puzzle has a unique solution.
- 70.** [24] Assemble five cubes with uppercase letters on each face, using the patterns

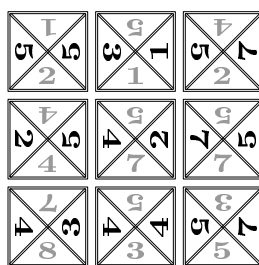


By extending the principles of Instant Insanity, show that these cubes can be placed in a row so that four 5-letter words are visible. (Each word’s letters should have a consistent orientation. The letters C and U, H and I, N and Z are related by 90° rotation.)

- ▶ **73.** [23] (*The Fool’s Disk*.) “Rotate the four disks of the lefthand illustration below so that the four numbers on each ray sum to 12.” (The current sums are  $4 + 3 + 2 + 4 = 13$ , etc.) Show that this problem factors nicely, so that it can be solved readily by hand.



The Fool’s Disk



The Royal Aquarium Thirteen Puzzle

- ▶ **75.** [26] (*The Royal Aquarium Thirteen Puzzle*.) “Rearrange the nine cards of the righthand illustration above, optionally rotating some of them by 180°, so that the six horizontal sums of gray letters and the six vertical sums of black letters all equal 13.” (The current sums are  $1 + 5 + 4 = 10$ , ...,  $7 + 5 + 7 = 19$ .) The author of *Hoffmann’s Puzzles Old and New* (1893) stated that “There is no royal road to the solution. The proper order must be arrived at by successive transpositions until the conditions are fulfilled.” Prove that he was wrong: “Factor” this problem and solve it by hand.

biased random walks  
 Cayley  
 binary partitions  
 partitions  
 profiles  
 height  $n$   
 Instant Insanity  
 factored  
 speedy schizophrenia  
 cubes  
 uppercase letters  
 alphabet  
 Instant Insanity  
 5-letter words  
 Fool’s Disk  
 Royal Aquarium Thirteen Puzzle  
 Le Nombre Treize, see Royal Aquarium Thirteen Puzzle  
 Hoffmann  
 Factor

**Table 666**

TWENTY QUESTIONS (SEE EXERCISE 90)

Woods  
questionnaire  
paradoxical

- 
1. The first question whose answer is A is:  
(A) 1      (B) 2      (C) 3      (D) 4      (E) 5
  2. The next question with the same answer as this one is:  
(A) 4      (B) 6      (C) 8      (D) 10      (E) 12
  3. The only two consecutive questions with identical answers are questions:  
(A) 15 and 16    (B) 16 and 17    (C) 17 and 18    (D) 18 and 19    (E) 19 and 20
  4. The answer to this question is the same as the answers to questions:  
(A) 10 and 13    (B) 14 and 16    (C) 7 and 20    (D) 1 and 15    (E) 8 and 12
  5. The answer to question 14 is:  
(A) B      (B) E      (C) C      (D) A      (E) D
  6. The answer to this question is:  
(A) A      (B) B      (C) C      (D) D      (E) none of those
  7. An answer that appears most often is:  
(A) A      (B) B      (C) C      (D) D      (E) E
  8. Ignoring answers that appear equally often, the least common answer is:  
(A) A      (B) B      (C) C      (D) D      (E) E
  9. The sum of all question numbers whose answers are correct and the same as this one is:  
(A)  $\in [59 \dots 62]$     (B)  $\in [52 \dots 55]$     (C)  $\in [44 \dots 49]$     (D)  $\in [61 \dots 67]$     (E)  $\in [44 \dots 53]$
  10. The answer to question 17 is:  
(A) D      (B) B      (C) A      (D) E      (E) wrong
  11. The number of questions whose answer is D is:  
(A) 2      (B) 3      (C) 4      (D) 5      (E) 6
  12. The number of *other* questions with the same answer as this one is the same as the number of questions with answer:  
(A) B      (B) C      (C) D      (D) E      (E) none of those
  13. The number of questions whose answer is E is:  
(A) 5      (B) 4      (C) 3      (D) 2      (E) 1
  14. No answer appears exactly this many times:  
(A) 2      (B) 3      (C) 4      (D) 5      (E) none of those
  15. The set of odd-numbered questions with answer A is:  
(A) {7}      (B) {9}      (C) not {11}    (D) {13}      (E) {15}
  16. The answer to question 8 is the same as the answer to question:  
(A) 3      (B) 2      (C) 13      (D) 18      (E) 20
  17. The answer to question 10 is:  
(A) C      (B) D      (C) B      (D) A      (E) correct
  18. The number of prime-numbered questions whose answers are vowels is:  
(A) prime    (B) square    (C) odd      (D) even      (E) zero
  19. The last question whose answer is B is:  
(A) 14      (B) 15      (C) 16      (D) 17      (E) 18
  20. The maximum score that can be achieved on this test is:  
(A) 18      (B) 19      (C) 20      (D) indeterminate  
(E) achievable only by getting this question wrong
- 

► 90. [M29] (Donald R. Woods, 2000.) Find all ways to maximize the number of correct answers to the questionnaire in Table 666. Each question must be answered with a letter from A to E. *Hint:* Begin by clarifying the exact meaning of this exercise. What answers are best for the following two-question, two-letter “warmup problem”?

1. (A) Answer 2 is B.      (B) Answer 1 is A.
2. (A) Answer 1 is correct.    (B) Either answer 2 is wrong or answer 1 is A, but not both.

91. [HM28] Show that exercise 90 has a surprising, somewhat paradoxical answer if two changes are made to Table 666: 9(E) becomes ‘ $\in [39 \dots 43]$ ’; 15(C) becomes ‘{11}’.

- **95.** [30] (*A clueless anacrostic.*) The letters of 29 five-letter words

$\overline{1\ 2\ 3\ 4\ 5}$ ,  $\overline{6\ 7\ 8\ 9\ 10}$ ,  $\overline{11\ 12\ 13\ 14\ 15}$ ,  $\overline{16\ 17\ 18\ 19\ 20}$ , ...,  $\overline{141\ 142\ 143\ 144\ 145}$ ,

all belonging to WORDS(1000), have been shuffled to form the following mystery text:

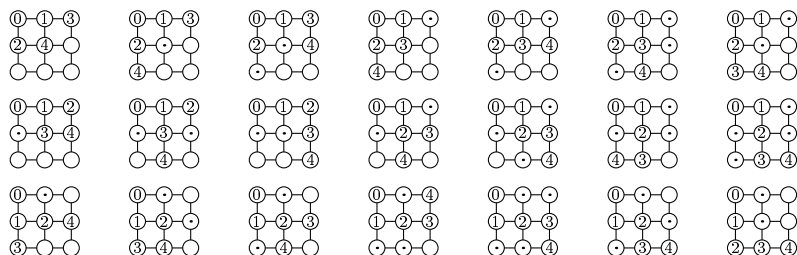
$\overline{30\ 29\ 9}$   $\overline{140\ 12\ 13\ 145\ 90\ 45\ 99}$   $\overline{26\ 107}$   $\overline{47\ 84\ 53\ 51\ 27\ 133\ 39}$   $\overline{137\ 139}$   $\overline{66\ 112\ 69\ 14\ 8\ 20\ 91\ 129\ 70}$   
 $\overline{16\ 7\ 93\ 19\ 85}$   $\overline{101\ 76\ 78\ 44\ 10\ 106\ 60}$   $\overline{118\ 119}$   $\overline{24\ 25\ 100}$   $\overline{1\ 5\ 64\ 11\ 71}$   $\overline{42\ 122\ 123}$   
 $\overline{103\ 104\ 63\ 49\ 31\ 121\ 98\ 79\ 80}$   $\overline{46\ 48}$   $\overline{134\ 135\ 131}$   $\overline{143\ 96\ 142\ 120\ 50\ 132\ 33\ 43\ 34\ 40}$  ...  
 $\overline{111\ 97\ 113\ 105\ 38\ 102\ 62\ 65\ 114}$   $\overline{74\ 82\ 81\ 83\ 136\ 37\ 21\ 61\ 88\ 86\ 55}$  ( $\overline{32\ 35}$   $\overline{117\ 116\ 23\ 52}$   
 $\overline{56\ 17\ 18\ 94\ 67}$   $\overline{128\ 15\ 57\ 58\ 89}$   $\overline{87\ 109}$   $\overline{2\ 4\ 6\ 28\ 95\ 3\ 126\ 77\ 144\ 54\ 41}$ )  $\overline{68\ 115}$   
 $\overline{75\ 138\ 73\ 124\ 36\ 130\ 127\ 141}$   $\overline{22\ 92}$   $\overline{72\ 59}$   $\overline{108\ 125\ 110}$

clueless  
 anacrostic  
 five-letter words  
 WORDS(n)  
 mystery text  
 English words  
 factorization  
 Connected subsets  
 canonical  
 grid  
 pentominoes  
 spanning tree  
 lexicographically smallest  
 SGB format  
 ARCS  
 TIP  
 NEXT  
 polyominoes

Furthermore, their initial letters  $\overline{1}$ ,  $\overline{6}$ ,  $\overline{11}$ ,  $\overline{16}$ , ...,  $\overline{141}$  identify the source of that quotation, which consists entirely of common English words. What does it say?

- **96.** [21] The fifteenth mystery word in exercise 95 is ' $\overline{134\ 135\ 131}$ '. Why does its special form lead to a partial *factorization* of that problem?
- **100.** [30] (*Connected subsets.*) Let  $v$  be a vertex of some graph  $G$ , and let  $H$  be a connected subset of  $G$  that contains  $v$ . The vertices of  $H$  can be listed in a canonical way by starting with  $v_0 \leftarrow v$  and then letting  $v_1, v_2, \dots$  be the neighbors of  $v_0$  that lie in  $H$ , followed by the neighbors of  $v_1$  that haven't yet been listed, and so on. (We assume that the neighbors of each vertex are listed in some fixed order.)

For example, if  $G$  is the  $3 \times 3$  grid  $P_3 \square P_3$ , exactly 21 of its connected five-element subsets contain the upper left corner element  $v$ . Their canonical orderings are



if we order the vertices from top to bottom and left to right when listing a vertex's neighbors. (Vertices labeled 0, 1, 2, 3, 4 indicate  $v_0, v_1, v_2, v_3, v_4$ . Other vertices are not in  $H$ .) Notice that in each case the numbered vertices are implicitly connected by a spanning tree whose edges each have the form  $v_i - v_j$  for some  $i < j$ .

The canonical ordering corresponds to  $H$ 's lexicographically smallest spanning tree; for example, the spanning tree in the first solution is  $v_0 - v_1, v_0 - v_2, v_1 - v_3, v_1 - v_4$ . Furthermore, the 21 solutions appear here in lexicographic order of their respective spanning trees.

Design a backtrack algorithm to generate all of the  $n$ -element connected subsets that contain a specified vertex  $v$ , given a graph that is represented in SGB format (which has ARCS, TIP, and NEXT fields, as described near the beginning of Chapter 7).

- **101.** [23] Use the algorithm of exercise 100 to generate *all* of the connected  $n$ -element subsets of a given graph  $G$ . How many such subsets does  $P_n \square P_n$  have, for  $1 \leq n \leq 9$ ?

**102.** [M22] A  $v$ -reachable subset of a directed graph  $G$  is a nonempty set of vertices  $H$  with the property that every  $u \in H$  can be reached from  $v$  by at least one oriented path in  $G|H$ . (In particular,  $v$  itself must be in  $H$ .)

- a) The digraph  $P_3^{\rightarrow} \square P_3^{\rightarrow}$  is like  $P_3 \square P_3$ , except that all arcs between vertices are directed downward or to the right. Which of the 21 connected subsets in exercise 100 are also  $v$ -reachable from the upper left corner element  $v$  of  $P_3^{\rightarrow} \square P_3^{\rightarrow}$ ?
- b) True or false:  $H$  is  $v$ -reachable if and only if  $G|H$  contains a dual oriented spanning tree rooted at  $v$ . (An oriented tree has arcs  $u \rightarrow p_u$ , where  $p_u$  is the parent of the nonroot node  $u$ ; in a *dual* oriented tree, the arcs are reversed:  $p_u \rightarrow u$ .)
- c) True or false: If  $G$  is undirected, so that  $w \rightarrow u$  whenever  $u \rightarrow w$ , its  $v$ -reachable subsets are the same as the connected subsets that contain  $v$ .
- d) Modify the algorithm of exercise 100 so that it generates all of the  $n$ -element  $v$ -reachable subsets of a digraph  $G$ , given  $n$ ,  $v$ , and  $G$ .

**999.** [M00] this is a temporary exercise (for dummies)

$v$ -reachable subset  
 reachable subsets  
 grid, oriented  
 oriented grid  
 dual oriented spanning tree  
 oriented tree  
 parent  
 directed graph versus undirected  
 undirected graph versus directed



## SECTION 7.2.2

1. Although many formulations are possible, the following may be the nicest: (i)  $D_k$  is arbitrary (but hopefully finite), and  $P_l$  is always true. (ii)  $D_k = \{1, 2, \dots, n\}$  and  $P_l = 'x_j \neq x_k \text{ for } 1 \leq j < k \leq l'$ . (iii) For combinations of  $n$  things from  $N$ ,  $D_k = \{1, \dots, N + 1 - k\}$  and  $P_l = 'x_1 > \dots > x_l'$ . (iv)  $D_k = \{0, 1, \dots, \lfloor n/k \rfloor\}$ ;  $P_l = 'x_1 \geq \dots \geq x_l \text{ and } n - (n-l)x_l \leq x_1 + \dots + x_l \leq n'$ . (v) For restricted growth strings,  $D_k = \{0, \dots, k-1\}$  and  $P_l = 'x_{j+1} \leq 1 + \max(x_1, \dots, x_j) \text{ for } 1 \leq j < l'$ . (vi) For indices of left parentheses (see 7.2.1.6–(8)),  $D_k = \{1, \dots, 2k - 1\}$  and  $P_l = 'x_1 < \dots < x_l'$ .

2. True. (If not, set  $D_1 \leftarrow D_1 \cap \{x \mid P_1(x)\}$ .)

3. We can restrict  $D_1$  to  $\{1, 2, 3, 4\}$ , because the reflection  $(9-x_1)\dots(9-x_8)$  of every solution  $x_1 \dots x_8$  is also a solution. (H. C. Schumacher made this observation in a letter to Gauss, 24 September 1850.) Notice that Fig. 68 is left-right symmetric.

4.  $try(l) =$  “If  $l > n$ , visit  $x_1 \dots x_n$ . Otherwise, for  $x_l \leftarrow \min D_l, \min D_l + 1, \dots, \max D_l$ , if  $P_l(x_1, \dots, x_l)$  call  $try(l + 1)$ .”

This formulation is elegant, and fine for simple problems. But it doesn't give any clue about why the method is called “backtrack”! Nor does it yield efficient code for important problems whose inner loop is performed billions of times. We will see that the key to efficient backtracking is to provide good ways to update and downgrade the data structures that speed up the testing of property  $P_l$ . The overhead of recursion can get in the way, and the actual iterative structure of Algorithm B isn't difficult to grasp.

5. Excluding cases with  $j = r$  or  $k = r$  from (3) yields respectively (312, 396, 430, 458, 458, 430, 396, 312) solutions. (With column  $r$  also omitted there are just (40, 46, 42, 80, 80, 42, 46, 40).)

6. Yes, almost surely for all  $n > 16$ . One such is  $x_1 x_2 \dots x_{17} = 2 \ 17 \ 12 \ 10 \ 7 \ 14 \ 3 \ 5 \ 9 \ 13 \ 15 \ 4 \ 11 \ 8 \ 6 \ 1 \ 16$ . [See *Proc. Edinburgh Math. Soc.* **8** (1890), 43 and Fig. 52.] Preußer and Engelhardt found 34,651,355,392 solutions when  $n = 27$ .

7. Yes: (42736815, 42736851); also therefore (57263148, 57263184).

8. Yes, at least when  $m = 4$ ; e.g.,  $x_1 \dots x_{16} = 5 \ 8 \ 13 \ 16 \ 3 \ 7 \ 15 \ 11 \ 6 \ 2 \ 10 \ 14 \ 1 \ 4 \ 9 \ 12$ . There are no solutions when  $m = 5$ , but 7 10 13 20 17 24 3 6 23 11 16 21 4 9 14 2 19 22 1 8 5 12 15 18 works for  $m = 6$ . (Are there solutions for all even  $m \geq 4$ ?) C. F. de Jaenisch, *Traité des applications de l'analyse mathématique au jeu des échecs* **2** (1862), 132–133, noted that all 8-queen solutions have four of each color. He proved that the number of white queens must be even, because  $\sum_{k=1}^{4m} (x_k + k)$  is even.)

9. Let bit vectors  $a_l, b_l, c_l$  represent the “useful” elements of the sets in (6), with  $a_l = \sum \{2^{x-1} \mid x \in A_l\}$ ,  $b_l = \sum \{2^{x-1} \mid x \in B_l \cap [1..n]\}$ ,  $c_l = \sum \{2^{x-1} \mid x \in C_l \cap [1..n]\}$ . Then step W2 sets  $s_l \leftarrow \mu \& \bar{a}_l \& \bar{b}_l \& \bar{c}_l$ , where  $\mu$  is the mask  $2^n - 1$ .

In step W3 we can set  $t \leftarrow s_l \& (-s_l)$ ,  $a_l \leftarrow a_{l-1} + t$ ,  $b_l \leftarrow (b_{l-1} + t) \gg 1$ ,  $c_l \leftarrow ((c_{l-1} + t) \ll 1) \& \mu$ ; and it's also convenient to set  $s_l \leftarrow s_l - t$  at this time, instead of deferring this change to step W4.

(There's no need to store  $x_l$  in memory, or even to compute  $x_l$  in step W3 as an integer in  $[1..n]$ , because  $x_l$  can be deduced from  $a_l - a_{l-1}$  when a solution is found.)

10. (a) Only when  $n = 1$ , because reflected queens can capture each other.

(b) Queens not in the center must appear in groups of four.

(c) The four queens occupy the same rows, columns, and diagonals in both cases.

(d) In each solution counted by  $c_n$  we can independently tilt (or not) each of the  $\lfloor n/4 \rfloor$  groups of four. [*Mathematische Unterhaltungen und Spiele* **1**, second edition (Leipzig: Teubner, 1910), 249–258.]

restricted growth strings  
reflection  
Schumacher  
Gauss  
symmetric  
inner loop  
backtracking, efficient  
iteration versus recursion  
Preußer  
Engelhardt  
de Jaenisch  
mask

11. Suppose the  $x_k$  are distinct. Then  $\sum_{k=1}^n (x_k + k) = 2 \binom{n+1}{2} \equiv 0 \pmod{n}$ . If the numbers  $(x_k + k) \pmod{n}$  are also distinct, we have also  $\sum_{k=1}^n k \equiv \binom{n+1}{2}$ . But that is impossible when  $n$  is even.

Now suppose further that the numbers  $(x_k - k) \pmod{n}$  are distinct. Then we have  $\sum_{k=1}^n (x_k + k)^2 \equiv \sum_{k=1}^n (x_k - k)^2 \equiv \sum_{k=1}^n k^2 = n(n+1)(2n+1)/6$ . And we also have  $\sum_{k=1}^n (x_k + k)^2 + \sum_{k=1}^n (x_k - k)^2 = 4n(n+1)(2n+1)/6 \equiv 2n/3$ , which is impossible when  $n$  is a multiple of 3. [See W. Ahrens, *Mathematische Unterhaltungen und Spiele* 2, second edition (1918), 364–366, where G. Pólya cites a more general result of A. Hurwitz that applies to wraparound diagonals of other slopes.]

Conversely, if  $n$  isn't divisible by 2 or 3, we can let  $x_n = n$  and  $x_k = (2k) \pmod{n}$  for  $1 \leq k < n$ . (The rule  $x_k = (3k) \pmod{n}$  also works. See Édouard Lucas, *Récréations Mathématiques* 1 (1882), 84–86.)

12. The  $(n+1)$  queens problem clearly has a solution with a queen in a corner if and only if the  $n$  queens problem has a solution with a queen-free main diagonal. Hence by the previous answer there's always a solution when  $n \pmod{6} \in \{0, 1, 4, 5\}$ .

Another nice solution was found by J. Franel [L'Intermédiaire des Mathématiciens 1 (1894), 140–141] when  $n \pmod{6} \in \{2, 4\}$ : Let  $x_k = (n/2 + 2k - 3[2k \leq n]) \pmod{n+1}$ , for  $1 \leq k \leq n$ . With this setup we find that  $x_k - x_j = \pm(k - j)$  and  $1 \leq j < k \leq n$  implies  $(1 \text{ or } 3)(k - j) + (0 \text{ or } 3) \equiv 0 \pmod{n}$ ; hence  $k - j = n - (1 \text{ or } 3)$ . But the values of  $x_1, x_2, x_3, x_{n-2}, x_{n-1}, x_n$  give no attacking queens except when  $n = 2$ .

Franel's solution has empty diagonals, so it provides solutions also for  $n \pmod{6} \in \{3, 5\}$ . We conclude that only  $n = 2$  and  $n = 3$  are impossible.

[A more complicated construction for all  $n > 3$  had been given earlier by E. Pauls, in *Deutsche Schachzeitung* 29 (1874), 129–134, 257–267. Pauls also explained how to find all solutions, in principle, by building the tree level by level (*not* backtracking).]

13. For  $1 \leq j \leq n$ , let  $x_1^{(j)} \dots x_m^{(j)}$  be a solution for  $m$  queens, and let  $y_1 \dots y_n$  be a solution for  $n$  toroidal queens. Then  $X_{(i-1)n+j} = (x_i^{(j)} - 1)n + y_j$  (for  $1 \leq i \leq m$  and  $1 \leq j \leq n$ ) is a solution for  $mn$  queens. [I. Rivin, I. Vardi, and P. Zimmermann, *AMM* 101 (1994), 629–639, Theorem 2.]

14. [Rivin, Vardi, and Zimmermann, in the paper just cited, observe that in fact the sequence  $(\ln Q(n))/(n \ln n)$  appears to be *increasing*.]

15. Let the queen in row  $k$  be in cell  $k$ . Then we have a “relaxation” of the  $n$  queens problem, with  $|x_k - x_j|$  becoming just  $x_k - x_j$  in (3); so we can ignore the  $b$  vector in Algorithm B\* or in exercise 9. We get

$$\begin{array}{cccccccccccccccc} n = & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ H(n) = & 1 & 1 & 1 & 3 & 7 & 23 & 83 & 405 & 2113 & 12657 & 82297 & 596483 & 4698655 & 40071743 & 367854835 \end{array}$$

[N. J. Cavenagh and I. M. Wanless, *Discr. Appl. Math.* 158 (2010), 136–146, Table 2.]

16. It fails spectacularly in step L5. The minus signs, which mark decisions that were previously forced, are crucial tags for backtracking.

17.  $x_4 \dots x_8 = \bar{2}10\bar{4}0$ ,  $p_0 \dots p_4 = 33300$ , and  $y_1 y_2 y_3 = 130$ . (If  $x_i \leq 0$  the algorithm will never look at  $y_i$ ; hence the current state of  $y_4 \dots y_8$  is irrelevant. But  $y_4 y_5$  happens to be 20, because of past history;  $y_6, y_7$ , and  $y_8$  haven't yet been touched.)

19. We could say  $D_l$  is  $\{-n, \dots, -2, -1, 1, 2, \dots, n\}$ , or  $\{k \mid k \neq 0 \text{ and } 2 - l \leq k \leq 2n - l - 1\}$ , or anything in between. (But this observation isn't very useful.)

Ahrens  
Pólya  
Hurwitz  
Lucas  
Franel  
Pauls  
tree  
breadth first search  
Rivin  
Vardi  
Zimmermann  
semi-queens  
Cavenagh  
Wanless

**20.** First we add a Boolean array  $a_1 \dots a_n$ , where  $a_k$  means “ $k$  has appeared,” as in Algorithm B\*. It’s  $0 \dots 0$  in step L1; we set  $a_k \leftarrow 1$  in step L3,  $a_k \leftarrow 0$  in step L5.

The loop in step L2 becomes “while  $x_l < 0$ , go to L5 if  $l \geq n - 1$  and  $a_{2n-l-1} = 0$ , otherwise set  $l \leftarrow l + 1$ .” After finding  $l + k + 1 \leq 2n$  in L3, and before testing  $x_{l+k+1}$  for 0, insert this: “If  $l \geq n - 1$  and  $a_{2n-l-1} = 0$ , while  $l + k + 1 \neq 2n$  set  $j \leftarrow k$ ,  $k \leftarrow p_k$ .”

**21.** (a) In any solution  $x_k = n \iff x_{k+n+1} = -n \iff x_{n-k}^D = n$ .

(b)  $x_k = n - 1$  for some  $k \leq n/2$  if and only if  $x_k^D = n - 1$  for some  $k > n/2$ .

(c) Let  $n' = n - [n \text{ is even}]$ . Change ‘ $l \geq n - 1$  and  $a_{2n-l-1} = 0$ ’ in the modified step L2 to ‘ $l = \lfloor n/2 \rfloor$  and  $a_{n'} = 0$ ’ or ‘ $l \geq n - 1$  and  $a_{2n-l-1} = 0$ ’. Insert the following before the other insertion into step L3: “If  $l = \lfloor n/2 \rfloor$  and  $a_{n'} = 0$ , while  $k \neq n'$  set  $j \leftarrow k$ ,  $k \leftarrow p_k$ .” And in step L5—this subtle detail is needed when  $n$  is even—go to L5 instead of L4 if  $l = \lfloor n/2 \rfloor$  and  $k = n'$ .

**22.** The solutions  $1\bar{1}$  and  $2\bar{1}\bar{1}\bar{2}$  for  $n = 1$  and  $n = 2$  are self-dual; the solutions for  $n = 4$  and  $n = 5$  are  $431\bar{1}\bar{2}\bar{3}\bar{4}\bar{2}$ ,  $245\bar{2}31\bar{1}\bar{4}\bar{3}\bar{5}$ ,  $451\bar{1}\bar{2}3\bar{4}\bar{2}\bar{5}\bar{3}$ , and their duals. The total number of solutions for  $n = 1, 2, \dots$  is  $1, 1, 0, 2, 4, 20, 0, 156, 516, 2008, 0, 52536, 297800, 1767792, 0, 75678864, \dots$ ; there are none when  $n \bmod 4 = 3$ , by a parity argument.

Algorithm L needs only obvious changes. To compute solutions by a streamlined method like exercise 21, use  $n' = n - (0, 1, 2, 0)$  and substitute ‘ $l = \lfloor n/4 \rfloor + (0, 1, 2, 1)$ ’ for ‘ $l = \lfloor n/2 \rfloor$ ’, when  $n \bmod 4 = (0, 1, 2, 3)$ ; also replace ‘ $l \geq n - 1$  and  $a_{2n-l-1} = 0$ ’ by ‘ $l \geq \lfloor n/2 \rfloor$  and  $a_{\lfloor (4n+2-2l)/3 \rfloor} = 0$ ’. The case  $n = 15$  is proved impossible with 397 million nodes and 9.93 gigamems.

**23.**  $\text{slums} \rightarrow \text{sluff, slump, slurs, slurp, or sluts; (slums, total)} \rightarrow \text{(slams, tonal)}$ .

**24.** Build the list of 5-letter words and the trie of 6-letter words in step B1; also set  $a_{01}a_{02}a_{03}a_{04}a_{05} \leftarrow 00000$ . Use  $\min D_l = 1$  in step B2 and  $\max D_l = 5757$  in step B4. Testing  $P_l$  in step B3, if word  $x_3$  is  $c_1c_2c_3c_4c_5$ , consists of forming  $a_{i1} \dots a_{i5}$ , where  $a_{lk} = \text{trie}[a_{(l-1)k}, c_k]$  for  $1 \leq k \leq 5$ ; but jump to B4 if any  $a_{lk}$  is zero.

**25.** There are  $5 \times 26$  singly linked lists, accessed from pointers  $h_{kc}$ , all initially zero. The  $x$ th word  $c_{x1}c_{x2}c_{x3}c_{x4}c_{x5}$ , for  $1 \leq x \leq 5757$ , belongs to 5 lists and has five pointers  $l_{x1}l_{x2}l_{x3}l_{x4}l_{x5}$ . To insert it, set  $l_{xk} \leftarrow h_{kc_{xk}}$ ,  $h_{kc_{xk}} \leftarrow x$ , and  $s_{kc_{xk}} \leftarrow s_{kc_{xk}} + 1$ , for  $1 \leq k \leq 5$ . (Thus  $s_{kc}$  will be the length of the list accessed from  $h_{kc}$ .)

We can store a “signature”  $\sum_{c=1}^{26} 2^{c-1} [\text{trie}[a, c] \neq 0]$  with each node  $a$  of the trie. For example, the signature for node 260 is  $2^0 + 2^4 + 2^8 + 2^{14} + 2^{17} + 2^{20} + 2^{24} = \#1124111$ , according to (11); here  $a \leftrightarrow 1, \dots, z \leftrightarrow 26$ .

The process of running through all  $x$  that match a given signature  $y$  with respect to position  $z$ , as needed in steps B2 and B4, now takes the following form: (i) Set  $i \leftarrow 0$ . (ii) While  $2^i \& y = 0$ , set  $i \leftarrow i + 1$ . (iii) Set  $x \leftarrow h_{z(i+1)}$ ; go to (vi) if  $x = 0$ . (iv) Visit  $x$ . (v) Set  $x \leftarrow l_{xz}$ ; go to (iv) if  $x \neq 0$ . (vi) Set  $i \leftarrow i + 1$ ; go to (ii) if  $2^i \leq y$ .

Let  $\text{trie}[a, 0]$  be the signature of node  $a$ . We choose  $z$  and  $y = \text{trie}[a_{(l-1)z}, 0]$  in step B2 so that the number of nodes to visit,  $\sum_{c=1}^{26} s_{zc} [2^{c-1} \& y \neq 0]$ , is minimum for  $1 \leq z \leq 5$ . For example, when  $l = 3$ ,  $x_1 = 1446$ , and  $x_2 = 185$  as in (10), that sum for  $z = 1$  is  $s_{11} + s_{15} + s_{19} + s_{1(15)} + s_{1(18)} + s_{1(21)} + s_{1(25)} = 296 + 129 + 74 + 108 + 268 + 75 + 47 = 997$ ; and the sums for  $z = 2, 3, 4, 5$  are 4722, 1370, 5057, and 1646. Hence we choose  $z = 1$  and  $y = \#1124111$ ; only 997 words, not 5757, need be tested for  $x_3$ .

The values  $y_l$  and  $z_l$  are maintained for use in backtracking. (In practice we keep  $x, y$ , and  $z$  in registers during most of the computation. Then we set  $x_l \leftarrow x, y_l \leftarrow y, z_l \leftarrow z$  before increasing  $l \leftarrow l + 1$  in step B3; and we set  $x \leftarrow x_l, y \leftarrow y_l, z \leftarrow z_l$  in

parity argument  
trie  
signature  
trie  
bitwise AND

step B5. We also keep  $i$  in a register, while traversing the sublists as above; this value is restored in step B5 by setting it to the  $z$ th letter of word  $x$ , decreased by 'a'.)

**26.** Here are the author's favorite  $5 \times 7$  and  $5 \times 8$ , and the *only*  $5 \times 9$ 's:

|         |          |           |            |
|---------|----------|-----------|------------|
| smashes | grandest | pastelist | varistors  |
| partial | renounce | accidence | agentival  |
| immense | episodes | mortgagor | coelomate  |
| emerged | basement | proreform | undeleated |
| sadness | eyesores | andesytes | oysterers  |

author  
Gattegno  
compressed trie  
trie, compressed  
saturating ternary addition

No  $5 \times 10$  word rectangles exist, according to our ground rules.

**27.** (1, 15727, 8072679, 630967290 90962081, 625415) and (15727.0, 4321.6, 1749.7, 450.4, 286.0). Total time  $\approx 18.3$  teramems. (In Section 7.2.2.1 we'll study a method that is symmetrical between rows and columns.)

**28.** Build a separate trie for the  $m$ -letter words; but instead of having trie nodes of size 26 as in (11), it's better to convert this trie into a *compressed* representation that omits the zeros. For example, the compressed representation of the node for prefix 'corne' in (12) consists of five consecutively stored pairs of entries ('a', 3879), ('d', 3878), ('l', 9602), ('r', 171), ('t', 5013), followed by (0, 0). Similarly, each shorter prefix with  $c$  descendants is represented by  $c$  consecutive pairs (character, link), followed by (0, 0) to mark the end of the node. Steps B3 and B4 are now very convenient.

Level  $l$  corresponds to row  $i_l = 1 + (l-1) \bmod m$  and column  $j_l = 1 + \lfloor (l-1)/m \rfloor$ . For backtracking we store the  $n$ -trie pointer  $a_{i_l, j_l}$  as before, together with an index  $x_l$  into the compressed  $m$ -trie.

This method was suggested by Bernard Gattegno in 1996 (unpublished). It finds all  $5 \times 6$  word rectangles in just 400 gigamems; and its running time for "transposed"  $6 \times 5$  rectangles turns out to be slightly less (380 gigamems). Notice that only one mem is needed to access each (character, link) pair in the compressed trie.

**29.** Leave out **face** and (of course) **dada**; the remaining eleven are fine.

**30.** Keep tables  $p_i, p'_{ij}, p''_{ijk}, s_i, s'_{ij}, s''_{ijk}$ , for  $0 \leq i, j, k < m$ , each capable of storing a ternary digit. Also keep a table  $x_0, x_1, \dots$  of tentatively accepted words. Begin with  $g \leftarrow 0$ . Then for each input  $w_j = abcd$ , where  $0 \leq a, b, c, d < m$ , set  $x_g \leftarrow abcd$  and also do the following: Set  $p_a \leftarrow p_a + 1, p'_{ab} \leftarrow p'_{ab} + 1, p''_{abc} \leftarrow p''_{abc} + 1, s_d \leftarrow s_d + 1, s'_{cd} \leftarrow s'_{cd} + 1, s''_{bcd} \leftarrow s''_{bcd} + 1$ , where  $x + y = \min(2, x + y)$  denotes saturating ternary addition. Then if  $s_a p'_{b'c'd'} + s'_{a'b'} p'_{c'd'} + s''_{a'b'c'} p_{d'} = 0$  for all  $x_k = a'b'c'd'$ , where  $0 \leq k \leq g$ , set  $g \leftarrow g + 1$ . Otherwise reject  $w_j$  and set  $p_a \leftarrow p_a - 1, p'_{ab} \leftarrow p'_{ab} - 1, p''_{abc} \leftarrow p''_{abc} - 1, s_d \leftarrow s_d - 1, s'_{cd} \leftarrow s'_{cd} - 1, s''_{bcd} \leftarrow s''_{bcd} - 1$ .

**31.** (a) The word  $bc$  appears in message  $abcd$  if and only if  $a \rightarrow b, b \rightarrow c$ , and  $c \rightarrow d$ .

(b) For  $0 \leq k < r$ , put vertex  $v$  into class  $k$  if the longest path from  $v$  has length  $k$ . Given any such partition, we can include all arcs from class  $k$  to class  $j < k$  without increasing the path lengths. So it's a question of finding the maximum of  $\sum_{0 \leq j < k < r} p_j p_k$  subject to  $p_0 + p_1 + \dots + p_{r-1} = m$ . The values  $p_j = \lfloor (m+j)/r \rfloor$  achieve this (see exercise 7.2.1.4–68(a)). When  $r = 3$  the maximum simplifies to  $\lfloor m^2/3 \rfloor$ .

**32.** (a) The factors of the period, 15 926 535 89 79 323 8314, begin at the respective boundary points 3, 5, 8, 11, 13, 15, 18 (and then  $3 + 19 = 22$ , etc.). Thus round 1 retains boundaries 5, 8, and 15. The second-round substrings  $y_0 = 926, y_1 = 5358979, y_2 = 323831415$  have different lengths, so lexicographic comparison is unnecessary; the answer is  $y_2 y_0 y_1 = x_{15} \dots x_{33}$ .

(b) Each substring consists of at least three substrings of the previous round.

(c) Let  $a_0 = 0$ ,  $b_0 = 1$ ,  $a_{e+1} = a_e a_e b_e$ ,  $b_{e+1} = a_e b_e b_e$ ; use  $a_e$  or  $b_e$  when  $n = 3^e$ .

(d) We use an auxiliary subroutine 'less( $i$ )', which returns  $[y_{i-1} < y_i]$ , given  $i > 0$ : If  $b_i - b_{i-1} \neq b_{i+1} - b_i$ , return  $[b_i - b_{i-1} < b_{i+1} - b_i]$ . Otherwise, for  $j = 0, 1, \dots$ , while  $b_i + j < b_{i+1}$ , if  $x_{b_{i-1}+j} \neq x_{b_i+j}$  return  $[x_{b_{i-1}+j} < x_{b_i+j}]$ . Otherwise return 0.

The tricky part of the algorithm is to discard initial factors that aren't periodic. The secret is to let  $i_0$  be the smallest index such that  $y_{i-3} \geq y_{i-2} < y_{i-1}$ ; then we can be sure that a factor begins with  $y_i$ .

**O1.** [Initialize.] Set  $x_j \leftarrow x_{j-n}$  for  $n \leq j < 2n$ ,  $b_j \leftarrow j$  for  $0 \leq j < 2n$ , and  $t \leftarrow n$ .

**O2.** [Begin a round.] Set  $t' \leftarrow 0$ . Find the smallest  $i > 0$  such that  $\text{less}(i) = 0$ . Then find the smallest  $j \geq i + 2$  such that  $\text{less}(j - 1) = 1$  and  $j \leq t + 2$ . (If no such  $j$  exists, report an error: The input  $x$  was equal to one of its cyclic shifts.) Set  $i \leftarrow i_0 \leftarrow j \bmod t$ . (Now a dip of the period begins at  $i_0$ .)

**O3.** [Find the next factor.] Find the smallest  $j \geq i + 2$  such that  $\text{less}(j - 1) = 1$ . If  $j - i$  is even, go to O5.

**O4.** [Retain a boundary.] If  $j < t$ , set  $b'_{t'} \leftarrow b_j$ ; otherwise set  $b'_k \leftarrow b'_{k-1}$  for  $t' \geq k > 0$  and  $b'_0 \leftarrow b_{j-t}$ . Finally set  $t' \leftarrow t' + 1$ .

**O5.** [Done with round?] If  $j < i_0 + t$ , set  $i \leftarrow j$  and return to O3. Otherwise, if  $t' = 1$ , terminate;  $\sigma x$  begins at item  $x_{i_0}$ . Otherwise set  $t \leftarrow t'$ ,  $b_k \leftarrow b'_k$  for  $0 \leq k < t$ , and  $b_k \leftarrow b_{k-t} + n$  for  $k \geq t$  while  $b_{k-t} < n$ . Return to O2. ■

(e) Say that a "superdip" is a dip of odd length followed by zero or more dips of even length. Any infinite sequence  $y$  that begins with an odd-length dip has a unique factorization into superdips. Those superdips can, in turn, be regarded as atomic elements of a higher-level string that can be factored into dips. The result  $\sigma x$  of Algorithm O is an infinite periodic sequence that allows repeated factorization into infinite periodic sequences of superdips at higher and higher levels, until becoming constant.

Notice that the first dip of  $\sigma x$  ends at position  $i_0$  in the algorithm, because its length isn't 2. Therefore we can prove the commafree property by observing that, if codeword  $\sigma x''$  appears within the concatenation  $\sigma x \sigma x'$  of two codewords, its superdip factors are also superdip factors of those codewords. This yields a contradiction if any of  $\sigma x$ ,  $\sigma x'$ , or  $\sigma x''$  is a superdip. Otherwise the same observation applies to the superdip factors at the next level. [Eastman's original algorithm was essentially the same, but presented in a more complicated way; see *IEEE Trans. IT-11* (1965), 263-267. R. A. Scholtz subsequently discovered an interesting and totally different way to define the set of codewords produced by Algorithm O, in *IEEE Trans. IT-15* (1969), 300-306.]

**33.** Let  $f_k(m)$  be the number of dips of length  $k$  for which  $m > z_1$  and  $z_k < m$ . The number of such sequences with  $z_2 = j$  is  $(m - j - 1) \binom{m-j+k-3}{k-2} = (k-1) \binom{m-j+k-3}{k-1}$ ; summing for  $0 \leq j < m$  gives  $f_k(m) = (k-1) \binom{m+k-2}{k}$ . Thus  $F_m(z) = \sum_{k=0}^{\infty} f_k(m) z^k = (mz-1)/(1-z)^m$ . (The fact that  $f_0(m) = -1$  in these formulas turns out to be useful!)

Algorithm O finishes in one round if and only if some cyclic shift of  $x$  is a superdip. The number of aperiodic  $x$  that finish in one round is therefore  $n[z^n] G_m(z)$ , where

$$G_m(z) = \frac{F_m(-z) - F_m(z)}{F_m(-z) + F_m(z)} = \frac{(1+mz)(1-z)^m - (1-mz)(1+z)^m}{(1+mz)(1-z)^m + (1-mz)(1+z)^m}.$$

To get the stated probability, divide by  $\sum_{d|n} \mu(d) m^{n/d}$ , the number of aperiodic  $x$ . (See Eq. 7.2.1.1-(60). For  $n = 3, 5, 7, 9$  these probabilities are 1, 1, 1, and  $1 - 3/\binom{m^3-1}{3}$ .)

**34.** If so, it couldn't have 0011, 0110, 1100, or 1001.

superdip  
Eastman  
Scholtz  
aperiodic

**35.** That section considered such representations of stacks and queues, but not of unordered sets, because large blocks of sequential memory were either nonexistent or ultra-expensive in olden days. Linked lists were the only decent option for families of variable-size sets, because they could more readily fit in a limited high-speed memory.

stacks  
queues  
memory constraints, historic  
unordered sets  
Linked lists  
deletion

**36.** (a) The blue word  $x$  with  $\alpha = d$  (namely 1101) appears in its P2 list at location 5e.  
(b) The P3 list for words of the form 010\* is empty. (Both 0100 and 0101 are red.)

**37.** (a) The S2 list of 0010 has become closed (hence 0110 and 1110 are hidden).  
(b) Word 1101 moved to the former position of 1001 in its S1 list, when 1001 became red. (Previously 1011 had moved to the former position of 0001.)

**38.** In this case, which of course happens rarely, it's safe to set all elements of STAMP to zero and set  $\sigma \leftarrow 1$ . (Do *not* be tempted to save one line of code by setting all STAMP elements to  $-1$  and leaving  $\sigma = 0$ . That might fail when  $\sigma$  reaches the value  $-1$ !)

**39.** (a) Set  $r \leftarrow m + 1$ . Then for  $k \leftarrow 0, 1, \dots, f - 1$ , set  $t \leftarrow \text{FREE}[k]$ ,  $j \leftarrow \text{MEM}[\text{CLOFF} + 4t + m^4] - (\text{CLOFF} + 4t)$ , and if  $j < r$  set  $r \leftarrow j$ ,  $c \leftarrow t$ ; break out of the loop if  $r = 0$ .

(b) If  $r > 0$  set  $x \leftarrow \text{MEM}[\text{CLOFF} + 4cl(\text{ALF}[x])]$ .

(c) If  $r > 1$  set  $q \leftarrow 0$ ,  $p' \leftarrow \text{MEM}[\text{PP}]$ , and  $p \leftarrow \text{POISON}$ . While  $p < p'$  do the following steps: Set  $y \leftarrow \text{MEM}[p]$ ,  $z \leftarrow \text{MEM}[p + 1]$ ,  $y' \leftarrow \text{MEM}[y + m^4]$ , and  $z' \leftarrow \text{MEM}[z + m^4]$ . (Here  $y$  and  $z$  point to the heads of prefix or suffix lists;  $y'$  and  $z'$  point to the tails.) If  $y = y'$  or  $z = z'$ , delete entry  $p$  from the poison list; this means, as in (18), to set  $p' \leftarrow p' - 2$ , and if  $p \neq p'$  to store( $p, \text{MEM}[p']$ ) and store( $p + 1, \text{MEM}[p' + 1]$ ). Otherwise set  $p \leftarrow p + 2$ ; if  $y' - y \geq z' - z$  and  $y' - y > q$ , set  $q \leftarrow y' - y$  and  $x \leftarrow \text{MEM}[z]$ ; if  $y' - y < z' - z$  and  $z' - z > q$ , set  $q \leftarrow z' - z$  and  $x \leftarrow \text{MEM}[y]$ . Finally, after  $p$  has become equal to  $p'$ , store( $\text{PP}, p'$ ) and set  $c \leftarrow cl(\text{ALF}[x])$ . (Experiments show that this "max kill" strategy for  $r > 1$  slightly outperforms a selection strategy based on  $r$  alone.)

**40.** (a) First there's a routine  $\text{rem}(\alpha, \delta, o)$  that removes an item from a list, following the protocol (21): Set  $p \leftarrow \delta + o$  and  $q \leftarrow \text{MEM}[p + m^4] - 1$ . If  $q \geq p$  (meaning that list  $p$  isn't closed or being killed), store( $p + m^4, q$ ), set  $t \leftarrow \text{MEM}[\alpha + o - m^4]$ ; and if  $t \neq q$  also set  $y \leftarrow \text{MEM}[q]$ , store( $t, y$ ), and store( $\text{ALF}[y] + o - m^4, t$ ).

Now, to redden  $x$  we set  $\alpha \leftarrow \text{ALF}[x]$ , store( $\alpha, \text{RED}$ ); then  $\text{rem}(\alpha, p_1(\alpha), \text{P10FF})$ ,  $\text{rem}(\alpha, p_2(\alpha), \text{P20FF})$ ,  $\dots$ ,  $\text{rem}(\alpha, s_3(\alpha), \text{S30FF})$ , and  $\text{rem}(\alpha, 4cl(\alpha), \text{CLOFF})$ .

(b) A simple routine  $\text{close}(\delta, o)$  closes list  $\delta + o$ : Set  $p \leftarrow \delta + o$  and  $q \leftarrow \text{MEM}[p + m^4]$ ; if  $q \neq p - 1$ , store( $p + m^4, p - 1$ ).

Now, to green  $x$  we set  $\alpha \leftarrow \text{ALF}[x]$ , store( $\alpha, \text{GREEN}$ ); then  $\text{close}(p_1(\alpha), \text{P10FF})$ ,  $\text{close}(p_2(\alpha), \text{P20FF})$ ,  $\dots$ ,  $\text{close}(s_3(\alpha), \text{S30FF})$ , and  $\text{close}(4cl(\alpha), \text{CLOFF})$ . Finally, for  $p \leq r < q$  (using the  $p$  and  $q$  that were just set within 'close'), if  $\text{MEM}[r] \neq x$  redden  $\text{MEM}[r]$ .

(c) First set  $p' \leftarrow \text{MEM}[\text{PP}] + 6$ , and store( $p' - 6, p_1(\alpha) + \text{S10FF}$ ), store( $p' - 5, s_3(\alpha) + \text{P30FF}$ ), store( $p' - 4, p_2(\alpha) + \text{S20FF}$ ), store( $p' - 3, s_2(\alpha) + \text{P20FF}$ ), store( $p' - 2, p_3(\alpha) + \text{S30FF}$ ), store( $p' - 1, s_1(\alpha) + \text{P10FF}$ ); this adds the three poison items (27).

Then set  $p \leftarrow \text{POISON}$  and do the following while  $p < p'$ : Set  $y, z, y', z'$  as in answer 39(c), and delete poison entry  $p$  if  $y = y'$  or  $z = z'$ . Otherwise if  $y' < y$  and  $z' < z$ , go to C6 (a poisoned suffix-prefix pair is present). Otherwise if  $y' > y$  and  $z' > z$ , set  $p \leftarrow p + 2$ . Otherwise if  $y' < y$  and  $z' > z$ , store( $z + m^4, z$ ), redden  $\text{MEM}[r]$  for  $z \leq r < z'$ , and delete poison entry  $p$ . Otherwise (namely if  $y' > y$  and  $z' < z$ ), store( $y + m^4, y$ ), redden  $\text{MEM}[r]$  for  $y \leq r < y'$ , and delete poison entry  $p$ .

Finally, after  $p$  has become equal to  $p'$ , store( $\text{PP}, p'$ ).

**42.** Exercise 32 exhibits such codes explicitly for all odd  $n$ . The earliest papers on the subject gave solutions for  $n = 2, 4, 6, 8$ . Yoji Niho subsequently found a code for  $n = 10$  but was unable to resolve the case  $n = 12$  [*IEEE Trans. IT-19* (1973), 580–581].

This problem can readily be encoded in CNF and given to a SAT solver. The case  $n = 10$  involves 990 variables and 8.6 million clauses, and is solved by Algorithm 7.2.2.2C in 10.5 gigamems. The case  $n = 12$  involves 4020 variables and 175 million clauses. After being split into seven independent subproblems (by appending mutually exclusive unit clauses), it was proved *unsatisfiable* by that algorithm after about 86 teramems of computation.

So the answer is “No.” The maximum-size code for  $n = 12$  remains unknown.

**44.** (a) There are 28 commafree binary codes of size 3 and length 4; Algorithm C produces half of them, because it assumes that cycle class [0001] is represented by 0001 or 0010. They form eight equivalence classes, two of which are symmetric under the operation of complementation-and-reflection; representatives are {0001, 0011, 0111} and {0010, 0011, 1011}. The other six are represented by {0001, 0110, 0111 or 1110}, {0001, 1001, 1011 or 1101}, {0001, 1100, 1101}, {0010, 0011, 1101}.

(b) Algorithm C produces half of the 144 solutions, which form twelve equivalence classes. Eight are represented by {0001, 0002, 1001, 1002, 2201, 2001, 2002, 2011, 2012, 2102, 2112, 2122 or 2212} and ({0102, 1011, 1012} or {1020, 1101, 2101}) and ({1202, 2202, 2111} or {2021, 2022, 1112}); four are represented by {0001, 0020, 0021, 0022, 1001, 1020, 1021, 1022, 1121 or 1211, 1201, 1202, 1221, 2001, 2201, 2202} and ({1011, 1012, 2221} or {1101, 2101, 1222}).

(c) Algorithm C yields half of the 2304 solutions, which form 48 equivalence classes. Twelve classes have unique representatives that omit cycle classes [0123], [0103], [1213], one such being the code {0010, 0020, 0030, 0110, 0112, 0113, 0120, 0121, 0122, 0130, 0131, 0132, 0133, 0210, 0212, 0213, 0220, 0222, 0230, 0310, 0312, 0313, 0320, 0322, 0330, 0332, 0333, 1110, 1112, 1113, 2010, 2030, 2110, 2112, 2113, 2210, 2212, 2213, 2230, 2310, 2312, 2313, 2320, 2322, 2330, 2332, 2333, 3110, 3112, 3113, 3210, 3212, 3213, 3230, 3310, 3312, 3313}. The others each have two representatives that omit classes [0123], [0103], [0121], one such being the code {0001, 0002, 0003, 0201, 0203, 1001, 1002, 1003, 1011, 1013, 1021, 1022, 1023, 1031, 1032, 1033, 1201, 1203, 1211, 1213, 1221, 1223, 1231, 1232, 1233, 1311, 1321, 1323, 1331, 2001, 2002, 2003, 2021, 2022, 2023, 2201, 2203, 2221, 2223, 3001, 3002, 3003, 3011, 3013, 3021, 3022, 3023, 3031, 3032, 3033, 3201, 3203, 3221, 3223, 3321, 3323, 3331} and its isomorphic image under reflection and (01)(23).

**45.** (The maximum size of such a code is currently unknown. Algorithm C isn’t fast enough to solve this problem on a single computer, but a sufficiently large cluster of machines and/or an improved algorithm should be able to discover the answer. The case  $m = 3$  and  $n = 6$  is also currently unsolved; a SAT solver shows quickly that a full set of  $(3^6 - 3^3 - 3^2 + 3^1)/6 = 116$  codewords cannot be achieved.)

**47.** The 3-bit sequences 101, 111, 110 were rejected before seeing 000. In general, to make a uniformly random choice from  $q$  possibilities, the text suggests looking at the next  $t = \lceil \lg q \rceil$  bits  $b_1 \dots b_t$ . If  $(b_1 \dots b_t)_2 < q$ , we use choice  $(b_1 \dots b_t)_2 + 1$ ; otherwise we reject  $b_1 \dots b_t$  and try again. [This simple method is optimum when  $q \leq 4$ , and the best possible running time for other values of  $q$  uses more than half as many bits. But a better scheme is available for  $q = 5$ , using only  $3\frac{1}{5}$  bits per choice instead of  $4\frac{4}{5}$ ; and for  $q = 6$ , one random bit reduces to the case  $q = 3$ . See D. E. Knuth and A. C. Yao, *Algorithms and Complexity*, edited by J. F. Traub (Academic Press, 1976), 357–428, §2.]

Niho  
CNF  
SAT solver  
unit clauses  
uniformly random  
reject  
Knuth  
Yao  
complexity of calculation  
computational complexity  
Traub

**48.** It's the number of nodes on level  $l + 1$  (depth  $l$ ) of the search tree. (Hence we can estimate the profile. Notice that  $D = D_1 \dots D_{l-1}$  in step E2 of Algorithm E.)

**49.**  $Z_0 = C()$ ,  $Z_{l+1} = c() + D_1 c(X_1) + D_1 D_2 c(X_1 X_2) + \dots + D_1 \dots D_l c(X_1 \dots X_l) + D_1 \dots D_{l+1} C(X_1 \dots X_{l+1})$ .

**50.** (a) True: The generating function is  $z(z+1)\dots(z+n-1)/n!$ ; see Eq. 1.2.10-(g).

(b) For instance, suppose  $Y_1 Y_2 \dots Y_l = 1457$  and  $n = 9$ . Alice's probability is  $\frac{1}{1} \frac{1}{2} \frac{2}{3} \frac{1}{4} \frac{1}{5} \frac{5}{6} \frac{1}{7} \frac{7}{8} \frac{8}{9} = \frac{1}{3} \frac{1}{4} \frac{1}{6} \frac{1}{9}$ . Elmo obtains  $X_1 X_2 \dots X_l = 7541$  with probability  $\frac{1}{9} \frac{1}{6} \frac{1}{4} \frac{1}{3}$ .

(c) The upper tail inequality (see exercise 1.2.10-22 with  $\mu = H_n$ ) tells us that  $\Pr(l \geq (\ln n)(\ln \ln n)) \leq \exp(-(\ln n)(\ln \ln n)(\ln \ln \ln n) + O(\ln n)(\ln \ln n))$ .

(d) If  $k \leq n/3$  we have  $\sum_{j=0}^k \binom{n}{j} \leq 2 \binom{n}{k}$ . By exercise 1.2.6-67, the number of nodes on the first  $(\ln n)(\ln \ln n)$  levels is therefore at most  $2(ne/((\ln n)(\ln \ln n)))^{(\ln n)(\ln \ln n)}$ .

**51.** The key idea is to introduce recursive formulas analogous to (29):

$$\begin{aligned} m(x_1 \dots x_l) &= c(x_1 \dots x_l) + \min(m(x_1 \dots x_l x_{l+1}^{(1)} d), \dots, m(x_1 \dots x_l x_{l+1}^{(d)} d)); \\ M(x_1 \dots x_l) &= c(x_1 \dots x_l) + \max(M(x_1 \dots x_l x_{l+1}^{(1)} d), \dots, M(x_1 \dots x_l x_{l+1}^{(d)} d)); \\ \widehat{C}(x_1 \dots x_l) &= c(x_1 \dots x_l)^2 + \sum_{i=1}^d (\widehat{C}(x_1 \dots x_l x_{l+1}^{(i)} d) + 2c(x_1 \dots x_l)C(x_1 \dots x_l x_{l+1}^{(i)})). \end{aligned}$$

They can be computed via auxiliary arrays MIN, MAX, KIDS, COST, and CHAT as follows:

At the beginning of step B2, set  $\text{MIN}[l] \leftarrow \infty$ ,  $\text{MAX}[l] \leftarrow \text{KIDS}[l] \leftarrow \text{COST}[l] \leftarrow \text{CHAT}[l] \leftarrow 0$ . Set  $\text{KIDS}[l] \leftarrow \text{KIDS}[l] + 1$  just before  $l \leftarrow l + 1$  in step B3.

At the beginning of step B5, set  $m \leftarrow c(x_1 \dots x_{l-1}) + \text{KIDS}[l] \times \text{MIN}[l]$ ,  $M \leftarrow c(x_1 \dots x_{l-1}) + \text{KIDS}[l] \times \text{MAX}[l]$ ,  $C \leftarrow c(x_1 \dots x_{l-1}) + \text{COST}[l]$ ,  $\widehat{C} \leftarrow c(x_1 \dots x_{l-1})^2 + \text{KIDS}[l] \times \text{CHAT}[l] + 2 \times \text{COST}[l]$ . Then, after  $l \leftarrow l - 1$  is positive, set  $\text{MIN}[l] \leftarrow \min(m, \text{MIN}[l])$ ,  $\text{MAX}[l] \leftarrow \max(M, \text{MAX}[l])$ ,  $\text{COST}[l] \leftarrow \text{COST}[l] + C$ ,  $\text{CHAT}[l] \leftarrow \text{CHAT}[l] + \widehat{C}$ . But when  $l$  reaches zero in step B5, return the values  $m, M, C, \widehat{C} - C^2$ .

**52.** Let  $p(i) = p_{x_1 \dots x_{l-1}}(y_i)$ , and simply set  $D \leftarrow D/p(I)$  instead of  $D \leftarrow Dd$ . Then node  $x_1 \dots x_l$  is reached with probability  $\Pi(x_1 \dots x_l) = p(x_1)p_{x_1}(x_2) \dots p_{x_1 \dots x_l}(x_l)$ , and  $c(x_1 \dots x_l)$  has weight  $1/\Pi(x_1 \dots x_l)$  in  $S$ ; the proof of Theorem E goes through as before. Notice that  $p(I)$  is the *a posteriori* probability of having taken branch  $I$ .

(The formulas of answer 51 should now use ' $p(i)$ ' instead of ' $d$ '; and that algorithm should be modified appropriately, no longer needing the KIDS array.)

**53.** Let  $p_{x_1 \dots x_{l-1}}(y_i) = C(x_1 \dots x_{l-1} y_i) / (C(x_1 \dots x_{l-1}) - c(x_1 \dots x_{l-1}))$ . (Of course we generally need to know the cost of the tree before we know the exact values of these ideal probabilities, so we cannot achieve zero variance in practice. But the form of this solution shows what kinds of bias are likely to reduce the variance.)

**55.** The effects of lookahead, dynamic ordering, and reversible memory are all captured easily by a well-designed cost function at each node. But there's a fundamental difference in step C2, because different codeword classes can be selected for branching at the same node (that is, with the same ancestors  $x_1 \dots x_{l-1}$ ) after C5 has undone the effects of a prior choice. The level  $l$  never surpasses  $L + 1$ , but in fact the search tree involves hidden levels of branching that are implicitly combined into single nodes.

Thus it's best to view Algorithm C's search tree as a sequence of *binary* branches: Should  $x$  be one of the codewords or not? (At least this is true when the "max kill" strategy of answer 39 has selected the branching variable  $x$ . But if  $r > 1$  and the poison list is empty, an  $r$ -way branch is reasonable (or an  $(r + 1)$ -way branch when the slack is positive), because  $r$  will be reduced by 1 and the same class  $c$  will be chosen after  $x$  has been explored.)

profile  
generating function  
tail inequality  
cumulative binomial distribution  
recursive formulas  
a posteriori  
cost function  
search tree  
poison list  
slack



If  $x$  has been selected because it kills many other potential codewords, we probably should bias the branch probability as in exercise 52, giving smaller weight to the “yes” branch because the branch that includes  $x$  is less likely to lead to a large subtree.

**57.** Let  $p_k = 1/D^{(k)}$  be the probability that Algorithm E terminates at the  $k$ th leaf. Then  $\sum_{k=1}^M (1/M) \lg(1/(Mp_k))$  is the Kullback–Leibler divergence  $D(q||p)$ , where  $q$  is the uniform distribution (see exercise MPR–121). Hence  $\frac{1}{M} \sum_{k=1}^M \lg D^{(k)} \geq \lg M$ . (The result of this exercise is essentially true in *any* probability distribution.)

**58.** Let  $\infty$  be any convenient value  $\geq n$ . When vertex  $v$  becomes part of the path we will perform a two-phase algorithm. The first phase identifies all “tarnished” vertices, whose DIST must change; these are the vertices  $u$  from which every path to  $t$  passes through  $v$ . It also forms a queue of “resource” vertices, which are untarnished but adjacent to tarnished ones. The second phase updates the DISTs of all tarnished vertices that are still connected to  $t$ . Each vertex has LINK and STAMP fields in addition to DIST.

For the first phase, set  $d \leftarrow \text{DIST}(v)$ ,  $\text{DIST}(v) \leftarrow \infty + 1$ ,  $R \leftarrow \Lambda$ ,  $T \leftarrow v$ ,  $\text{LINK}(v) \leftarrow \Lambda$ , then do the following while  $T \neq \Lambda$ : (\*) Set  $u \leftarrow T$ ,  $T \leftarrow S \leftarrow \Lambda$ . For each  $w \text{ --- } u$ , if  $\text{DIST}(w) < d$  do nothing (this happens only when  $u = v$ ); if  $\text{DIST}(w) \geq \infty$  do nothing ( $w$  is gone or already known to be tarnished); if  $\text{DIST}(w) = d$ , make  $w$  a resource (see below); otherwise  $\text{DIST}(w) = d + 1$ . If  $w$  has no neighbor at distance  $d$ ,  $w$  is tarnished: Set  $\text{LINK}(w) \leftarrow T$ ,  $\text{DIST}(w) \leftarrow \infty$ ,  $T \leftarrow w$ . Otherwise make  $w$  a resource (see below). Then set  $u \leftarrow \text{LINK}(u)$ , and return to (\*) if  $u \neq \Lambda$ .

The queue of resources will start at  $R$ . We will stamp each resource with  $v$  so that nothing is added twice to that queue. To make  $w$  a resource when  $\text{DIST}(w) = d$ , do the following (unless  $u = v$  or  $\text{STAMP}(w) = v$ ): Set  $\text{STAMP}(w) \leftarrow v$ ; if  $R = \Lambda$ , set  $R \leftarrow \text{RT} \leftarrow w$ ; otherwise set  $\text{LINK}(\text{RT}) \leftarrow w$  and  $\text{RT} \leftarrow w$ . To make  $w$  a resource when  $\text{DIST}(w) = d + 1$  and  $u \neq v$  and  $\text{STAMP}(w) \neq v$ , put it first on stack  $S$  as follows: Set  $\text{STAMP}(w) \leftarrow v$ ; if  $S = \Lambda$ , set  $S \leftarrow \text{SB} \leftarrow w$ ; otherwise set  $\text{LINK}(w) \leftarrow S$ ,  $S \leftarrow w$ .

Finally, when  $u = \Lambda$ , we append  $S$  to  $R$ : Nothing needs to be done if  $S = \Lambda$ . Otherwise, if  $R = \Lambda$ , set  $R \leftarrow S$  and  $\text{RT} \leftarrow \text{SB}$ ; but if  $R \neq \Lambda$ , set  $\text{LINK}(\text{RT}) \leftarrow S$  and  $\text{RT} \leftarrow \text{SB}$ . (These shenanigans keep the resource queue in order by DIST.)

Phase 2 operates as follows: Nothing needs to be done if  $R = \Lambda$ . Otherwise we set  $\text{LINK}(\text{RT}) \leftarrow \Lambda$ ,  $S \leftarrow \Lambda$ , and do the following while  $R \neq \Lambda$  or  $S \neq \Lambda$ : (i) If  $S = \Lambda$ , set  $d \leftarrow \text{DIST}(R)$ . Otherwise set  $u \leftarrow S$ ,  $d \leftarrow \text{DIST}(u)$ ,  $S \leftarrow \Lambda$ ; while  $u \neq \Lambda$ , update the neighbors of  $u$  and set  $u \leftarrow \text{LINK}(u)$ . (ii) While  $R \neq \Lambda$  and  $\text{DIST}(R) = d$ , set  $u \leftarrow R$ ,  $R \leftarrow \text{LINK}(u)$ , and update the neighbors of  $u$ . In both cases “update the neighbors of  $u$ ” means to look at all  $w \text{ --- } u$ , and if  $\text{DIST}(w) = \infty$  to set  $\text{DIST}(w) \leftarrow d + 1$ ,  $\text{STAMP}(w) \leftarrow v$ ,  $\text{LINK}(w) \leftarrow S$ , and  $S \leftarrow w$ . (It works!)

**59.** (a) Compute the generating function  $g(z)$  (see exercise 7.1.4–209) and then  $g'(1)$ .

(b) Let  $(A, B, C)$  denote paths that touch (center, NE corner, SW corner). Recursively compute eight counts  $(c_0, \dots, c_7)$  at each node, where  $c_j$  counts paths  $\pi$  with  $j = 4[\pi \in A] + 2[\pi \in B] + [\pi \in C]$ . At the sink node  $\square$  we have  $c_0 = 1$ ,  $c_1 = \dots = c_7 = 0$ . Other nodes have the form  $x = (\bar{e}^? x_l: x_h)$  where  $e$  is an edge. Two edges go across the center and affect  $A$ ; three edges affect each of  $B$  and  $C$ . Say that those edges have types 4, 2, 1, respectively; other edges have type 0. Suppose the counts for  $x_l$  and  $x_h$  are  $(c'_0, \dots, c'_7)$  and  $(c''_0, \dots, c''_7)$ , and  $e$  has type  $t$ . Then count  $c_j$  for node  $x$  is  $c'_j + [t=0]c''_j + [t \& j \neq 0](c''_j + c''_{j-t})$ .

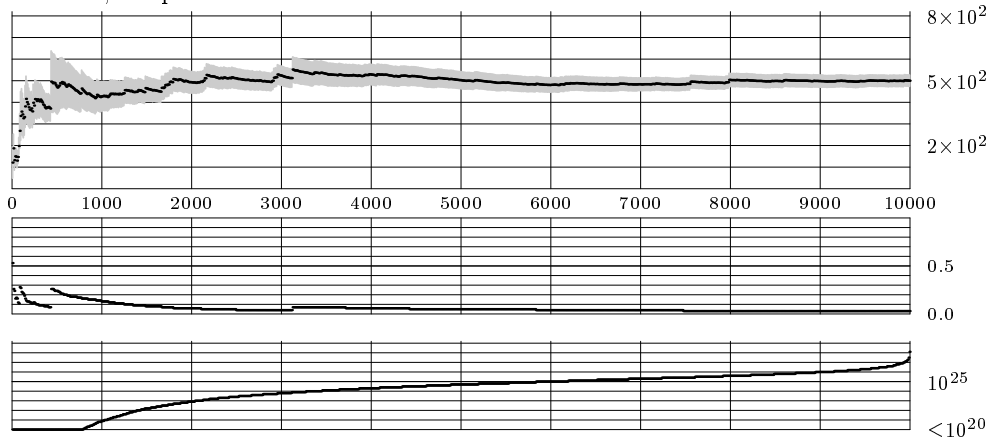
(This procedure yields the following exact “Venn diagram” set counts at the root:  $c_0 = |\bar{A} \cap \bar{B} \cap \bar{C}| = 7653685384889019648091604$ ;  $c_1 = c_2 = |\bar{A} \cap \bar{B} \cap C| = |\bar{A} \cap B \cap \bar{C}| = 7755019053779199171839134$ ;  $c_3 = |\bar{A} \cap B \cap C| = 7857706970503366819944024$ ;

bias  
Kullback  
Leibler  
divergence  
generating function  
Venn diagram

$c_4 = |A \cap \overline{B} \cap \overline{C}| = 4888524166534573765995071$ ;  $c_5 = c_6 = |A \cap \overline{B} \cap C| = |A \cap B \cap \overline{C}| = 4949318991771252110605148$ ;  $c_7 = |A \cap B \cap C| = 5010950157283718807987280$ .)

recurrence  
Cook  
Kleber  
Mahler  
unique solution  
Gridgeman

60. Yes, the paths are less chaotic and the estimates are better:



61. (a) Let  $x_k$  be the number of nodes at distance  $k - 1$  from the root.  
 (b) Let  $Q_n^{(m)} = P_n^{(1)} + \dots + P_n^{(m)}$ . Then we have the joint recurrence  $P_1^{(m)} = 1$ ,  $P_{n+1}^{(m)} = Q_n^{(2m)}$ ; in particular,  $Q_1^{(m)} = m$ . And for  $n \geq 2$ , we have  $Q_n^{(m)} = \sum_{k=1}^n a_{nk} \binom{m}{k}$  for certain constants  $a_{nk}$  that can be computed as follows: Set  $t_k \leftarrow P_n^{(k)}$  for  $1 \leq k \leq n$ . Then for  $k = 2, \dots, n$  set  $t_n \leftarrow t_n - t_{n-1}, \dots, t_k \leftarrow t_k - t_{k-1}$ . Finally  $a_{nk} \leftarrow t_k$  for  $1 \leq k \leq n$ . For example,  $a_{21} = a_{22} = 2$ ;  $a_{31} = 6, a_{32} = 14, a_{33} = 8$ . The numbers  $P_n^{(m)}$  have  $O(n^2 + n \log m)$  bits, so this method needs  $O(n^5)$  bit operations to compute  $P_n$ .  
 (c)  $P_n^{(m)}$  corresponds to random paths with  $X_1 = m, D_k = 2X_k, X_{k+1} = \lceil 2U_k X_k \rceil$ , where each  $U_k$  is an independent uniform deviate. Therefore  $P_n^{(m)} = E(D_1 \dots D_{n-1})$  is the number of nodes on level  $n$  of an infinite tree. We have  $X_{k+1} \geq 2^k U_1 \dots U_k m$ , by induction; hence  $P_n^{(m)} \geq E(2^{\binom{n}{2}} U_1^{n-2} U_2^{n-3} \dots U_{n-2}^1 m^{n-1}) = 2^{\binom{n}{2}} m^{n-1} / (n-1)!$ .

[M. Cook and M. Kleber have discussed similar sequences in *Electronic Journal of Combinatorics* 7 (2000), #R44. See also K. Mahler's asymptotic formula for binary partitions, in *J. London Math. Society* 15 (1940), 115–123, which shows that  $\lg P_n = \binom{n}{2} - \lg(n-1)! + \binom{\lg n}{2} + O(1)$ .]

66. Random trials indicate that the expected number of 2-regular graphs is  $\approx 3.115$ , and that the number of disjoint pairs is (0, 1, ..., 9, and  $\geq 10$ ) approximately (74.4, 4.3, 8.7, 1.3, 6.2, 0.2, 1.5, 0.1, 2.0, 0.0, and 12.2) percent of the time. If the cubes are restricted to cases where each color occurs at least five times, these numbers change to  $\approx 4.89$  and (37.3, 6.6, 17.5, 4.1, 16.3, 0.9, 5.3, 0.3, 6.7, 0.2, 5.0).

However, the concept of “unique solution” is tricky, because a 2-regular graph with  $k$  cycles yields  $2^k$  ways to position the cubes. Let's say that a set of cubes has a *strongly unique* solution if (i) it has a unique disjoint pair of 2-regular graphs, and furthermore (ii) both elements of that pair are  $n$ -cycles. Such sets occur with probability only about 0.3% in the first case, and 0.4% in the second.

[N. T. Gridgeman, in *Mathematics Magazine* 44 (1971), 243–252, showed that puzzles with four cubes and four colors have exactly 434 “types” of solutions.]

67. It's easy to find such examples at random, as in the second part of the previous answer, since strongly unique sets occur about 0.5% of the time (and weakly unique

sets occur with probability  $\approx 8.4\%$ ). For example, the pairs of opposite faces might be (12, 13, 34), (02, 03, 14), (01, 14, 24), (04, 13, 23), (01, 12, 34).

(Incidentally, if we require each color to occur exactly *six* times, every set of cubes that has at least one solution will have at least *three* solutions, because the “hidden” pairs can be chosen in three ways.)

**70.** Each of these cubes can be placed in 16 different ways that contribute legitimate letters to all four of the visible words. (A cube whose faces contain only letters in  $\{C, H, I, N, O, U, X, Z\}$  can be placed in 24 ways. A cube with a pattern like  $\begin{array}{|c|c|c|} \hline \text{B} & \text{D} & \text{D} \\ \hline \end{array}$  cannot be placed at all.) We can restrict the first cube to just two placements; thus there are  $2 \cdot 16 \cdot 16 \cdot 16 = 131072$  ways to place those cubes without changing their order. Of these, only 6144 are “compatible,” in the sense that no rightside-up-only letter appears together with an upside-down-only letter in the same word.

The 6144 compatible placements can then each be reordered in  $5! = 120$  ways. One of them, whose words before reordering are GRHTI, NCICY, OΘRMN, INNNO, leads to the unique solution. (There’s a partial solution with three words out of four. There also are 39 ways to get two valid words, including one that has UNTIL adjacent to HOURS, and several with SYRUP opposite ECHOS.)

**73.** Call the rays N, NE, E, SE, S, SW, W, NW; call the disks 1, 2, 3, 4 from inside to outside. We can keep disk 1 fixed. The sum of rays N, S, E, W must be 48. It is 16 (on disk 1) plus 13 or 10 (on disk 2) plus 8 or 13 (on disk 3) plus 11 or 14. So it is attained either as shown, or after rotating disks 2 and 4 clockwise by  $45^\circ$ . (Or we could rotate any disk by a multiple of  $90^\circ$ , since that keeps the desired sum invariant.)

Next, with optional  $90^\circ$  rotations, we must make the sum of rays N + S equal to 24. In the first solution above it is 9 plus (6 or 7) plus (4 or 4) plus (7 or 4), hence never 24. But in the other solution it’s 9 plus (4 or 6) plus (4 or 4) plus (5 or 9); hence we must rotate disk 2 clockwise by  $90^\circ$ , and possibly also disk 3. However,  $90^\circ$  rotation of disk 3 would make the NE + SW sum equal to 25, so we musn’t move it.

Finally, to get NE’s sum to be 12, via optional rotations by  $180^\circ$ , we have 1 plus (2 or 5) plus (1 or 5) plus (3 or 4); we must shift disks 3 and 4. Hurray: That makes all eight rays correct. Factoring twice has reduced  $8^3$  trials to  $2^3 + 2^3 + 2^3$ .

[See George W. Ernst and Michael M. Goldstein, *JACM* **29** (1982), 1–23. Such puzzles go back to the 1800s; three early examples are illustrated on pages 28 of Slocum and Botermans’s *New Book of Puzzles* (1992). One of them, with six rings and six rays, factors from  $6^5$  trials to  $2^5 + 3^5$ . A five-ray puzzle would have defeated factorization.]

**75.** Call the cards **1525**, **5113**, . . . , **3755**. The key observation is that all 12 sums must be odd, so we can first solve the problem mod 2. For this purpose we may call the cards **1101**, **1111**, . . . , **1111**; only three cards now change under rotation, namely **1101**, **0100**, and **1100** (which are the mod 2 images of **1525**, **4542**, and **7384**).

A second observation is that each solution gives  $6 \times 6 \times 2$  others, by permuting rows and/or columns and/or by rotating all nine cards. Hence we can assume that the upper left card is **0011** (**8473**). Then **0100** (**4542**) must be in the first column, possibly rotated to **0001** (**4245**), to preserve parity in the left two black sums. We can assume that it’s in row 2. In fact, after retreating from 13 mod 2 to 13, we see that it *must* be rotated. Hence the bottom left card must be either **4725**, **7755**, or **3755**.

Similarly we see that **1101** (**1525**) must be in the first row, possibly rotated to **0111** (**2515**); we can put it in column 2. It must be rotated, and the top right card must be **3454** or **3755**. This leaves just six scenarios to consider, and we soon obtain the solution: **8473**, **2515**, **3454**; **4245**, **2547**, **7452**; **7755**, **1351**, **5537**.

invariant  
Ernst  
Goldstein  
Slocum  
Botermans  
Washington Monument Puzzle, see Fool’s Disk  
Rotating Century Puzzle, see Fool’s Disk  
Safe Combination Puzzle, see Fool’s Disk

**90.** Suppose there are  $n$  questions, whose answers each lie in a given set  $S$ . A *student* supplies an answer list  $\alpha = a_1 \dots a_n$ , with each  $a_j \in S$ ; a *grader* supplies a Boolean vector  $\beta = x_1 \dots x_n$ . There is a Boolean function  $f_{js}(\alpha, \beta)$  for each  $j \in \{1, \dots, n\}$  and each  $s \in S$ . A graded answer list  $(\alpha, \beta)$  is *valid* if and only if  $F(\alpha, \beta)$  is true, where

student  
grader  
valid  
dynamic ordering  
propagation algorithm  
search tree  
author

$$F(\alpha, \beta) = F(a_1 \dots a_n, x_1 \dots x_n) = \bigwedge_{j=1}^n \bigwedge_{s \in S} ([a_j = s] \Rightarrow x_j \equiv f_{js}(\alpha, \beta)).$$

The *maximum score* is the largest value of  $x_1 + \dots + x_n$  over all graded answer lists  $(\alpha, \beta)$  that are valid. A *perfect score* is achieved if and only if  $F(\alpha, 1 \dots 1)$  holds.

Thus, in the warmup problem we have  $n = 2$ ,  $S = \{A, B\}$ ;  $f_{1A} = [a_2 = B]$ ;  $f_{1B} = [a_1 = A]$ ;  $f_{2A} = x_1$ ;  $f_{2B} = \bar{x}_2 \oplus [a_1 = A]$ . The four possible answer lists are:

- AA:  $F = (x_1 \equiv [A = B]) \wedge (x_2 \equiv x_1)$
- AB:  $F = (x_1 \equiv [B = B]) \wedge (x_2 \equiv \bar{x}_2 \oplus [A = A])$
- BA:  $F = (x_1 \equiv [B = A]) \wedge (x_2 \equiv x_1)$
- BB:  $F = (x_1 \equiv [B = A]) \wedge (x_2 \equiv \bar{x}_2 \oplus [B = A])$

Thus AA and BA must be graded 00; AB can be graded either 10 or 11; and BB has no valid grading. Only AB can achieve the maximum score, 2; but 2 isn't guaranteed.

In Table 666 we have, for example,  $f_{1C} = [a_2 \neq A] \wedge [a_3 = A]$ ;  $f_{4D} = [a_1 = D] \wedge [a_{15} = D]$ ;  $f_{12A} = [\Sigma_A - 1 = \Sigma_B]$ , where  $\Sigma_s = \sum_{1 \leq j \leq 20} [a_j = s]$ . It's amusing to note that  $f_{14E} = [\{\Sigma_A, \dots, \Sigma_E\} = \{2, 3, 4, 5, 6\}]$ .

The other cases are similar (although often more complicated) Boolean functions — except for 20D and 20E, which are discussed further in exercise 91.

Notice that an answer list that contains both 10E and 17E must be discarded: It can't be graded, because 10E says ' $x_{10} \equiv \bar{x}_{17}$ ' while 17E says ' $x_{17} \equiv x_{10}$ '.

By suitable backtrack programming, we can prove first that no perfect score is possible. Indeed, if we consider the answers in the order (3, 15, 20, 19, 2, 1, 17, 10, 5, 4, 16, 11, 13, 14, 7, 18, 6, 8, 12, 9), many cases can quickly be ruled out. For example, suppose  $a_3 = C$ . Then we must have  $a_1 \neq a_2 \neq \dots \neq a_{16} \neq a_{17} = a_{18} \neq a_{19} \neq a_{20}$ , and early cutoffs are often possible. (We might reach a node where the remaining choices for answers 5, 6, 7, 8, 9 are respectively  $\{C, D\}$ ,  $\{A, C\}$ ,  $\{B, D\}$ ,  $\{A, B, E\}$ ,  $\{B, C, D\}$ , say. Then if answer 8 is forced to be B, answer 7 can only be D; hence answer 6 is also forced to be A. Also answer 9 can no longer be B.) An instructive little propagation algorithm will make such deductions nicely at every node of the search tree. On the other hand, difficult questions like 7, 8, 9, are best *not* handled with complicated mechanisms; it's better just to wait until all twenty answers have been tentatively selected, and to check such hard cases only when the checking is easy and fast. In this way the author's program showed the impossibility of a perfect score by exploring just 52859 nodes, after only 3.4 megamems of computation.

The next task was to try for score 19 by asserting that only  $x_j$  is false. This turned out to be impossible for  $1 \leq j \leq 18$ , based on very little computation whatsoever (especially, of course, when  $j = 6$ ). The hardest case,  $j = 15$ , needed just 56 nodes and fewer than 5 kilomems. But then, ta da, three solutions were found: One for  $j = 19$  (185 kilonodes, 11 megamems) and two for  $j = 20$  (131 kilonodes, 8 megamems), namely

- 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20  
D C E A B E B C E A B E A E D B D A b B (i)
- A E D C A B C D C A C E D B C A D A A c (ii)
- D C E A B A D C D A E D A E D B D B E e (iii)

(The incorrect answers are shown here as lowercase letters. The first two solutions establish the truth of 20B and the falsity of 20E.)

**91.** Now there's only *one* list of answers with score  $\geq 19$ , namely (iii). But that is paradoxical — because it claims 20E is false; hence the maximum score *cannot* be 19!

Paradoxical situations are indeed possible when the global function  $F$  of answer 90 is used recursively within one or more of the local functions  $f_{js}$ . Let's explore a bit of recursive territory by considering the following two-question, two-letter example:

1. (A) Answer 1 is incorrect. (B) Answer 2 is incorrect.
2. (A) Some answers can't be graded consistently. (B) No answers achieve a perfect score.

Here we have  $f_{1A} = \bar{x}_1$ ;  $f_{1B} = \bar{x}_2$ ;  $f_{2A} = \exists a_1 \exists a_2 \forall x_1 \forall x_2 \neg F(a_1 a_2, x_1 x_2)$ ;  $f_{2B} = \forall a_1 \forall a_2 \neg F(a_1 a_2, 11)$ . (Formulas quantified by  $\exists a$  or  $\forall a$  expand into  $|S|$  terms, while  $\exists x$  or  $\forall x$  expand into two; for example,  $\exists a \forall x g(a, x) = (g(A, 0) \wedge g(A, 1)) \vee (g(B, 0) \wedge g(B, 1))$  when  $S = \{A, B\}$ .) Sometimes the expansion is undefined, because it has more than one "fixed point"; but in this case there's no problem because  $f_{2A}$  is true: Answer AA can't be graded, since 1A implies  $x_1 \equiv \bar{x}_1$ . Also  $f_{2B}$  is true, because both BA and BB imply  $x_1 \equiv \bar{x}_2$ . Thus we get the maximum score 1 with either BA or BB and grades 01.

On the other hand the simple one-question, one-letter questionnaire '1. (A) The maximum score is 1' has an *indeterminate* maximum score. For in this case  $f_{1A} = F(A, 1)$ . We find that if  $F(A, 1) = 0$ , only (A, 0) is a valid grading, so the only possible score is 0; similarly, if  $F(A, 1) = 1$ , the only possible score is 1.

OK, suppose that the maximum score for the modified Table 666 is  $m$ . We know that  $m < 19$ ; hence (iii) isn't a valid grading. It follows that 20E is true, which means that *every valid graded list of score  $m$  has  $x_{20}$  false*. And we can conclude that  $m = 18$ , because of the following two solutions (which are the only possibilities with 20C false):

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
B A d A B E D C D A E D A E D E D B E c
A E D C A B C D C A C E D B a C D A A c

```

But wait: If  $m = 18$ , we can score 18 with 20A true and two errors, using (say)

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
D e D A B E D e C A E D A E D B D C C A

```

or 47 other answer lists. This *contradicts*  $m = 18$ , because  $x_{20}$  is true.

End of story? No. This argument has implicitly been predicated on the assumption that 20D is false. What if  $m$  is indeterminate? Then a new solution arises

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
D C E A B E D C E A E B A E D B D A d D

```

of score 19. With (iii) it yields  $m = 19$ ! If  $m$  is determinate, we've shown that  $m$  cannot actually be defined consistently; but if  $m$  is indeterminate, it's definitely 19.

*Question 20 was designed to create difficulties. [:-)]*

— DONALD R. WOODS (2001)

**95.** The 29 words *spark, often, lucky, other, month, ought, names, water, games, offer, lying, opens, magic, brick, lamps, empty, organ, noise, after, raise, drink, draft, backs, among, under, match, earth, roots, topic* yield this: "The success or failure of backtrack often depends on the skill and ingenuity of the programmer. . . . Backtrack programming (as many other types of programming) is somewhat of an art." — Solomon W. Golomb, Leonard D. Baumert.

recursively  
quantified  
fixed point  
indeterminate  
WOODS  
Golomb  
Baumert

That solution can be found interactively, using inspired guesses based on a knowledge of English and its common two-letter and three-letter words. But could a computer that knows common English words discover it without understanding their meanings?

We can formulate that question as follows: Let  $w_1, \dots, w_{29}$  be the unknown words from WORDS(1000), and let  $q_1, \dots, q_{29}$  be the unknown words of the quotation. (By coincidence there happen to be just 29 of each.) We can restrict the  $q$ 's to words that appear, say, 32 times or more in the British National Corpus. That gives respectively (85, 562, 1863, 3199, 4650, 5631, 5417, 4724, 3657, 2448) choices for words of (2, 3,  $\dots$ , 11) letters; in particular, we allow 3199 possibilities for the five-letter words  $q_7, q_{11}, q_{21}, q_{22}$ , because they aren't required to lie in WORDS(1000). Is there a unique combination of words  $w_i$  and  $q_j$  that meets the given anacrostic constraints?

This is a challenging problem, which we shall discuss in later sections. The answer turns out (surprisingly?) to be *no*; in fact, here is the first solution found by the author's machine(!): "The success or failure of backtrack often depends on roe skill and ingenuity at the programmer. . . . Backtrack programming (as lacy offal types of programming) as somewhat al an art." (The OSPD4 includes 'al' as the name of the Indian mulberry tree; the BNC has 'al' 3515 times, mostly in unsuitable contexts, but that corpus is a blunt instrument.) Altogether 720 solutions satisfy the stated constraints; they differ from the "truth" only in words of at most five letters.

Anacrostic puzzles, which are also known by other names such as double-crostics, were invented in 1933 by E. S. Kingsley. See E. S. Spiegelthal, *Proceedings of the Eastern Joint Computer Conference* **18** (1960), 39–56, for an interesting early attempt to solve them—*without* backtracking—on an IBM 704 computer.

**96.** Instead of considering 1000 possibilities for  $\frac{\text{131 132 133 134 135}}{\text{137 139 118 119 46 48}}$ , it suffices to consider the 43 pairs  $xy$  such that  $xyab$  is in WORDS(1000) and  $abc$  is a common three-letter word. (Of these pairs **ab**, **ag**,  $\dots$ , **ve**, only **ar** leads to a solution. And indeed, the 720 solutions factor into three sets of 240, corresponding to choosing **earth**, **harsh**, or **large** as  $\frac{\text{131 132 133 134 135}}{\text{137 139 118 119 46 48}}$ .) Similar reductions, but not so dramatic, occur with respect to  $\frac{\text{131 132 133 134 135}}{\text{137 139 118 119 46 48}}$ , and  $\frac{\text{32 35}}{\text{46 48}}$ .

**100.** The following algorithm uses an integer utility field  $\text{TAG}(u)$  in the representation of each vertex  $u$ , representing the number of times  $u$  has been "tagged." The operations "tag  $u$ " and "untag  $u$ " stand respectively for  $\text{TAG}(u) \leftarrow \text{TAG}(u) + 1$  and  $\text{TAG}(u) \leftarrow \text{TAG}(u) - 1$ . Vertices shown as '⊙' in the 21 examples have a nonzero TAG field, indicating that the algorithm has decided not to include them in this particular  $H$ .

State variables  $v_l$  (a vertex),  $i_l$  (an index), and  $a_l$  (an arc) are used at level  $l$  for  $0 \leq l < n$ . We assume that  $n > 1$ .

- R1.** [Initialize.] Set  $\text{TAG}(u) \leftarrow 0$  for all vertices  $u$ . Then set  $v_0 \leftarrow v$ ,  $i \leftarrow i_0 \leftarrow 0$ ,  $a \leftarrow a_0 \leftarrow \text{ARCS}(v)$ ,  $\text{TAG}(v) \leftarrow 1$ ,  $l \leftarrow 1$ , and go to R4.
- R2.** [Enter level  $l$ .] (At this point  $i = i_{l-1}$ ,  $v = v_i$ , and  $a = a_{l-1}$  is an arc from  $v$  to  $v_{i-1}$ .) If  $l = n$ , visit the solution  $v_0 v_1 \dots v_{n-1}$  and set  $l \leftarrow n - 1$ .
- R3.** [Advance  $a$ .] Set  $a \leftarrow \text{NEXT}(a)$ , the next neighbor of  $v$ .
- R4.** [Done with level?] If  $a \neq \Lambda$ , go to R5. Otherwise if  $i = l - 1$ , go to R6. Otherwise set  $i \leftarrow i + 1$ ,  $v \leftarrow v_i$ ,  $a \leftarrow \text{ARCS}(v)$ .
- R5.** [Try  $a$ .] Set  $u \leftarrow \text{TIP}(a)$  and tag  $u$ . If  $\text{TAG}(u) > 1$ , return to R3. Otherwise set  $i_l \leftarrow i$ ,  $a_l \leftarrow a$ ,  $v_l \leftarrow u$ ,  $l \leftarrow l + 1$ , and go to R2.

two-letter  
three-letter  
British National Corpus  
constraints  
author  
OSPD4  
double-crostics  
Kingsley  
Spiegelthal  
IBM 704  
utility field  
tagged  
Λ the null pointer

**R6.** [Backtrack.] Set  $l \leftarrow l - 1$ , and stop if  $l = 0$ . Otherwise set  $i \leftarrow i_l$ . Untag all neighbors of  $v_k$ , for  $l \geq k > i$ . Then set  $a \leftarrow \text{NEXT}(a_l)$ ; while  $a \neq \Lambda$ , untag  $\text{TIP}(a)$  and set  $a \leftarrow \text{NEXT}(a)$ . Finally set  $a \leftarrow a_l$  and return to R3. ■

backtracking, variant structure  
 $n$ -omino placement

This instructive algorithm differs subtly from the conventional structure of Algorithm B. Notice in particular that  $\text{TIP}(a_l)$  is not untagged in step R6; that vertex won't be untagged and chosen again until some previous decision has been reconsidered.

**101.** Let  $G$  have  $N$  vertices. For  $1 \leq k \leq N$ , perform Algorithm R on the  $k$ th vertex  $v$  of  $G$ , except that step R1 should tag the first  $k - 1$  vertices so that they are excluded. (A tricky shortcut can be used: If we untag all neighbors of  $v = v_0$  after Algorithm R stops, the net effect will be to tag only  $v$ .)

The  $n$ -omino placement counts 1, 4, 22, 113, 571, 2816, 13616, 64678, 302574 are computed almost instantly, for small  $n$ . (Larger  $n$  are discussed in Section 7.2.3.)

- 102.** (a) All but the 13th and 18th, which require an upward or leftward step.  
 (b) True. If  $u \in H$  and  $u \neq v$ , let  $p_u$  be any node of  $H$  that's one step closer to  $v$ .  
 (c) Again true: The oriented spanning trees are also ordinary spanning trees.  
 (d) The same algorithm works, except that step R4 must return to itself after setting  $a \leftarrow \text{ARCS}(v)$ . (We can no longer be sure that  $\text{ARCS}(v) \neq \Lambda$ .)

**999.** ...

# INDEX AND GLOSSARY

GOLDSMITH

*He writes indexes to perfection.*

— OLIVER GOLDSMITH, *Citizen of the World* (1762)

When an index entry refers to a page containing a relevant exercise, see also the *answer* to that exercise for further information. An answer page is not indexed here unless it refers to a topic not included in the statement of the exercise.

- 2-letter block codes, 28.
- 2-letter words of English, 8, 48.
- 2-regular graphs, 25, 31.
- 3-letter words of English, 8, 48.
- 4-letter codewords, 9–18, 28–30.
- 4-letter words of English, 8, 48.
- 5-letter words of English, 8–9, 28, 31, 33.
- 6-letter and  $k$ -letter words of English, 8–9, 48.
- $\gamma$  (Euler’s constant), as source of “random” data, 19.
- $\Lambda$  (the null link), 43, 48.
- $\pi$  (circle ratio), as source of “random” data, 19–20, 22, 29.
- $\phi$  (golden ratio), as source of “random” data, 19.
  
- A posteriori probability, 42.
- Active elements of a list, 12.
- Ahrens, Joachim Heinrich Lüdecke, 6, 27, 36.
- Alekhnovich, Michael (Misha) Valentinovich (Алехнович, Михаел (Миша) Валентинович), 26.
- Alphabet, 31.
- Anacrostic puzzle, 33.
- Analysis of algorithms, 29, 30.
- Aperiodic words, 10, 28, 39.
- ARCS( $v$ ) (first arc of vertex  $v$ ), 33, 48.
- Armbruster, Franz Owen, 24.
  
- Backjumping, 26.
- Backmarking, 26.
- Backtrack programming, 2– $\infty$ .
  - efficiency of, 35.
  - history of, 2, 5–6, 25–26.
  - introduction to, 2–32.
  - variant structure, 48–49.
- Backtrack trees, 3, 4, 7, 9–11, 18–20, 24, 26, 41, 42, 44–46.
  - estimating the size of, 20, 41.
- Baumert, Leonard Daniel, 26, 47, 51.
- Bees, queen, 27.
- Bernoulli, Jacques (= Jakob = James), 25.
- Bezzel, Max Friedrich Wilhelm, 25.
- Biased random walks, 30–31, 43.
- Binary partitions, 31.
- Binomial trees, 21.
- Bitner, James Richard, 6, 26.
  
- Bitwise operations, 5, 27, 37.
- Block codes, 9, 28.
- Borodin, Allan Bertram, 26.
- Botermans, Jacobus (= Jack) Petrus Hermana, 45.
- Boundary markers, 29.
- Breadth-first search, 36.
- Breaking symmetry, 8, 14.
- British National Corpus, 8, 48.
- Broken diagonals, *see* Wraparound.
- Bumping the current stamp, 16, 29.
- Bunch, Steve Raymond, 6.
- Buresh-Oppenheimer, Joshua, 26.
  
- Cache-friendly data structures, 11.
- Canonical labeling, 33.
- Carroll, Lewis (= Dodgson, Charles Lutwidge), iii.
- Carteblanche, Filet de (pseudonym, most likely of C. A. B. Smith), 24.
- Cavenagh, Nicholas John, 36.
- Cayley, Arthur, 31.
- Cells of memory, 11.
- Chatterjee, Sourav (সৌরভ চ্যাটার্জী), 22.
- Chessboard, 2–6, 22–26, 27, 31.
- Chiral symmetry, 27.
- Closed lists, 14.
- Clueless anacrostic, 33.
- CNF: Conjunctive normal form, 41.
- Codewords, commafree, 9–18, 28–30.
- Coin flipping, 30.
- Combinations, 27.
- Commafree codes, 9–18, 26, 28–30.
- Compilers, 15.
- Complexity of calculation, 41.
- Compressed tries, 38.
- Computational complexity, 41.
- Concatenation, 9.
- Connected subsets, 33.
- Constraints, 48.
- Cook, Matthew Makonnen, 44.
- Corner-to-corner paths, 22–23, 30–31.
- Cost function, 19, 42.
- Crick, Francis Harry Compton, 9.
- Cubes, 24–25, 31.
- Cumulative binomial distribution, 42.
- Cutoff principle, 7.
- Cutoff properties, 2, 5, 10, 18, 27.
- Cyclic shifts, 10, 28.



- Dancing links, 7.  
 Data structures, 4–6, 9, 11–14, 18, 29.  
 de Cartebianche, Filet (pseudonym, most likely of C. A. B. Smith), 24.  
 de Jaenisch, Carl Friedrich Andreevitch (Янишъ, Карлъ Андреевичъ), 35.  
 Degree of a node, 19.  
 Deletion operation, 7, 12–13, 40.  
 Depth-first search, 25.  
 Diaconis, Persi Warren, 22.  
 Diagonal lines (slope  $\pm 1$ ), 3, 36, *see also* Wraparound.  
 Digraphs, 28, 34.  
 DIMACS: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, inaugurated in 1990.  
 Dips, 29.  
 Directed graphs versus undirected graphs, 34.  
 Discarded data, 22.  
 Discrete probabilities, 1.  
 Disjoint sets, 25, 44.  
 Distributed computations, 6.  
 Divergence, Kullback–Leibler, 43.  
 Divide and conquer paradigm, 24.  
 Dodgson, Charles Lutwidge, iii.  
 Domains, 2, 27, 28.  
 Double-croscics, 48.  
 DOWDATING versus UPDATING, 4–5, 11, 15.  
 Dual oriented spanning tree, 34.  
 Dual solutions, 6, 8, 28.  
 Dynamic ordering, 10–11, 24, 26, 30, 46.  
 Dynamic shortest distances, 30.
- e*, as source of “random” data, 19.  
 Eastman, Willard Lawrence, 28, 29, 39.  
 Eight queens problem, 3–4, 19–20, 25–26.  
 Empty lists, 12, 17.  
 Engelhardt, Matthias Rüdiger, 6, 35.  
 English words, 8–9, 28, 33.  
 Ernst, George Werner, 45.  
 Error bars, 22.  
 Estimates of run time, 18–21.  
 Estimating the number of solutions, 21–23.
- Factorization of problems, 24–25, 31, 33.  
 Fallback points, 16.  
 Finkel, Raphael Ari, 26.  
 Five-letter words, 8–9, 28.  
 Fixed point of recursive formula, 47.  
 Floyd, Robert W., 15.  
 Fool’s Disk, 31.  
 Four-letter codewords, 9–18, 28–30.  
 FPGA devices: Field-programmable gate arrays, 6.  
 Frames, 11.  
 Franel, Jérôme, 36.
- Gardner, Erle Stanley, 2.  
 Gaschnig, John Gary, 26.  
 Gattegno, Bernard, 38.  
 Gauß (= Gauss), Johann Friderich Carl (= Carl Friedrich), 25, 35.  
 Generating functions, 29, 42, 43.  
 Gigamem ( $G\mu$ ): One billion memory accesses, 5.  
 Global variables, 27.  
 Goldsmith, Oliver, 50.  
 Goldstein, Michael Milan, 45.  
 Golomb, Solomon Wolf, 9, 26, 47, 51.  
 Gordon, Basil, 9, 51.  
 Graders, 46.  
 Grid graphs, 27, 33.  
   oriented, 34.  
 Gridgeman, Norman Theodore, 44.  
 Griffith, John Stanley, 9.
- Hales, Alfred Washington, 51.  
 Hall, Marshall, Jr., 26.  
 Hamilton, William Rowan, paths, 23.  
 Hammersley, John Michael, 26.  
 Handscomb, David Christopher, 26.  
 Height of binary trees, 31.  
 Hexagons, 27.  
 Historical notes, 2, 5–6, 25–26.  
 Hoffmann, Louis (pen name of Angelo John Lewis), 24.  
 Homomorphic images, 24.  
 Honeycombs, 27.  
 Hurwitz, Adolf, 36.
- IBM 704 computer, 48.  
 IBM 1620 computer, 6.  
 IBM System 360-75 computer, 6.  
 Impagliazzo, Russell Graham, 26.  
 Importance sampling, 26.  
 Independent subproblems, 24.  
 Indeterminate statements, 47.  
 Inner loops, 35.  
 Insertion operation, 12.  
 Instant Insanity, 24–25, 31.  
 Integer partitions, 27, 31.  
 Internet, ii, iii.  
 Invariant relations, 45.  
 Inverse lists, 12–15, 17.  
 Inverse permutations, 12–13.  
 Iteration versus recursion, 27, 35.
- Jaenisch, Carl Friedrich Andreevitch de (Янишъ, Карлъ Андреевичъ), 35.  
 Jewett, Robert Israel, 51.  
 Jiggs, B. H. (pen name of Baumert, Hales, Jewett, Imaginary, Golomb, Gordon, and Selfridge), 17.
- Kennedy, Michael David, 6.  
 Kilomem ( $K\mu$ ): One thousand memory accesses, 44.  
 King, Benjamin Franklin, Jr., 2.

- King paths, 22–23, 26, 30–31.  
 Kingsley, Hannah Elizabeth Seelman, 48.  
 Kleber, Michael Steven, 44.  
 Knight moves, 23.  
 Knuth, Donald Ervin (高德纳), i, iv, 18, 26, 27, 38, 41, 46, 48.  
 Kullback, Solomon, 43.
- Langford, Charles Dudley, pairs, 6–8, 27–28.  
 Latin squares, 24.  
 Laxdal, Albert Lee, 17.  
 Le Nombre Treize, *see* Royal Aquarium Thirteen Puzzle.  
 Lehmer, Derrick Henry, 26.  
 Leibler, Richard Arthur, 43.  
 Lennon, John Winston Ono, 2.  
 Lewis, Angelo John, 51.  
 Lexicographic order, 2, 7, 25, 29, 33.  
 Lifting, 24–25.  
 Linked lists, 6–7, 40.  
 Load balancing, 26.  
 Lookahead, 10, 16, 26, 30.  
 Loose Langford pairs, 28.  
 Lucas, François Edouard Anatole, 25, 36.
- Magen, Avner (אבנר מגן), 26.  
 Mahler, Kurt, 44.  
 Manber, Udi (עודי מנבר) [not אודי!], 26.  
 Martingales, 30.  
 Masks, 35.  
 Mason, Perry, 2.  
 Megamem ( $M\mu$ ): One million memory accesses, 18.  
 MEM, an array of “cells,” 11–18, 29.  
 Memory constraints, historic, 40.  
 Mem ( $\mu$ ): One 64-bit memory access, 4.  
 Minimum remaining values heuristic, 26.  
 MMIX computer, ii.  
 Monte Carlo estimates, 18–23, 26, 30.  
 Moves, 11.  
 MPR: Mathematical Preliminaries Redux, v, 1.  
 Mystery text, 33.
- $n$ -letter words of English, 8.  
 $n$ -omino placement, 49.  
 $n$  queens problem, 3–6, 18–20, 25–27.  
 $n$ -tuples, 27.  
 Nauck, Franz, 25.  
 Nested parentheses, 27.  
 NEXT( $a$ ) (the next arc with the same initial vertex as  $a$ ), 33, 48.  
 Niho, Yoji Goff (仁保洋二), 41.  
 Nonisomorphic solutions, 30.
- Onnen, Hendrick, Sr., 6.  
 Optimization, 26.  
 Orgel, Leslie Eleazer, 9.  
 Oriented grids, 34.
- Oriented trees, 34.  
 Orthogonal lists, 28.  
 OSPD4: *Official SCRABBLE® Players Dictionary*, 8, 48.  
 Overflow of memory, 12, 16.
- $P_0()$ , 2.  
 Paradox, 32.  
 Parallel programming, 46.  
 Parent in a tree, 34.  
 Parentheses, 27.  
 Parity argument, 37, 45.  
 Parker, Ernest Tilden, 24.  
 Partitions, 27, 31.  
 Patents, 24.  
 Paths, simple, 22, 26, 30.  
 Pauls, Emil, 36.  
 Pencil-and-paper method, 18–20.  
 Pentominoes, 33.  
 Periodic sequences, 28.  
 Periodic words, 10, 13.  
 Permutations, 6, 27.  
 Phi ( $\phi$ ), as source of “random” data, 19.  
 Pi ( $\pi$ ), as source of “random” data, 19–20, 22, 29.  
 Pitassi, Toniann, 26.  
 Poison list, 16–17, 29–30, 42.  
 Pólya, György (= George), 36.  
 Polyominoes, 33.  
 Preußner, Thomas Bernd, 6, 35.  
 Prime strings, 10.  
 Priority branching trees, 26.  
 Probabilities, 1.  
 Profile of a tree, 3, 9, 28, 31, 42.  
 Propagation algorithm, 46.  
 Properties: Logical propositions (relations), 2, 27.
- q.s., 30.  
 Quantified Boolean formulas, 47.  
 Queen bees, 27.  
 Queens, *see*  $n$  queens problem.  
 Questionnaires, 31.  
 Queues, 40, 43.  
 Quick, Jonathan Horatio, 27.
- Radix  $m$  representation, 13.  
 Random bits, 30.  
 Random sampling, 18.  
 Random variables, 19.  
 Random walks, 18–23, 30–31.  
 Reachable subsets, 34.  
 Recurrence relations, 42, 44.  
   in a Boolean equation, 47.  
 Recursion versus iteration, 27, 35.  
 Recursive algorithms, 27, 42.  
 Reflection symmetry, 14, 27, 35, *see also* Dual solutions.  
 Registers, 5, 37–38.

- Reingold, Edward Martin (ריינגולד, יצחק משה בן חיים), 26.
- Rejection method, 19, 41.
- Restricted growth strings, 35.
- Reversible memory technique, 16, 30.
- Rivin, Igor (Ривин, Игорь Евгеньевич), 36.
- Root node, 19.
- Rosenbluth, Arianna Wright, 26.
- Rosenbluth, Marshall Nicholas, 26.
- Rotating Century Puzzle, *see* Fool's Disk.
- Rotation by  $90^\circ$ , 27.
- Royal Aquarium Thirteen Puzzle, 31.
- Running time estimates, 18–21.
- Safe Combination Puzzle, *see* Fool's Disk.
- Sample variance, 22.
- SAT solvers, 41.
- Saturating ternary addition, 38.
- Scholtz, Robert Arno, 39.
- Schossow, Frederick Alvin, 24.
- Schumacher, Heinrich Christian, 25, 35.
- Search rearrangement, *see* Dynamic ordering.
- Search trees, 3, 4, 7, 9–11, 18–20, 24, 26, 36, 41, 42, 44–46.  
estimating the size, 20, 41.
- Self-avoiding walks, 22, 26, 30.
- Self-reference, 32, 53.
- Self-synchronizing block codes, 9.
- Selfridge, John Lewis, 51.
- Semi-queens, 36.
- Sequential allocation, 29.
- Sequential lists, 11–15.
- Set partitions, 27.
- SGB, *see* Stanford GraphBase.
- Shortest distances, dynamic, 30.
- Signature of a trie node, 37.
- Simple paths, 22, 26, 30.
- Slack, 18, 42.
- Slocum, Gerald Kenneth (= Jerry), 45.
- Smith, Cedric Austen Bardell, 51.
- Spanning trees, 33–34.
- Speedy Schizophrenia, 31.
- Spiegelthal, Edwin Simeon, 48.
- Sprague, Thomas Bond, 5, 6, 27.
- Stacks, 16, 40.
- Stamping, iv, 16–19, 29.
- Standard deviation, 20, 22, 30.
- Stanford GraphBase, ii, 8.  
format for digraphs and graphs, 33, 48.
- Statistics, 22.
- Stirling, James, cycle numbers, 30.
- Students, 46.
- Substrings, 29.
- Subtrees, 20, 26.
- Superdips, 39.
- SWAC computer, 6.
- Symmetries, 30, 35.  
breaking, 8, 14.
- November 12, 2016
- Tagged vertices, 48–49.
- Tail inequality, 42.
- Tantalizer, *see* Instant Insanity.
- Teramem ( $T\mu$ ): One trillion memory accesses, 9.
- TIP( $a$ ) (final vertex of arc  $a$ ), 33, 48.
- Torus, 27.
- Tot tibi ..., 25.
- Traub, Joseph Frederick, 41.
- Trémaux, Charles Pierre, 25.
- Tries, 8–9, 28, 37.  
compressed, 38.
- Tuples, 27.
- Twenty Questions, 32.
- Two-letter block codes, 28.
- UCLA: The University of California at Los Angeles, 6.
- Undirected graphs versus directed graphs, 34.
- UNDO stack, 16.
- Undoing, 4–5, 7, 15–16, 29.
- Uniformly random numbers, 41.
- Unique solutions, 44–45.
- Unit clauses, 41.
- University of California, 6.
- University of Dresden, 6.
- University of Illinois, 6.
- University of Tennessee, 6.
- Unordered sequential lists, 11.
- Unordered sets, 40.
- Uppercase letters, 31.
- Utility fields in SGB format, 47.
- $v$ -reachable subsets, 34.
- Valid gradings, 46.
- Vardi, Ilan, 36.
- Variance of a random variable, 22, 30.
- Venn, John, diagram, 43.
- Visiting an object, 2, 4, 5, 7, 18.
- Walker, Robert John, 2, 5, 6, 26.
- Wanless, Ian Murray, 36.
- Washington Monument Puzzle, *see* Fool's Disk.
- Welch, Lloyd Richard, 9.
- Wells, Mark Brimhall, 26.
- White squares, 27.
- Woods, Donald Roy, 32, 47.
- Word rectangles, 8–9, 28.
- WORDS( $n$ ), the  $n$  most common five-letter words of English, 8, 33.
- Worst-case bounds, 29.
- Wraparound, 27.
- Yao, Andrew Chi-Chih (姚期智), 41.
- ZDD: A zero-suppressed decision diagram, 23, 30.
- Zimmermann, Paul Vincent Marie, 36.

Note to readers:  
Please ignore these  
sidenotes; they're just  
hints to myself for  
preparing the index,  
and they're often flaky!

KNUTH

# THE ART OF COMPUTER PROGRAMMING

VOLUME 4    PRE-FASCICLE 5C

## DANCING LINKS

DONALD E. KNUTH *Stanford University*

ADDISON-WESLEY



April 15, 2017

Internet page <http://www-cs-faculty.stanford.edu/~knuth/taocp.html> contains current information about this book and related books.

See also <http://www-cs-faculty.stanford.edu/~knuth/sgb.html> for information about *The Stanford GraphBase*, including downloadable software for dealing with the graphs used in many of the examples in Chapter 7.

See also <http://www-cs-faculty.stanford.edu/~knuth/mmixware.html> for downloadable software to simulate the MMIX computer.

See also <http://www-cs-faculty.stanford.edu/~knuth/programs.html> for various experimental programs that I wrote while writing this material (and some data files).

Copyright © 2017 by Addison–Wesley

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher, except that the official electronic file may be used to print single copies for personal (not commercial) use.

Zeroth printing (revision -76), 11 April 2017

April 15, 2017

## PREFACE

*With this issue we have terminated the section "Short Notes."  
... It has never been "crystal clear" why a Contribution cannot be short,  
just as it has occasionally been verified in these pages  
that a Short Note might be long.*

— ROBERT A. SHORT, *IEEE Transactions on Computers* (1973)

THIS BOOKLET contains draft material that I'm circulating to experts in the field, in hopes that they can help remove its most egregious errors before too many other people see it. I am also, however, posting it on the Internet for courageous and/or random readers who don't mind the risk of reading a few pages that have not yet reached a very mature state. *Beware:* This material has not yet been proofread as thoroughly as the manuscripts of Volumes 1, 2, 3, and 4A were at the time of their first printings. And alas, those carefully-checked volumes were subsequently found to contain thousands of mistakes.

Given this caveat, I hope that my errors this time will not be so numerous and/or obtrusive that you will be discouraged from reading the material carefully. I did try to make the text both interesting and authoritative, as far as it goes. But the field is vast; I cannot hope to have surrounded it enough to corral it completely. So I beg you to let me know about any deficiencies that you discover.

To put the material in context, this portion of fascicle 5 previews Section 7.2.2.1 of *The Art of Computer Programming*, entitled "Dancing links." It develops an important data structure technique that is suitable for *backtrack programming*, which is the main focus of Section 7.2.2. Several subsections (7.2.2.2, 7.2.2.3, etc.) will follow.

\* \* \*

The explosion of research in combinatorial algorithms since the 1970s has meant that I cannot hope to be aware of all the important ideas in this field. I've tried my best to get the story right, yet I fear that in many respects I'm woefully ignorant. So I beg expert readers to steer me in appropriate directions.

Please look, for example, at the exercises that I've classed as research problems (rated with difficulty level 46 or higher), namely exercises 182, 263, ...; I've also implicitly mentioned or posed additional unsolved questions in the answers to exercises 82, 86, 210, 250, 256, 261, ... Are those problems still open? Please inform me if you know of a solution to any of these intriguing

questions. And of course if no solution is known today but you do make progress on any of them in the future, I hope you'll let me know.

I urgently need your help also with respect to some exercises that I made up as I was preparing this material. I certainly don't like to receive credit for things that have already been published by others, and most of these results are quite natural "fruits" that were just waiting to be "plucked." Therefore please tell me if you know who deserves to be credited, with respect to the ideas found in exercises 5, 6, 10, 15, 20, 21, 31, 40, 58, 60, 61, 75, 85, 158, 163, 177, 198(d), 206, 207, 208, 210, 218, 222, 240, 241, 242, 243, 244, 247, 248, 249, 252, 253, 255, 258, 261, 262, . . . . Furthermore I've credited exercises . . . to unpublished work of . . . . Have any of those results ever appeared in print, to your knowledge?

Jellis  
Huang  
Sicherman  
*FGbook*  
Knuth

\* \* \*

Special thanks are due to George Jellis for answering dozens of historical queries, as well as to Wei-Hwa Huang, George Sicherman, and . . . for their detailed comments on my early attempts at exposition. And I want to thank numerous other correspondents who have contributed crucial corrections.

\* \* \*

I happily offer a "finder's fee" of \$2.56 for each error in this draft when it is first reported to me, whether that error be typographical, technical, or historical. The same reward holds for items that I forgot to put in the index. And valuable suggestions for improvements to the text are worth 32¢ each. (Furthermore, if you find a better solution to an exercise, I'll actually do my best to give you immortal glory, by publishing your name in the eventual book:—)

In the preface to Volume 4B I plan to introduce the abbreviation *FGbook* for my book *Selected Papers on Fun and Games* (Stanford: CSLI Publications, 2011), because I will be making frequent reference to it in connection with recreational problems.

Cross references to yet-unwritten material sometimes appear as '00'; this impossible value is a placeholder for the actual numbers to be supplied later.

Happy reading!

*Stanford, California*  
*99 Umbruary 2016*

D. E. K.

## Part of the Preface to Volume 4B

During the years that I've been preparing Volume 4, I've often run across basic techniques of probability theory that I would have put into Section 1.2 of Volume 1 if I'd been clairvoyant enough to anticipate them in the 1960s. Finally I realized that I ought to collect most of them together in one place, near the beginning of Volume 4B, because the story of these developments is too interesting to be broken up into little pieces scattered here and there.

Therefore this volume begins with a special section entitled “Mathematical Preliminaries Redux,” and future sections use the abbreviation ‘MPR’ to refer to its equations and its exercises.

\* \* \*

Several exercises involve the lists of English words that I've used in preparing examples. You'll need the data from

`http://www-cs-faculty.stanford.edu/~knuth/wordlists.tgz`

if you have the courage to work those exercises.



*What a dance  
do they do  
Lordy, how I'm tellin' you!*

— HARRY BARRIS, *Mississippi Mud* (1927).

BARRIS  
FIELDS  
undoing  
exact covering-  
0s and 1s

*Don't lose your confidence if you slip,  
Be grateful for a pleasant trip,  
And pick yourself up, dust yourself off, start all over again.*

— DOROTHY FIELDS, *Pick Yourself Up* (1936)

**7.2.2.1. Dancing links.** One of the chief characteristics of backtrack algorithms is the fact that they usually need to *undo* everything that they *do* to their data structures. Blah blah de blah blah blah.

\* \* \*

**Exact cover problems.** We will be seeing many examples where links dance happily and efficiently, as we study more and more examples of backtracking. The beauty of the idea can perhaps be seen most naturally in an important class of problems known as *exact covering*: We're given an  $m \times n$  matrix  $A$  of 0s and 1s, and the problem is to find a subset of rows whose sum is exactly 1 in every column. For example, consider the  $6 \times 7$  matrix

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}. \quad (20)$$

Each row of  $A$  corresponds to a subset of a 7-element universe. A moment's thought shows that there's only one way to cover all seven of these columns with disjoint rows, namely by choosing rows 1, 4, and 5. We want to teach a computer how to solve such problems, when there are many, many rows and many columns.

DUDENEY  
 CLARKE  
 GOLOMB  
 Golomb  
 Conway

*If mounted on cardboard, [these pieces]  
 will form a source of perpetual amusement in the home.*

— HENRY E. DUDENEY, *The Canterbury Puzzles* (1907)

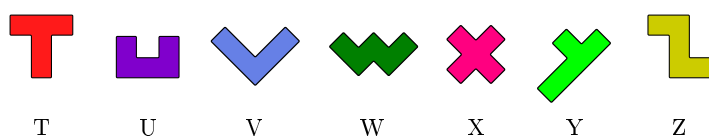
*Very gently, he replaced the titanite cross  
 in its setting between the F, N, U, and V pentominoes.*

— ARTHUR C. CLARKE, *Imperial Earth* (1976)

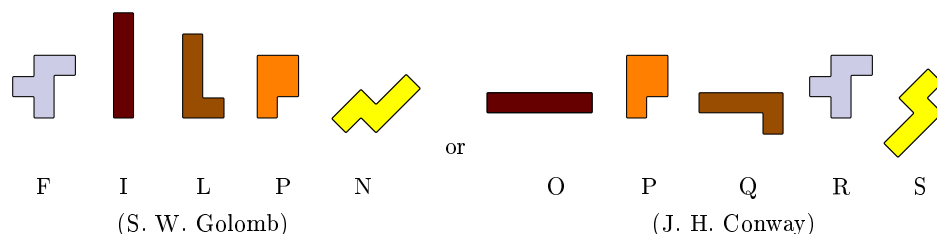
*Which English nouns ending in -o pluralize with -s and which with -es?  
 If the word is still felt as somewhat alien, it takes -s,  
 while if it has been fully naturalized into English, it takes -es.  
 Thus, echoes, potatoes, tomatoes, dingoes, embargoes, etc.,  
 whereas Italian musical terms are altos, bassos, cantos, pianos, solos, etc.,  
 and there are Spanish words like tangos, armadillos, etc.  
 I once held a trademark on 'Pentomino(-es)', but I now prefer  
 to let these words be my contribution to the language as public domain.*

— SOLOMON W. GOLOMB, letter to Donald Knuth (16 February 1994)

Everybody agrees that seven of the pentominoes should be named after seven consecutive letters of the alphabet:



But two different systems of nomenclature have been proposed for the other five:



where Golomb likes to think of the word 'Filipino' while Conway prefers to map the twelve pentominoes onto the twelve consecutive letters. Conway's scheme tends to work better in computer programs.

*A minimum number of blocks of simple form are employed. . . . Experiments and calculations have shown that from the set of seven blocks it is possible to construct approximately the same number of geometrical figures as could be constructed from twenty-seven separate cubes.*

— PIET HEIN, *United Kingdom Patent Specification 420,349* (1934)

HEIN  
cuboids  
parallelepipeds  
Hein  
Soma  
Gardner  
Parker Brothers  
pentominoes  
canonical  
factoring

\* \* \*

The simplest polycubes are *cuboids*—also called rectangular parallelepipeds by people who like long names. But things get even more interesting when we consider noncuboidal shapes. Piet Hein noticed in 1933 that the seven smallest shapes of that kind, namely



1: bent    2: ell    3: tee    4: skew    5: L-twist    6: R-twist    7: claw

can be put together to form a  $3 \times 3 \times 3$  cube, and he liked the pieces so much that he called them *Soma*. Notice that the first four pieces are essentially planar, while the other three are inherently three-dimensional. Moreover, the two twists are mirror images: We can't change one into the other without entering the *fourth* dimension. Martin Gardner wrote about the joys of Soma in *Scientific American* **199, 3** (September 1958), 182–188, and it soon became wildly popular: More than two million SOMA<sup>®</sup> cubes were sold in America alone, after Parker Brothers began to market a well-made set with an instruction booklet written by Hein.

The task of packing these seven pieces into a cube is easy to formulate as an exact cover problem, just as we did when packing pentominoes. This time we have 24 3D-rotations of the pieces to consider, instead of 8 2D-rotations and/or 3D-reflections; so exercise 200 is used instead of exercise 140 to generate the rows of the problem. It turns out that there are 688 rows, involving 34 columns that we can call 1, 2, . . . , 7, 111, 112, . . . , 333. For example, the first row

1 111 121 211

characterizes one of the potential ways to place the “bent” piece 1.

Algorithm D needs just 407 megamems to find all 11,520 solutions to this problem. Furthermore, we can save most of that time by taking advantage of symmetry: Every solution can be rotated into a unique “canonical” solution in which the “ell” piece 2 has not been rotated; hence we can restrict that piece to only six placements, namely (111, 121, 131, 211), (112, 122, 132, 212), . . . , (213, 223, 233, 313)—all shifts of each other. This removes 138 rows, and the algorithm now finds the 480 canonical solutions in just 20 megamems. (These canonical solutions form 240 mirror-image pairs.)

**Factoring an exact cover problem.** In fact, we can simplify the Soma cube problem much further, so that all of its solutions can actually be found by hand in a reasonable time, by *factoring* the problem in a clever way. . . .

**Color-controlled covering.** *Take a break!* Before reading any further, please spend a minute or two solving the “word search” puzzle in Fig. 71; comparatively mindless puzzles like this one provide a low-stress way to sharpen your word-recognition skills. It can be solved easily—for instance, by making eight passes over the array—and the solution appears in Fig. 72.

color-controlled-  
word search  
color codes

**Fig. 71.** Find the mathematicians\*:

Put ovals around the following names where they appear in the  $15 \times 15$  array shown here, reading either forward or backward or upward or downward, or diagonally in any direction. After you’ve finished, the leftover letters will form a hidden message. (The solution appears on the next page.)

|           |           |             |
|-----------|-----------|-------------|
| ABEL      | HENSEL    | MELLIN      |
| BERTRAND  | HERMITE   | MINKOWSKI   |
| BOREL     | HILBERT   | NETTO       |
| CANTOR    | HURWITZ   | PERRON      |
| CATALAN   | JENSEN    | RUNGE       |
| FROBENIUS | KIRCHHOFF | STERN       |
| GLAISHER  | KNOPP     | STIELTJES   |
| GRAM      | LANDAU    | SYLVESTER   |
| HADAMARD  | MARKOFF   | WEIERSTRASS |

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| O | T | H | E | S | C | A | T | A | L | A | N | D | A | U |
| T | S | E | A | P | U | S | T | H | O | R | S | R | O | F |
| T | L | S | E | E | A | Y | R | R | L | Y | H | A | P | A |
| E | P | E | A | R | E | L | R | G | O | U | E | M | S | I |
| N | N | A | R | R | C | V | L | T | R | T | A | A | M | A |
| I | T | H | U | O | T | E | K | W | I | A | N | D | E | M |
| L | A | N | T | N | B | S | I | M | I | C | M | A | A | W |
| L | G | D | N | A | R | T | R | E | B | L | I | H | C | E |
| E | R | E | C | I | Z | E | C | E | P | T | N | E | D | Y |
| M | E | A | R | S | H | R | H | L | I | P | K | A | T | H |
| E | J | E | N | S | E | N | H | R | I | E | O | N | E | T |
| H | S | U | I | N | E | B | O | R | F | E | W | N | A | R |
| T | M | A | R | K | O | F | F | O | F | C | S | O | K | M |
| P | L | U | T | E | R | P | F | R | O | E | K | G | R | A |
| G | M | M | I | N | S | E | J | T | L | E | I | T | S | G |

Our goal in this section is not to discuss how to *solve* such puzzles; instead, we shall consider how to *create* them. It’s by no means easy to pack those 27 names into the box in such a way that their 184 characters occupy only 135 cells, with eight directions well mixed. How can that be done with reasonable efficiency?

For this purpose we shall extend the idea of exact covering by introducing “color codes.” . . .

\* The journal *Acta Mathematica* celebrated its 21st birthday by publishing a special *Table Générale des Tomes 1–35*, edited by Marcel Riesz (Uppsala: 1913), 179 pp. It contained a complete list of all papers published so far in that journal, together with portraits and brief biographies of all the authors. The 27 mathematicians mentioned in Fig. 71 are those who were subsequently mentioned in Volumes 1, 2, or 3 of *The Art of Computer Programming*—except for people like MITTAG-LEFFLER or POINCARÉ, whose names contain special characters.

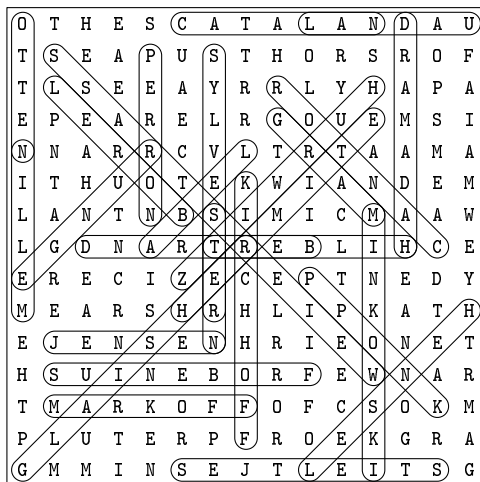
**Fig. 72.** Solution to the puzzle of the hidden mathematicians (Fig. 71). Notice that the central letter R actually participates in six different names:

- BERTRAND
- GLAISHER
- HERMITE
- HILBERT
- KIRCHHOFF
- WEIERSTRASS

The T to its left participates in five.

Here's what the leftover letters say:

These authors of early papers in *Acta Mathematica* were cited years later in *The Art of Computer Programming*.



### EXERCISES — First Set

- **5.** [26] Let  $T$  be any tree. Construct an unsolvable exact cover problem for which  $T$  is the backtrack tree traversed by Algorithm D; a *unique* column should have the minimum size whenever step D? is encountered. Illustrate your construction when  $T = \wedge \wedge$ .
- 6.** [25] Continuing exercise 5, let  $T$  be a tree in which certain leaves have been distinguished from the others and designated as “solutions.”
- Show that some such trees never match the behavior of Algorithm D.
  - Characterize all such trees that *do* arise, having solutions where indicated.
- **10.** [21] Modify Algorithm D so that it doesn't require the presence of any primary columns in the rows. A valid solution should not contain any purely secondary rows; but it must intersect every such row. (For example, if only columns 1 and 2 of (20) were primary, the only valid solutions would be to choose rows  $\{2, 3\}$  or  $\{3, 4\}$ .)
- 15.** [M21] The solution to an exact cover problem with matrix  $A$  can be regarded as a binary vector  $x$  such that  $xA = 11\dots 1$ . The *distance* between two solutions  $x$  and  $x'$  can then be defined as the Hamming distance  $d(x, x') = \nu(x \oplus x')$ , the number of places where  $x$  and  $x'$  differ. The *diversity* of  $A$  is the minimum distance between two of its solutions. (If  $A$  has at most one solution, its diversity is infinite.)
- Is it possible to have diversity 1?
  - Is it possible to have diversity 2?
  - Is it possible to have diversity 3?
  - Prove that if  $A$  represents a *uniform* exact cover problem, the distance between solutions is always even.
  - Most of the exact cover problems that arise in applications are at least *quasi-uniform*, in the sense that they have a nonempty subset  $C$  of primary columns such that  $A \upharpoonright C$  has the same number of 1s in every row. (For example, every polyomino or polycube packing problem is quasi-uniform, because every row of the matrix specifies exactly one piece name.) Can such problems have odd distances?
- 19.** [M16] Given an exact cover problem  $A$ , construct an exact cover problem  $A'$  that has exactly one more solution than  $A$  does. [Consequently it is NP-hard to determine whether an exact cover problem with at least one solution has more than one solution.] Assume that  $A$  contains no all-zero rows.
- 20.** [M25] Given an exact cover problem  $A$ , construct an exact cover problem  $A'$  such that (i)  $A'$  has at most three 1s in every column; (ii)  $A'$  and  $A$  have exactly the same number of solutions.
- 21.** [M21] Continuing exercise 20, construct  $A'$  having *exactly* three 1s per column.
- **24.** [30] Given an  $m \times n$  exact cover problem  $A$  with exactly three 1s per column, construct a generalized “instant insanity” problem with  $N = O(n)$  cubes and  $N$  colors that is solvable if and only if  $A$  is solvable. (See 7.2.2–(36).)
- **26.** [M24] A *grope* is a set  $G$  together with a binary operation  $\circ$ , in which the identity  $x \circ (y \circ x) = y$  is satisfied for all  $x \in G$  and  $y \in G$ .
- Prove that the identity  $(x \circ y) \circ x = y$  also holds, in every grope.
  - Which of the following “multiplication tables” define a grope on  $\{0, 1, 2, 3\}$ ?

backtrack tree  
 distance  
 Hamming distance  
 diversity  
 uniform  
 quasi-uniform  
 NP-hard  
 unique solution  
 instant insanity  
 grope  
 binary operation  
 multiplication tables

|      |      |      |      |      |
|------|------|------|------|------|
| 0123 | 0321 | 0132 | 0231 | 0312 |
| 1032 | 3210 | 1023 | 3102 | 2130 |
| 2301 | 2103 | 3210 | 1320 | 3021 |
| 3210 | 1032 | 2301 | 2013 | 1203 |











(In the first example,  $x \circ y = x \oplus y$ ; in the second,  $x \circ y = (-x - y) \bmod 4$ . The last two have  $x \circ y = x \oplus f(x \oplus y)$  for certain functions  $f$ .)

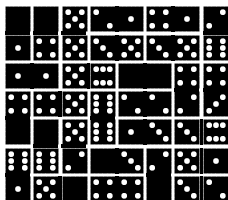
- c) For all  $n$ , construct a grope whose elements are  $\{0, 1, \dots, n-1\}$ .  
 d) Consider the exact cover problem that has  $n^2$  columns  $(x, y)$  for  $0 \leq x, y < n$  and the following  $n + (n^3 - n)/3$  rows:  
 i)  $\{(x, x)\}$ , for  $0 \leq x < n$ ;  
 ii)  $\{(x, x), (x, y), (y, x)\}$ , for  $0 \leq x < y < n$ ;  
 iii)  $\{(x, y), (y, z), (z, x)\}$ , for  $0 \leq x < y, z < n$ .

Show that its solutions are in one-to-one correspondence with the multiplication tables of gropes on the elements  $\{0, 1, \dots, n-1\}$ .

- e) Element  $x$  of a grope is *idempotent* if  $x \circ x = x$ . If  $k$  elements are idempotent and  $n - k$  are not, prove that  $k \equiv n^2 \pmod{3}$ .

**27.** [21] Modify the exact cover problem of exercise 26(d) in order to find the multiplication tables of (a) all idempotent gropes—gropes such that  $x \circ x = x$  for all  $x$ ; (b) all commutative gropes—gropes such that  $x \circ y = y \circ x$  for all  $x$  and  $y$ ; (c) all gropes with an identity element—gropes such that  $x \circ 0 = 0 \circ x = x$  for all  $x$ .

**30.** [21] *Dominosa* is a solitaire game in which you “shuffle” the 28 pieces , , , , , , , , ,  of double-six dominoes and place them at random into a  $7 \times 8$  frame. Then you write down the number of spots in each cell, put the dominoes away, and try to reconstruct their positions based only on that  $7 \times 8$  array of numbers. For example,



yields the array

$$\begin{pmatrix} 0 & 0 & 5 & 2 & 1 & 4 & 1 & 2 \\ 1 & 4 & 5 & 3 & 5 & 3 & 5 & 6 \\ 1 & 1 & 5 & 6 & 0 & 0 & 4 & 4 \\ 4 & 4 & 5 & 6 & 2 & 2 & 2 & 3 \\ 0 & 0 & 5 & 6 & 1 & 3 & 3 & 6 \\ 6 & 6 & 2 & 0 & 3 & 2 & 5 & 1 \\ 1 & 5 & 0 & 4 & 4 & 0 & 3 & 2 \end{pmatrix}.$$

- a) Show that *another* placement of dominoes also yields the same matrix of numbers.  
 b) What domino placement yields the array

$$\begin{pmatrix} 3 & 3 & 6 & 5 & 1 & 5 & 1 & 5 \\ 6 & 5 & 6 & 1 & 2 & 3 & 2 & 4 \\ 2 & 4 & 3 & 3 & 3 & 6 & 2 & 0 \\ 4 & 1 & 6 & 1 & 4 & 4 & 6 & 0 \\ 3 & 0 & 3 & 0 & 1 & 1 & 4 & 4 \\ 2 & 6 & 2 & 5 & 0 & 5 & 0 & 0 \\ 2 & 5 & 0 & 5 & 4 & 2 & 1 & 6 \end{pmatrix}?$$

► **31.** [20] Show that *Dominosa* reconstruction is a special case of 3D MATCHING.

**32.** [M22] Generate random instances of *Dominosa*, and estimate the probability of obtaining a  $7 \times 8$  matrix with a unique solution. Use two models of randomness: (i) Each matrix whose elements are permutations of the multiset  $\{8 \times 0, 8 \times 1, \dots, 8 \times 6\}$  is equally likely; (ii) each matrix obtained from a random shuffle of the dominoes is equally likely.

**39.** [20] By setting up an exact cover problem and solving it with Algorithm D, show that the queen graph  $Q_8$  (exercise 7.1.4–241) cannot be colored with eight colors.

**40.** [21] In how many ways can  $Q_8$  be colored in a “balanced” fashion, using eight queens of color 0 and seven each of colors 1 to 8?

idempotent  
 commutative  
 identity element  
 Dominosa  
 solitaire  
 game  
 Pijanowski solitaire, see Dominosa  
 dominoes  
 3D MATCHING  
 permutations of the multiset  
 queen graph  
 colored

- **50.** [21] If we merely want to count the number of solutions to an exact cover problem, without actually constructing them, a completely different approach based on bitwise manipulation instead of list processing is sometimes useful.

The following naïve algorithm illustrates the idea: We're given an  $m \times n$  matrix of 0s and 1s, represented as  $n$ -bit vectors  $r_1, \dots, r_m$ . The algorithm works with a (potentially huge) database of pairs  $(s_j, c_j)$ , where  $s_j$  is an  $n$ -bit number representing a set of columns, and  $c_j$  is a positive integer representing the number of ways to cover that set exactly. Let  $p$  be the  $n$ -bit mask that represents the primary columns.

**N1.** [Initialize.] Set  $N \leftarrow 1$ ,  $s_1 \leftarrow 0$ ,  $c_1 \leftarrow 1$ ,  $k \leftarrow 1$ .

**N2.** [Done?] If  $k > m$ , terminate; the answer is  $\sum_{j=1}^N c_j [s_j \& p = p]$ .

**N3.** [Append  $r_k$  where possible.] Set  $t \leftarrow r_k$ . For  $N \geq j \geq 1$ , if  $s_j \& t = 0$ , insert  $(s_j + t, c_j)$  into the database (see below).

**N4.** [Loop on  $k$ .] Set  $k \leftarrow k + 1$  and return to N2. ■

To insert  $(s, c)$  there are two cases: If  $s = s_i$  for some  $(s_i, c_i)$  already present, we simply set  $c_i \leftarrow c_i + c$ . Otherwise we set  $N \leftarrow N + 1$ ,  $s_N \leftarrow s$ ,  $c_N \leftarrow c$ .

Show that this algorithm can be significantly improved by using the following trick: Set  $u_k \leftarrow r_k \& \bar{f}_k$ , where  $f_k = r_{k+1} | \dots | r_m$  is the bitwise OR of all future rows. If  $u_k \neq 0$ , we can remove any item from the database for which  $s_j$  does not contain  $u_k \& p$ . We can also exploit the nonprimary columns of  $u_k$  to compress the database further.

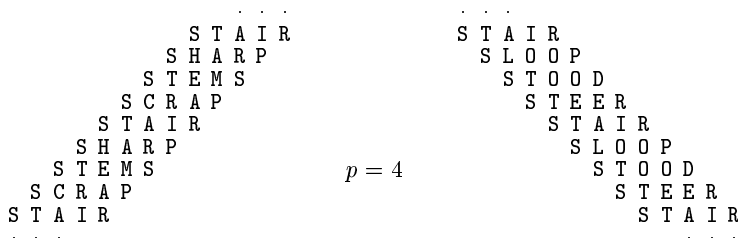
- 51.** [25] Implement the improved algorithm of the previous exercise, and compare its running time to that of Algorithm D when applied to the  $n$  queens problem.
- 52.** [M21] Explain how the method of exercise 50 could be extended to give representations of all solutions, instead of simply counting them.
- **58.** [20] Algorithm D can be extended in the following curious way: Let  $p$  be the primary column that is covered first, and suppose that there are  $k$  ways to cover it. Suppose further that the  $j$ th option for  $p$  ends with a secondary column  $s_j$ , where  $\{s_1, \dots, s_k\}$  are distinct. Modify the algorithm so that, whenever a solution contains the  $j$ th option for  $p$ , it leaves columns  $\{s_1, \dots, s_{j-1}\}$  uncovered. (In other words, the modified algorithm will emulate the behavior of the unmodified algorithm on a much larger instance, in which the  $j$ th option for  $p$  contains all of  $s_1, s_2, \dots, s_j$ .)
- **60.** [25] A *minimax solution* to an exact cover problem is one whose maximum row number is as small as possible. Explain how to modify Algorithm C so that it determines all of the minimax solutions (omitting any that are known to be worse).
- 61.** [22] Sharpen the algorithm of exercise 60 so that it produces *exactly one* minimax solution — unless, of course, there are no solutions at all.
- 64.** [20] A *double word square* is an  $n \times n$  array whose rows and columns contain  $2n$  different words. Encode this problem as an exact cover problem with color controls. Can you save a factor of 2 by not generating the transpose of previous solutions? Does Algorithm C compete with the algorithm of exercise 7.2.2–28 (which was designed explicitly to handle word-square problems)?
- 65.** [21] Instead of finding *all* of the double word squares, we usually are more interested in finding the *best* one, in the sense of using only words that are quite common. For example, it turns out that a double word square can be made from the words of WORDS(1720) but not from those of WORDS(1719). Show that it's rather easy to find the smallest  $N$  such that WORDS( $N$ ) supports a double word square, via dancing links.

exact cover problem  
bitwise manipulation  
breadth-first  
0s and 1s  
primary columns  
bitwise AND  
bitwise OR  
nonprimary columns  
 $n$  queens problem  
minimax solution  
double word square  
word square, double



**66.** [24] What are the best double word squares of sizes  $2 \times 2$ ,  $3 \times 3$ ,  $\dots$ ,  $7 \times 7$ , in the sense of exercise 65, with respect to *The Official SCRABBLE® Players Dictionary*? [Exercise 7.2.2–32 considered the analogous problem for *symmetric* word squares.]

► **68.** [22] A *word stair* of period  $p$  is a cyclic arrangement of words, offset stepwise, that contains  $2p$  distinct words across and down. They exist in two varieties, left and right:

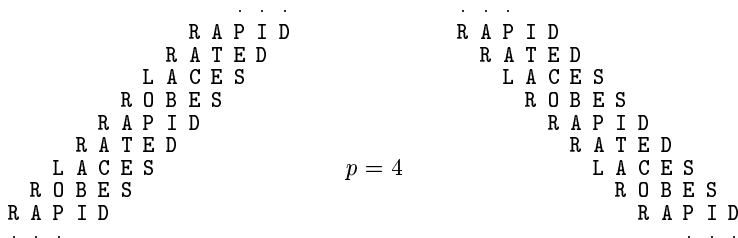


OSPD4  
word stair  
color controls  
NP-complete  
2D matching  
word search puzzle  
presidents  
I'm not sure  
how many of  
these names  
should go in  
the index

What are the best five-letter word stairs, in the sense of exercise 65, for  $1 \leq p \leq 10$ ?  
*Hint:* You can save a factor of  $2p$  by assuming that the first word is the most common.

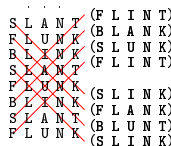
**69.** [40] For given  $N$ , find the largest  $p$  such that  $\text{WORDS}(N)$  supports a word stair of period  $p$ . (There are two questions for each  $N$ , examining stairs to the {left, right}.)

**70.** [24] Some  $p$ -word cycles define *two-way* word stairs that have  $3p$  distinct words:



What are the best five-letter examples of this variety, for  $1 \leq p \leq 10$ ?

**71.** [22] Another periodic arrangement of  $3p$  words, perhaps even nicer than that of exercise 70 and illustrated here for  $p = 3$ , lets us read them *diagonally* up or down, as well as across. What are the best five-letter examples of *this* variety, for  $1 \leq p \leq 10$ ? (Notice that there is  $2p$ -way symmetry.)



**75.** [25] Prove that the exact cover problem with color controls is NP-complete, even if every row of the matrix has only two entries.

**80.** [22] Using the “word search puzzle” conventions of Figs. 71 and 72, show that the words ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, ELEVEN, and TWELVE can all be packed into a  $6 \times 6$  square, leaving one cell untouched.

**81.** [22] Also pack *two* copies of ONE, TWO, THREE, FOUR, FIVE into a  $5 \times 5$  square.

► **82.** [32] The first 44 presidents of the U.S.A. had 38 distinct surnames: ADAMS, ARTHUR, BUCHANAN, BUSH, CARTER, CLEVELAND, CLINTON, COOLIDGE, EISENHOWER, FILLMORE, FORD, GARFIELD, GRANT, HARDING, HARRISON, HAYES, HOOVER, JACKSON, JEFFERSON, JOHNSON, KENNEDY, LINCOLN, MADISON, MCKINLEY, MONROE, NIXON, OBAMA, PIERCE, POLK, REAGAN, ROOSEVELT, TAFT, TAYLOR, TRUMAN, TYLER, VANBUREN, WASHINGTON, WILSON.

- a) What's the smallest square into which all of these names can be packed, using word search conventions, and requiring all words to be *connected* via overlaps?
  - b) What's the smallest *rectangle*, under the same conditions?
- **83.** [25] Pack as many of the following words as possible into a  $9 \times 9$  array, simultaneously satisfying the rules of *both* word search *and* sudoku:

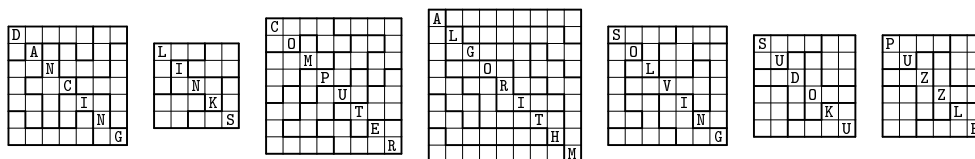
|      |          |           |       |           |      |
|------|----------|-----------|-------|-----------|------|
| ACRE | COMPARE  | CORPORATE | MACRO | MOTET     | ROAM |
| ART  | COMPUTER | CROP      | META  | PARAMETER | TAME |

- **85.** [28] A “wordcross puzzle” is the challenge of packing a given set of words into a rectangle under the following conditions: (i) All words must read either across or down, as in a crossword puzzle. (ii) No letters are adjacent unless they belong to one of the given words. (iii) The words are rookwise connected. For example, the eleven words ZERO, ONE, . . . , TEN can be placed into an  $8 \times 8$  square under constraints (i) and (ii) as shown; but (iii) is violated, because there are three different components.

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| T | H | R | E | E | F |
| W |   |   | S | I | X |
| O | N | E |   | V |   |
|   |   | Z | S | E | V |
|   |   | E | I | G | H |
|   |   | R |   | E | E |
| F | O | U | R | N |   |

Explain how to encode a wordcross puzzle as an exact cover problem with color controls. Use your encoding to find a correct solution to the problem above. Do those eleven words fit into a *smaller* rectangle, under conditions (i), (ii), and (iii)?

- 86.** [30] What's the smallest wordcross square that contains the surnames of the first 44 U.S. presidents? (Use the names in exercise 82, but change VANBUREN to VAN BUREN.)
- 87.** [21] Find all  $8 \times 8$  crossword puzzle diagrams that contain exactly (a) 12 3-letter words, 12 4-letter words, and 4 5-letter words; (b) 12 5-letter words, 8 2-letter words, and 4 8-letter words. They should have no words of other lengths.
- 90.** [24] Find the unique solutions to the following examples of polyomino sudoku:



- 100.** [M25] Consider a weighted exact cover problem in which we must choose 2 of 4 rows to cover column 1, and 5 of 7 rows to cover column 2; the rows don't interact.
- a) What's the size of the search tree if we branch first on column 1, then on column 2? Would it better to branch first on column 2, then on column 1?
  - b) Generalize part (a) to the case when column 1 needs  $p$  of  $p + d$  rows, while column 2 needs  $q$  of  $q + d$  rows, where  $q > p$  and  $d > 0$ .

connected  
word search  
sudoku  
wordcross  
crisscross puzzle, composing, see wordcross  
rookwise connected  
components  
crossword puzzle diagrams  
5-letter words  
polyomino sudoku  
sudoku  
weighted exact cover problem

## EXERCISES — Second Set

Hundreds of fascinating recreational problems have been based on polyominoes and their cousins (the polycubes, polyiamonds, polyhexes, polysticks, ...). The following exercises explore “the cream of the crop” of such classic puzzles, as well as a few gems that were not discovered until recently.

In most cases the idea is to find a good way to discover all solutions, usually by setting up an appropriate exact cover problem that can be solved without taking an enormous amount of time.

- 140. [25] Sketch the design of a utility program that will create sets of rows by which an exact cover solver will fill a given shape with a given set of polyominoes.

148. [18] Using Conway’s piece names, pack five pentominoes into the shape so that they spell a common English word when read from left to right.



- 150. [21] There are 1010 ways to pack the twelve pentominoes into a  $5 \times 12$  box, not counting reflections. What’s a good way to find them all, using Algorithm D?

151. [21] How many of those 1010 packings decompose into  $5 \times k$  and  $5 \times (12 - k)$ ?

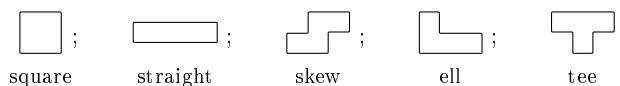
152. [21] In how many ways can the eleven nonstraight pentominoes be packed into a  $5 \times 11$  box, not counting reflections? (Reduce symmetry cleverly.)

154. [20] There are 2339 ways to pack the twelve pentominoes into a  $6 \times 10$  box, not counting reflections. What’s a good way to find them all, using Algorithm D?

155. [23] Continuing exercise 154, explain how to find special kinds of packings:

- Those that decompose into  $6 \times k$  and  $6 \times (10 - k)$ .
- Those that have all twelve pentominoes touching the outer boundary.
- Those with all pentominoes touching that boundary *except* for V, which doesn’t.
- Same as (c), with each of the other eleven pentominoes in place of V.
- Those with the *minimum* number of pentominoes touching the outer boundary.
- Those that are characterized by Arthur C. Clarke’s description, as quoted in the text. (That is, the X should touch only the F, N, U, and V—no others.)

157. [21] There are five different *tetrominoes*, namely



In how many essentially different ways can each of them be packed into an  $8 \times 8$  square together with the twelve pentominoes?

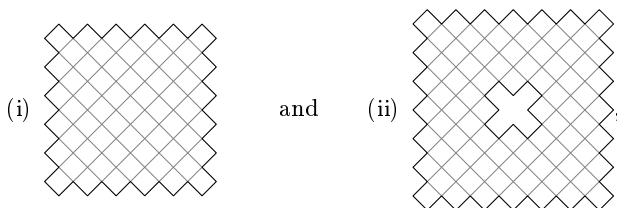
158. [21] If an  $8 \times 8$  checkerboard is cut up into thirteen pieces, representing the twelve pentominoes together with one of the tetrominoes, some of the pentominoes will have more black cells than white. Is it possible to do this in such a way that U, V, W, X, Y, Z have a black majority while the others do not?

159. [18] Design a nice, simple tiling pattern that’s based on the five tetrominoes.

160. [25] How many of the  $6 \times 10$  pentomino packings are *strongly three-colorable*, in the sense that each individual piece could be colored red, white, or blue in such a way that no pentominoes of the same color touch each other — not even at corner points?

Conway  
five-letter words  
pentominoes  
nonstraight  
symmetry  
pentominoes  
Clarke  
tetrominoes  
tetrominoes  
three-colorable  
graph coloring

- ▶ **162.** [20] The black cells of a square  $n \times n$  checkerboard form an interesting graph called the *Aztec diamond* of order  $n/2$ . For example, the cases  $n = 11$  and 13 are



checkerboard  
 Aztec diamond  
 symmetric  
 Möbius strip  
 faultfree  
 Benjamin  
 cube, wrapped  
 Knuth, Jill  
 Garcia, Hector  
 Kaplan  
 fence  
 holes  
 tatami  
 crossroads  
 O'Beirne  
 one-sided pentominoes

where (ii) has a “hole” showing the case  $n = 3$ . Thus (i) has 61 cells, and (ii) has 80.

- a) Find all ways to pack (i) with the twelve pentominoes and one monomino.
- b) Find all ways to pack (ii) with the 12 + 5 pentominoes and tetrominoes.

Speed up the process by not producing solutions that are symmetric to each other.

- ▶ **163.** [M26] Arrange the twelve pentominoes into a Möbius strip of width 4. The pattern should be “faultfree”: Every straight line must intersect some piece.

**164.** [40] (H. D. Benjamin, 1948.) Show that the twelve pentominoes can be wrapped around a cube of size  $\sqrt{10} \times \sqrt{10} \times \sqrt{10}$ . For example, here are front and back views of such a cube, made from twelve colorful fabrics by the author's wife in 1993:



(Photos by Hector Garcia)

What is the best way to do this, minimizing undesirable distortions at the corners?

- ▶ **165.** [22] (Craig S. Kaplan.) A polyomino can sometimes be surrounded by non-overlapping copies of itself that form a *fence*: Every cell that touches the polyomino — even at a corner — is part of the fence; conversely, every piece of the fence touches the inner polyomino. Furthermore, the pieces must not enclose any unoccupied “holes.”

Find the (a) smallest and (b) largest fences for each of the twelve pentominoes. (Some of these patterns are unique, and quite pretty.)

**166.** [22] Solve exercise 165 for fences that satisfy the *tatami* condition of exercise 7.1.4–215: No four edges of the tiles should come together at any “crossroads.”

- ▶ **167.** [27] Solomon Golomb discovered in 1965 that there's only one placement of two pentominoes in a  $5 \times 5$  square that blocks the placement of all the others.

Place (a)  $\{I, P, U, V\}$  and (b)  $\{F, P, T, U\}$  into a  $7 \times 7$  square in such a way that none of the other eight will fit in the remaining spaces.



**168.** [21] (T. H. O'Beirne, 1961.) The *one-sided pentominoes* are the eighteen distinct 5-cell pieces that can arise if we aren't allowed to flip pieces over:



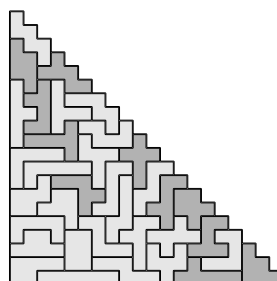
Notice that there now are two versions of F, L, P, N, Y, and Z.

In how many ways can all eighteen of them be packed into rectangles?

**169.** [21] Suppose you want to pack the twelve pentominoes into a  $6 \times 10$  box, *without* turning any pieces over. Then  $2^6$  different problems arise, depending on which sides of the one-sided pieces are present. Which of those 64 problems has (a) the fewest (b) the most solutions?

**170.** [21] When tetrominoes are both checkered and one-sided (see exercises 158 and 168), ten possible pieces arise. In how many ways can all ten of them fill a rectangle?

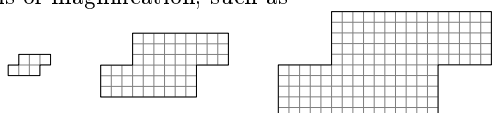
**175.** [20] There are 35 *hexominoes*, first enumerated in 1934 by the master puzzlist H. D. Benjamin. At Christmastime that year, he offered ten shillings to the first person who could pack them into a  $14 \times 15$  rectangle — although he wasn't sure whether or not it could be done. The prize was won by F. Kadner, who proved that the hexominoes actually *can't* be packed into *any* rectangle. Nevertheless, Benjamin continued to play with them, eventually discovering that they fit nicely into the triangle shown here.



tetrominoes  
checkered  
one-sided  
checkerboard dissections  
hexominoes  
Benjamin  
Kadner  
Hansson  
magnification  
triplication  
castles  
Stead  
pentominoes  
tetrominoes  
color controls  
hexominoes

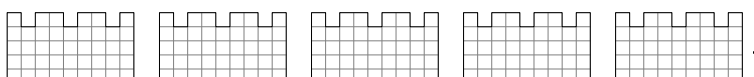
Prove Kadner's theorem. *Hint:* See exercise 158.

**176.** [24] (Frans Hansson, 1947.) The fact that  $35 = 1^2 + 3^2 + 5^2$  suggests that we might be able to pack the hexominoes into three boxes that represent a *single* hexomino shape at three levels of magnification, such as



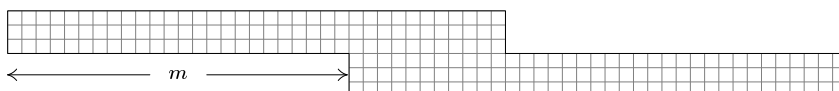
For which hexominoes can this be done?

► **177.** [30] Show that the 35 hexominoes can be packed into five “castles”:



In how many ways can this be done?

**178.** [41] For which values of  $m$  can the hexominoes be packed into a box like this?



**179.** [41] Perhaps the nicest hexomino packing uses a  $5 \times 45$  rectangle with 15 holes



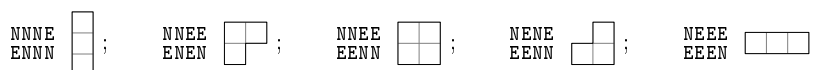
proposed by W. Stead in 1954. In how many ways can the 35 hexominoes fill it?

► **181.** [22] In how many ways can the twelve pentominoes be placed into an  $8 \times 10$  rectangle, leaving holes in the shapes of the five tetrominoes? (The holes should not touch the boundary, nor should they touch each other, even at corners; one example is shown at the right.) Explain how to encode this puzzle as an exact cover problem with color controls.



**182.** [46] If possible, solve the analog of exercise 181 for the case of 35 *hexominoes* in a  $5 \times 54$  rectangle, leaving holes in the shapes of the twelve *pentominoes*.

- ▶ **198.** [HM35] A *parallelogram polyomino*, or “parallomino” for short, is a polyomino whose boundary consists of two paths that each travel only north and/or east. (Equivalently, it is a “skew Young tableau” or a “skew Ferrers board,” the difference between the diagrams of two tableaux or partitions; see Sections 5.1.4 and 7.2.1.4.) For example, there are five parallominoes whose boundary paths have length 4:



- Find a one-to-one correspondence between the set of ordered trees with  $m$  leaves and  $n$  nodes and the set of parallominoes with width  $m$  and height  $n - m$ . The area of each parallomino should be the path length of its corresponding tree.
- Study the generating function  $G(w, x, y) = \sum_{\text{parallominoes}} w^{\text{area}} x^{\text{width}} y^{\text{height}}$ .
- Prove that the parallominoes whose width-plus-height is  $n$  have total area  $4^{n-2}$ .
- Part (c) suggests that we might be able to pack all of those parallominoes into a  $2^{n-2} \times 2^{n-2}$  square, *without* rotating them or flipping them over. Such a packing is clearly impossible when  $n = 3$  or  $n = 4$ ; but is it possible when  $n = 5$  or  $n = 6$ ?

**200.** [20] Extend exercise 140 to three dimensions. How many base placements do each of the seven Soma pieces have?

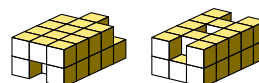
- ▶ **202.** [22] The *Somap* is the graph whose vertices are the 240 distinct solutions to the Soma cube problem, with  $u - v$  if and only if  $u$  can be obtained from  $v$  by changing the positions of at most three pieces. (Using the terminology of exercise 15(d), adjacent vertices correspond to solutions of *semidistance*  $\leq 3$ .) The *strong Somap* is similar, but it has  $u - v$  only when a change of just *two* pieces gets from one to the other.

- What are the degree sequences of these graphs?
- How many connected components do they have? How many bicomponents?

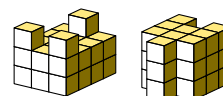
- ▶ **204.** [M25] Use factorization to prove that Fig. 80’s W-wall cannot be built.

**205.** [24] Figure 80(a) shows some of the many “low-rise” (2-level) shapes that can be built from the seven Soma pieces. Which of them is hardest (has the fewest solutions)? Which is easiest? Answer these questions also for the 3-level prism shapes in Fig. 80(b).

- ▶ **206.** [M23] Generalizing the first four examples of Fig. 80, study the set of *all* shapes obtainable by deleting three cubies from a  $3 \times 5 \times 2$  box. (Two examples are shown here.) How many essentially different shapes are possible? Which shape is easiest? Which shape is hardest?



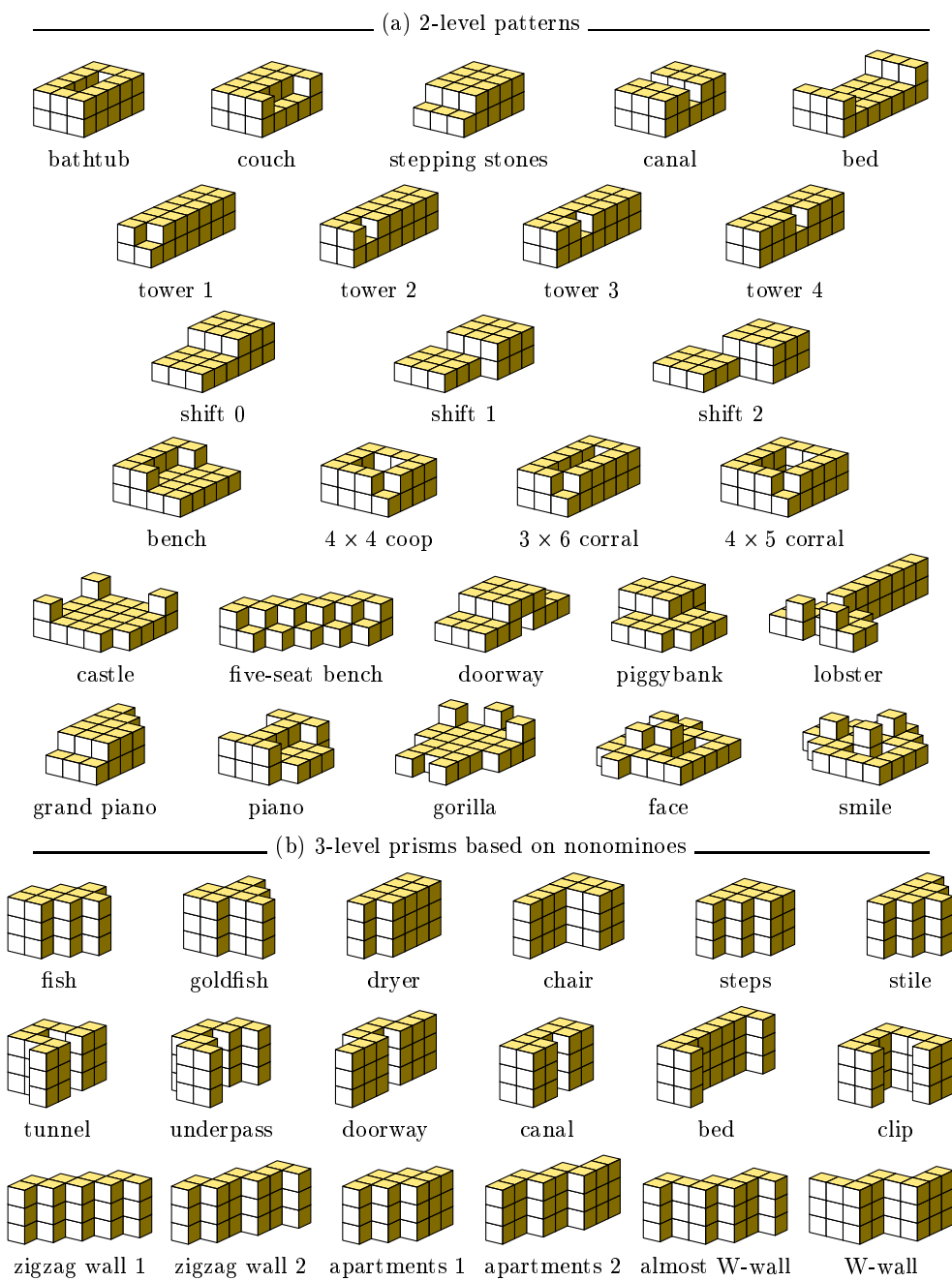
**207.** [22] Similarly, consider (a) all shapes that consist of a  $3 \times 4 \times 3$  box with just three cubies in the top level; (b) all 3-level prisms that fit into a  $3 \times 4 \times 3$  box.



**208.** [25] How many of the 1285 *nonominoes* define a prism that can be realized by the Soma pieces? Do any of those packing problems have a unique solution?

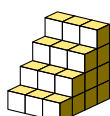
**210.** [M40] Make empirical tests of Piet Hein’s belief that the number of shapes achievable with seven Soma pieces is approximately the number of 27-cubie polycubes.

parallelogram polyomino  
 parallomino  
 skew Young tableau  
 Young tableaux  
 skew Ferrers board  
 Ferrers diagrams  
 tableaux  
 partitions  
 trees  
 path length  
 generating function  
 base placements  
 Somap  
 Soma cube  
 semidistance  
 degree sequences  
 connected components  
 bicomponents  
 factorization  
 W-wall  
 Soma pieces  
 nonominoes  
 Hein



**Fig. 80.** Gallery of noteworthy polycubes that contain 27 cubies. All of them can be built from the seven Soma pieces, except for the W-wall. Many constructions are also stable when tipped on edge and/or when turned upside down. (See exercises 204–214.)

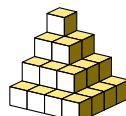
**212.** [20] (B. L. Schwartz, 1969.) Show that the Soma pieces can make shapes that appear to have more than 27 cubies, because of holes hidden inside or at the bottom:



staircase



penthouse



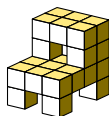
pyramid

In how many ways can these three shapes be constructed?

**213.** [22] Show that the seven Soma pieces can also make structures such as



casserole



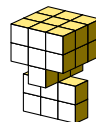
cot



vulture



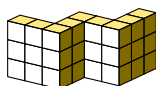
mushroom



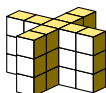
cantilever

which are “self-supporting” via gravity. (You may need to place a small book on top.)

► **214.** [M32] Impossible structures *can* be built, if we insist only that they look genuine when viewed from the front (like façades in Hollywood movies)! Find all solutions to



W-wall



X-wall



cube

that are visually correct. (To solve this exercise, you need to know that the illustrations here use the non-isometric projection  $(x, y, z) \mapsto (30x - 42y, 14x + 10y + 45z)u$  from three dimensions to two, where  $u$  is a scale factor.) All seven Soma pieces must be used.

**215.** [30] The earliest known example of a polycube puzzle is the “Cube Diabolique,” manufactured in late nineteenth century France by Charles Watilliaux; it contains six flat pieces of sizes 2, 3, . . . , 7:



a) In how many ways do these pieces make a  $3 \times 3 \times 3$  cube?

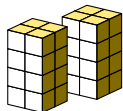
b) Are there six polycubes, of sizes 2, 3, . . . , 7, that make a cube in just *one* way?

**216.** [21] (*The L-bert Hall.*) Take two cubies and drill three holes through each of them; then glue them together and attach a solid cubie and dowel, as shown.



Prove that there’s only one way to pack nine such pieces into a  $3 \times 3 \times 3$  box.

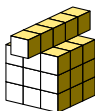
**217.** [22] Show that there are exactly eight different *tetracubes* — polycubes of size 4. Which of the following shapes can they make, respecting gravity? How many solutions are possible?



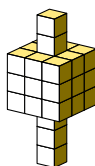
twin towers



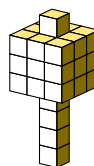
double claw



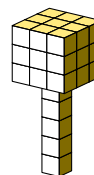
cannon



up 3



up 4



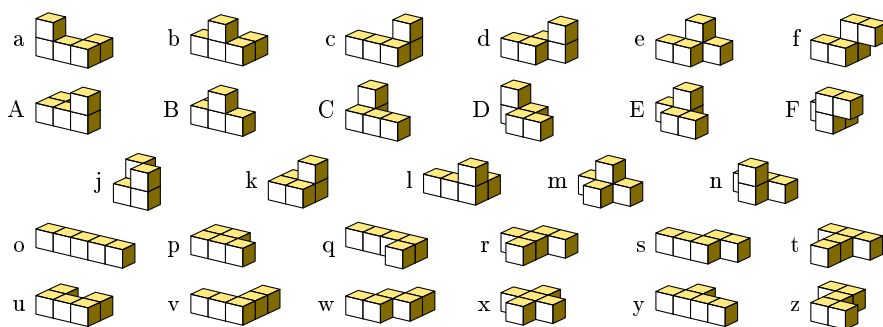
up 5

Schwartz  
self-supporting  
gravity  
façades  
movies  
isometric  
projection  
three dimensions  
Cube Diabolique  
Diabolical Cube  
Watilliaux  
L-bert Hall  
holes  
dowel  
tetracubes  
gravity



**218.** [25] How many of the 369 *octominoes* define a 4-level prism that can be realized by the tetracubes? Do any of those packing problems have a unique solution?

**220.** [30] There are 29 *pentacubes*, conveniently identified with one-letter codes:

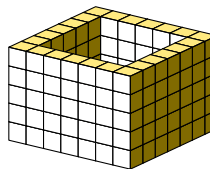


octominoes  
 pentacubes  
 solid pentominoes  
 flat pentacubes  
 mirror images  
 pentominoes  
 $5 \times 5 \times 5$  cube  
 Dowler's Box  
 chiral  
 mirror

Pieces o through z are called, not surprisingly, the *solid pentominoes* or *flat pentacubes*.

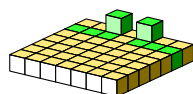
- a) What are the mirror images of a, b, c, d, e, f, A, B, C, D, E, F, j, k, l, . . . , z?
- b) In how many ways can the solid pentominoes be packed into an  $a \times b \times c$  cuboid?
- c) What "natural" set of 25 pentacubes is able to fill the  $5 \times 5 \times 5$  cube?

► **221.** [25] The full set of 29 pentacubes can build an enormous variety of elegant structures, including a particularly stunning example called "Dowler's Box." This  $7 \times 7 \times 5$  container, first considered by R. W. M. Dowler in 1979, is constructed from five flat slabs. Yet only 12 of the pentacubes lie flat; the other 17 must somehow be worked into the edges and corners.

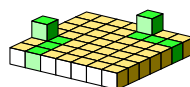


Despite these difficulties, Dowler's Box has so many solutions that we can actually impose many further conditions on its construction:

- a) Build Dowler's Box in such a way that the chiral pieces a, b, c, d, e, f and their images A, B, C, D, E, F all appear in horizontally mirror-symmetric positions.



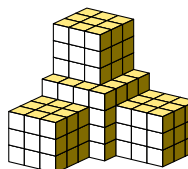
horizontally symmetric c and C



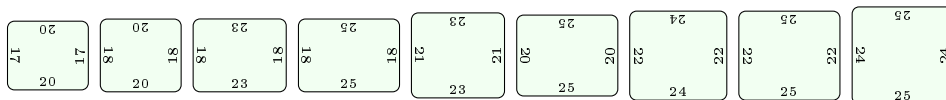
diagonally symmetric c and C

- b) Alternatively, build it so that those pairs are *diagonally* mirror-symmetric.
- c) Alternatively, place piece x in the center, and build the remaining structure from four congruent pieces that have seven pentacubes each.

**222.** [25] The 29 pentacubes can also be used to make the shape shown here, exploiting the curious fact that  $3^4 + 4^3 = 29 \cdot 5$ . But Algorithm D will take a long, long time before telling us how to construct it, unless we're lucky, because the space of possibilities is huge. How can we find a solution quickly?

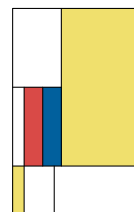


**239.** [29] Nick Baxter devised an innocuous-looking but maddeningly difficult “Square Dissection” puzzle for the International Puzzle Party in 2014, asking that the nine pieces



be placed flat into a  $65 \times 65$  square. One quickly checks that  $17 \times 20 + 18 \times 20 + \dots + 24 \times 25 = 65^2$ ; yet nothing seems to work! Solve his puzzle with the help of Algorithm D.

- ▶ **240.** [20] The next group of exercises is devoted to the decomposition of rectangles into rectangles, as in the Mondrianesque pattern shown here. The *reduction* of such a pattern is obtained by distorting it, if necessary, so that it fits into an  $m \times n$  grid, with each of the vertical coordinates  $\{0, 1, \dots, m\}$  used in at least one horizontal boundary and each of the horizontal coordinates  $\{0, 1, \dots, n\}$  used in at least one vertical boundary. For example, the illustrated pattern reduces to  $\begin{matrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{matrix}$ , where  $m = 3$  and  $n = 5$ . (Notice that the original rectangles needn't have rational width or height.)

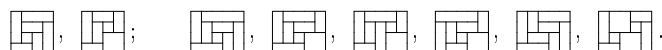


A pattern is called *reduced* if it is equal to its own reduction. Design an exact cover problem by which Algorithm M will discover all of the reduced decompositions of an  $m \times n$  rectangle, given  $m$  and  $n$ . How many of them are possible when  $(m, n) = (3, 5)$ ?

- 241.** [M25] The maximum number of subrectangles in a reduced  $m \times n$  pattern is obviously  $mn$ . What is the *minimum* number?
- 242.** [10] A reduced pattern is called *strictly reduced* if each of its subrectangles  $[a..b] \times [c..d]$  has  $(a, b) \neq (0, m)$  and  $(c, d) \neq (0, n)$  — in other words, if no subrectangle “cuts all the way across.” Modify the construction of exercise 240 so that it produces only strictly reduced solutions. How many  $3 \times 5$  patterns are strictly reduced?
- 243.** [20] A rectangle decomposition is called *faultfree* if it cannot be split into two or more rectangles. For example,  $\begin{matrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{matrix}$  is *not* faultfree, because it has a fault line between rows 2 and 3. (It's easy to see that every reduced faultfree pattern is *strictly* reduced, unless  $m = n = 1$ .) Modify the construction of exercise 240 so that it produces only faultfree solutions. How many reduced  $3 \times 5$  patterns are faultfree?
- 244.** [23] True or false: Every faultfree packing of an  $m \times n$  rectangle by  $1 \times 3$  trominoes is reduced, except in the trivial cases  $(m, n) = (1, 3)$  or  $(3, 1)$ .
- 247.** [22] (*Motley dissections.*) Many of the most interesting decompositions of an  $m \times n$  rectangle involve strictly reduced patterns whose subrectangles  $[a_i..b_i] \times [c_i..d_i]$  satisfy the extra condition

$$(a_i, b_i) \neq (a_j, b_j) \quad \text{and} \quad (c_i, d_i) \neq (c_j, d_j) \quad \text{when } i < j.$$

Thus no two subrectangles are cut off by the same pair of horizontal or vertical lines. The smallest such “motley dissections” are the  $3 \times 3$  pinwheels,  $\begin{matrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{matrix}$  and  $\begin{matrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{matrix}$ , which are considered to be essentially the same because they are mirror images of each other. There are eight essentially distinct motley rectangles of size  $4 \times n$ , namely



The two  $4 \times 4$ s can each be drawn in 8 different ways, under rotations and reflections. Similarly, most of the  $4 \times 5$ s can be drawn in 4 different ways. But the last two have only two forms, because they're symmetric under  $180^\circ$  rotation.

Baxter  
Square Dissection  
rectangles into rectangles  
Mondrian  
reduction  
strictly reduced  
faultfree  
trominoes  
straight trominoes:  $1 \times 3$   
Motley dissections  
pinwheels  
rotations and reflections  
symmetric under  $180^\circ$  rotation

Design an exact cover problem by which Algorithm M will discover all of the motley dissections of an  $m \times n$  rectangle, given  $m$  and  $n$ . (When  $m = n = 4$  the algorithm should find  $8 + 8$  solutions; when  $m = 4$  and  $n = 5$  it should find  $4 + 4 + 4 + 4 + 2 + 2$ .)

- **248.** [25] Improve the construction of the previous exercise by taking advantage of symmetry to cut the number of solutions in half. (When  $m = 4$  there will now be  $4 + 4$  solutions when  $n = 4$ , and  $2 + 2 + 2 + 2 + 1 + 1$  when  $n = 5$ .) *Hint:* A motley dissection is never identical to its left-right reflection, so we needn't visit both.

**249.** [20] The *order* of a motley dissection is the number of subrectangles it has. There are no motley dissections of order six. Show, however, that there are  $m \times m$  motley dissections of order  $2m - 1$  and  $m \times (m + 1)$  motley dissections of order  $2m$ , for all  $m > 3$ .

**250.** [21] An  $m \times n$  motley dissection must have order less than  $\binom{m+1}{2}$ , because only  $\binom{m+1}{2} - 1$  intervals  $[a_i \dots b_i]$  are permitted. What is the maximum order that's actually achievable by an  $m \times n$  motley dissection, for  $m = 5, 6$ , and  $7$ ?

- **252.** [23] Explain how to generate all of the  $m \times n$  motley dissections that have  $180^\circ$ -rotational symmetry, as in the last two examples of exercise 247, by modifying the construction of exercise 248. (In other words, if  $[a \dots b] \times [c \dots d]$  is a subrectangle of the dissection, its complement  $[m - b \dots m - a] \times [n - d \dots n - c]$  must also be one of the subrectangles, possibly the same one.) How many such dissections have size  $8 \times 16$ ?

**253.** [24] Further symmetry is possible when  $m = n$  (as in exercise 247's pinwheel).

- Explain how to generate all of the  $n \times n$  motley dissections that have  $90^\circ$ -rotational symmetry. This means that  $[a \dots b] \times [c \dots d]$  implies  $[c \dots d] \times [n - b \dots n - a]$ .
- Explain how to generate all of the  $n \times n$  motley dissections that are symmetric under reflection about both diagonals. This means that  $[a \dots b] \times [c \dots d]$  implies  $[c \dots d] \times [a \dots b]$  and  $[n - d \dots n - c] \times [n - b \dots n - a]$ , hence  $[n - b \dots n - a] \times [n - d \dots n - c]$ .
- What's the smallest  $n$  for which symmetric solutions of type (b) exist?

**255.** [26] A "perfectly decomposed rectangle" of order  $t$  is a dissection of a rectangle into  $t$  subrectangles  $[a_i \dots b_i] \times [c_i \dots d_i]$  such that the  $2t$  dimensions  $b_1 - a_1, d_1 - c_1, \dots, b_t - a_t, d_t - c_t$  are all distinct. For example, five rectangles of sizes  $1 \times 2, 3 \times 7, 4 \times 6, 5 \times 10$ , and  $8 \times 9$  can be assembled to make the perfectly decomposed  $13 \times 13$  square shown here. What are the *smallest possible* perfectly decomposed squares of orders 5, 6, 7, 8, 9, and 10, having integer dimensions?



**256.** [M28] An "incomparable dissection" of order  $t$  is a decomposition of a rectangle into  $t$  subrectangles, none of which will fit inside another. In other words, if the widths and heights of the subrectangles are respectively  $w_1 \times h_1, \dots, w_t \times h_t$ , we have neither  $(w_i \leq w_j \text{ and } h_i \leq h_j)$  nor  $(w_i \leq h_j \text{ and } h_i \leq w_j)$  when  $i \neq j$ .

- True or false: An incomparable dissection is perfectly decomposed.
- True or false: The reduction of an incomparable dissection is motley.
- True or false: The reduction of an incomparable dissection can't be a pinwheel.
- Prove that every incomparable dissection of order  $\leq 7$  reduces to the first  $4 \times 4$  motley dissection in exercise 247. Furthermore its seven regions can be labeled as shown, with  $w_1 < w_2 < \dots < w_6 < w_7$  and  $h_7 < h_6 < \dots < h_2 < h_1$ .

|   |   |   |   |
|---|---|---|---|
|   |   | 7 |   |
| 2 |   | 6 |   |
|   | 4 |   | 3 |
|   |   | 5 | 1 |

- Suppose the reduction of an incomparable dissection is  $m \times n$ , and suppose its regions have been labeled  $\{1, \dots, t\}$ . Then there are numbers  $x_1, \dots, x_n, y_1, \dots, y_m$  such that the widths are sums of the  $x$ 's and the heights are sums of the  $y$ 's. (For example, in (d) we have  $w_2 = x_1, h_2 = y_1 + y_2 + y_3, w_7 = x_2 + x_3 + x_4, h_7 = y_1$ , etc.) Prove that such a dissection exists with  $w_1 < w_2 < \dots < w_t$  if and only if the

symmetry  
order  
 $180^\circ$ -rotational symmetry  
complement  
pinwheel  
 $90^\circ$ -rotational symmetry  
reflection about both diagonals  
bidiagonal symmetry  
perfectly decomposed rectangle  
incomparable dissection  
motley  
reduction

linear inequalities  $w_1 < w_2 < \dots < w_t$  have a positive solution  $(x_1, \dots, x_n)$  and the linear inequalities  $h_1 > h_2 > \dots > h_t$  have a positive solution  $(y_1, \dots, y_m)$ .

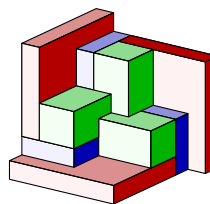
linear inequalities  
Kim

**257.** [M29] Among all the incomparable dissections of order (a) seven and (b) eight, restricted to integer sizes, find the rectangles with smallest possible perimeter. Also find the smallest possible *squares* that have incomparable dissections in integers. *Hint:* Show that there are  $2^t$  potential ways to mix the  $w$ 's with the  $h$ 's, preserving their order; and find the smallest perimeter for each of those cases.

► **258.** [M25] Find seven *different* rectangles of area  $1/7$  that can be assembled into a square of area 1, and prove that the answer is unique.

► **260.** [18] There's a natural way to extend the idea of motley dissection to three dimensions, by subdividing an  $l \times m \times n$  cuboid into subcuboids  $[a_i \dots b_i] \times [c_i \dots d_i] \times [e_i \dots f_i]$  that have no repeated intervals  $[a_i \dots b_i]$  or  $[c_i \dots d_i]$  or  $[e_i \dots f_i]$ .

For example, Scott Kim has discovered a remarkable motley  $7 \times 7 \times 7$  cube consisting of 23 individual blocks, 11 of which are illustrated here. (Two of them are hidden behind the others.) The full cube is obtained by suitably placing a mirror image of these pieces in front, together with a  $1 \times 1 \times 1$  cubie in the center.



By studying this picture, show that Kim's construction can be defined by coordinate intervals  $[a_i \dots b_i] \times [c_i \dots d_i] \times [e_i \dots f_i]$ , with  $0 \leq a_i, b_i, c_i, d_i, e_i, f_i \leq 7$  for  $1 \leq i \leq 23$ , in such a way that the pattern is symmetrical under the transformation  $xyz \mapsto \bar{y}\bar{z}\bar{x}$ . In other words, if  $[a \dots b] \times [c \dots d] \times [e \dots f]$  is one of the subcuboids, so is  $[7 - d \dots 7 - c] \times [7 - f \dots 7 - e] \times [7 - b \dots 7 - a]$ .

**261.** [29] Use exercise 260 to construct a perfectly decomposed  $108 \times 108 \times 108$  cube, consisting of 23 subcuboids that have 69 distinct integer dimensions. (See exercise 256.)

**262.** [24] By generalizing exercises 247 and 248, explain how to find *every* dissection of an  $l \times m \times n$  cuboid, using Algorithm M. *Note:* In three dimensions, the strictness condition ' $(a_i, b_i) \neq (0, m)$  and  $(c_i, d_i) \neq (0, n)$ ' of exercise 242 should become

$$[(a_i, b_i) = (0, l)] + [(c_i, d_i) = (0, m)] + [(e_i, f_i) = (0, n)] \leq 1.$$

What are the results when  $l = m = n = 7$ ?

**263.** [M46] Do motley cuboids of size  $l \times m \times n$  exist only when  $l = m = n = 7$ ?

**999.** [M00] this is a temporary exercise (for dummies)

*Dr Pell was wont to say, that in the Resolution of Questiones, the main matter is the well stating them: which requires a good mother-witt & Logick: as well as Algebra: for let the Question be but well-stated, and it will worke of it selfe: ... By this way, an man cannot intangle his notions, & make a false Steppe.*  
 — JOHN AUBREY, *An Idea of Education of Young Gentlemen* (c. 1684)

Pell  
 AUBREY  
 second death  
 semidistance  
 Matsui  
 Matsui  
 NP-complete

**SECTION 7.2.2.1**

5. If  $T$  has only a root node, let there be one column, no rows. Otherwise let  $T$  have  $d \geq 1$  subtrees  $T_1, \dots, T_d$ , and assume that we've constructed problems with rows  $R_j$  and columns  $C_j$  for each  $T_j$ . Let  $C = C_1 \cup \dots \cup C_d \cup \{1, \dots, d\}$ . The problem for  $T$  is obtained by appending  $d+1$  new columns  $\{0, 1, \dots, d\}$  and the following new rows:  
 (i) '0 and all columns of  $C \setminus C_j$ ', for  $1 \leq j \leq d$ ; (ii) 'all columns of  $C \setminus j$ ', for  $1 \leq j \leq d$ . This construction works except when  $d = 1$  and  $T_1$  is a leaf; in that case we can use columns  $\{0, 1, 2, 3\}$ , rows '0 1 2', '1 3', '2 3'. The matrix for the example tree has 17 columns and 16 rows.

```
01111110000000000
10111110000000000
11011110000000000
11100110000000000
11101010000000000
11100110000000000
00000000111111000
00000001011111000
00000001101111000
00000001110011000
00000001110101000
00000001110110000
00000001111111111
11111110000000111
11111111111111001
1111111111111010
```

6. (a) If a solution isn't at the root, its parent must have exactly one child. (Alternatively, if duplicate rows are permitted, all siblings of a solution must be solutions.)  
 (b) Use the previous construction; a solution node corresponds to column 0, row '0'.

10. Use PREV and NEXT to cyclically link all uncovered secondary columns. Then, when all primary columns have been covered, accept a solution only if  $LEN(ND[c]) = 0$  for all columns  $c$  on that list. [This algorithm is called the "second death" method, because it checks that all of the purely secondary rows have been killed off by primary covering.]

15. (a) No. Otherwise  $A$  would have a row that's zero in all primary columns.  
 (b) Yes, but only if  $A$  has two rows that are identical in all primary columns.  
 (c) Yes, but only if  $A$  has two rows whose sum is also a row, when restricted to primary columns.

(d) The number of places,  $j$ , where  $x = 1$  and  $x' = 0$  must be the same as the number where  $x = 0$  and  $x' = 1$ . For if  $A$  has exactly  $k$  primary 1s in every row, exactly  $jk$  primary columns are being covered in different ways.

(e) Again the distances must be even, because every solution to  $A$  is also a solution to the uniform problem  $A|C$ . (Therefore it makes sense to speak of the *semidistance*  $d(x, x')/2$  between solutions of a quasi-uniform exact covering problem. The semidistance in a polyform packing problem is the number of pieces that are packed differently.)

19. (Solution by T. Matsui.) Add one new column at the left of  $A$ , all 0s. Then add two rows of length  $n + 1$  at the bottom:  $10 \dots 0$  and  $11 \dots 1$ . This  $(m + 2) \times (n + 1)$  matrix  $A'$  has one solution that chooses only the last row. All other solutions choose the second-to-last row, together with rows that solve  $A$ .

20. (Solution by T. Matsui.) Assume that all 1s in column 1 appear in the first  $t$  rows, where  $t > 3$ . Add two new columns at the left, and two new rows  $1100 \dots 0, 1010 \dots 0$  of length  $n + 2$  at the bottom. For  $1 \leq k \leq t$ , if row  $k$  was  $1\alpha_k$ , replace it by  $010\alpha_k$  if  $k \leq t/2, 011\alpha_k$  if  $k > t/2$ . Insert  $00$  at the left of the remaining rows  $t + 1$  through  $m$ .

This construction can be repeated (with suitable row and column permutations) until no column sum exceeds 3. If the original column sums were  $(c_1, \dots, c_n)$ , the new  $A'$  has  $2T$  more rows and  $2T$  more columns than  $A$  did, where  $T = \sum_{j=1}^n (c_j \div 3)$ .

One consequence is that the exact cover problem is NP-complete even when restricted to cases where all row and column sums are at most 3.

Notice, however, that this construction is *not* useful in practice, because it disguises the structure of  $A$ : It essentially *destroys* the minimum remaining values heuristic, because all columns whose sum is 2 look equally good to the solver!

**21.** Take a matrix with column sums  $(c_1, \dots, c_n)$ , all  $\leq 3$ , and extend it with three columns of 0s at the right. Then add the following four rows:  $(x_1, \dots, x_n, 0, 1, 1)$ ,  $(y_1, \dots, y_n, 1, 0, 1)$ ,  $(z_1, \dots, z_n, 1, 1, 0)$ , and  $(0, \dots, 0, 1, 1, 1)$ , where  $x_j = [c_j < 3]$ ,  $y_j = [c_j < 2]$ ,  $z_j = [c_j < 1]$ . The bottom row must be chosen in any solution.

**24.** Consider a set of cubes and colors called  $\{*, 0, 1, 2, 3, 4, \dots\}$ , where (i) all faces of cube  $*$  are colored  $*$ ; (ii) colors 1, 2, 3, 4 occur only on cubes 0, 1, 2, 3, 4; (iii) the opposite face-pairs of those five cubes are respectively  $(00, 12, **)$ ,  $(11, 12, 34)$ ,  $(22, 34, \alpha)$ ,  $(33, 12, \beta)$ ,  $(44, 34, \gamma)$ , where  $\alpha, \beta, \gamma$  are pairs of colors  $\notin \{1, 2, 3, 4\}$ . Any solution to the cube problem has disjoint 2-regular graphs  $X$  and  $Y$  containing two faces of each color. Since  $X$  and  $Y$  both contain  $**$  from cube  $*$ , we can assume that  $X$  contains 00 and  $Y$  contains 12 from cube 0. Hence  $Y$  can't contain 11 or 22; it must contain 12 from cube 1 or cube 3. If  $X$  doesn't contain 11 or 22, it must contain 12 from cube 1 and cube 3. Hence  $X$  contains 11, 22, 33, and 44. We're left with only three possibilities for  $Y$  from cubes 1, 2, 3, 4, namely  $(34, \alpha, 12, 34)$ ,  $(12, 34, \beta, 34)$ ,  $(34, 34, 12, \gamma)$ .

Now let  $a_{j1}, a_{j2}, a_{j3}$  denote the 1s in column  $j$  of  $A$ . We construct  $N = 8n + 1$  cubes and colors called  $*$ ,  $a_{jk}, b_{jl}$ , where  $1 \leq j \leq n$ ,  $1 \leq k \leq 3$ ,  $0 \leq l \leq 4$ . The opposite face-pairs of  $*$  are  $(**, **, **)$ . Those of  $a_{jk}$  are  $(a_{jk}a_{jk}, a_{jk}a_{jk}, a_{jk}b_{j'0})$ , where  $j'$  is the column of  $a_{jk}$ 's cyclic successor to the right in its row. Those of  $b_{j0}, b_{j1}, b_{j2}, b_{j3}, b_{j4}$  are respectively  $(b_{j0}b_{j0}, b_{j1}b_{j2}, **)$ ,  $(b_{j1}b_{j1}, b_{j1}b_{j2}, b_{j3}b_{j4})$ ,  $(b_{j2}b_{j2}, b_{j3}b_{j4}, b_{j0}a_{j1})$ ,  $(b_{j3}b_{j3}, b_{j1}b_{j2}, b_{j0}a_{j2})$ ,  $(b_{j4}b_{j4}, b_{j3}b_{j4}, b_{j0}a_{j3})$ . By the previous paragraph, solutions to the cube problem correspond to 2-regular graphs  $X$  and  $Y$  such that, for each  $j$ ,  $X$  or  $Y$  contains all the pairs  $b_{jl}b_{jl}$  and the other "selects" one of the three pairs  $b_{j0}a_{jk}$ . The face-pairs of each selected  $a_{jk}$  ensure that  $a_{jk}$ 's cyclic successor is also selected.

[See E. Robertson and I. Munro, *Utilitas Mathematica* **13** (1978), 99–116.]

**26.** (a)  $(x \circ y) \circ x = (x \circ y) \circ (y \circ (x \circ y)) = y$ .

(b) All five are legitimate. (The last two are gropes because  $f(t + f(t)) = t$  for  $0 \leq t < 4$  in each case. They are isomorphic if we interchange any two elements. The third is isomorphic to the second if we interchange  $1 \leftrightarrow 2$ . There are 18 grope tables of order 4, of which  $(4, 12, 2)$  are isomorphic to the first, third, and last tables shown here.)

(c) For example, let  $x \circ y = (-x - y) \bmod n$ . (More generally, if  $G$  is any group and if  $\alpha \in G$  satisfies  $\alpha^2 = 1$ , we can let  $x \circ y = \alpha x^{-1} \alpha y^{-1} \alpha$ . If  $G$  is commutative and  $\alpha \in G$  is arbitrary, we can let  $x \circ y = x^{-1} y^{-1} \alpha$ .)

(d) For each row of type (i) in an exact covering, define  $x \circ x = x$ ; for each row of type (ii), define  $x \circ x = y$ ,  $x \circ y = y \circ x = x$ ; for each row of type (iii), define  $x \circ y = z$ ,  $y \circ z = x$ ,  $z \circ x = y$ . Conversely, every grope table yields an exact covering in this way.

(e) Such a grope covers  $n^2$  columns with  $k$  rows of size 1, all other rows of size 3. [F. E. Bennett proved, in *Discrete Mathematics* **24** (1978), 139–146, that such gropes exist for *all*  $k$  with  $0 \leq k \leq n$  and  $k \equiv n^2 \pmod{3}$ , except when  $k = n = 6$ .]

*Notes:* The identity  $x \circ (y \circ x) = y$  seems to have first been considered by E. Schröder in *Math. Annalen* **10** (1876), 289–317 [see '(C<sub>0</sub>)' on page 306], but he didn't do much with it. In a class for sophomore mathematics majors at Caltech in 1968, the author defined gropes and asked the students to discover and prove as many theorems about them as they could, by analogy with the theory of groups. The idea was to "grope for results." The official modern term for a grope is a real jawbreaker: *semisymmetric quasigroup*.

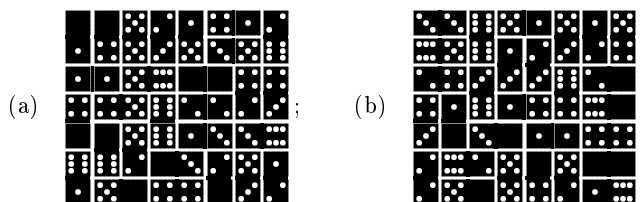
minimum remaining values heuristic  
2-regular graphs  
Robertson  
Munro  
isomorphic  
isomorphic  
Bennett  
Caltech  
author  
groups  
quasigroup  
semisymmetric quasigroup

**27.** (a) Eliminate the  $n$  columns for  $(x, x)$ ; use only the  $2\binom{n}{3}$  rows of type (iii) for which  $y \neq z$ . (Idempotent gropes are equivalent to “Mendelsohn triples,” which are families of  $n(n-1)/3$  three-cycles  $(xyz)$  that include every ordered pair of distinct elements. N. S. Mendelsohn proved [*Computers in Number Theory* (New York: Academic Press, 1971), 323–338] that such systems exist for all  $n \not\equiv 2 \pmod{3}$ , except when  $n = 6$ .)

(b) Use only the  $\binom{n+1}{2}$  columns  $(x, y)$  for  $0 \leq x \leq y < n$ ; replace rows of type (ii) by  $\{(x, x), (x, y)\}$  and  $\{(x, y), (y, y)\}$  for  $0 \leq x < y < n$ ; replace those of type (iii) by  $\{(x, y), (x, z), (y, z)\}$  for  $0 \leq x < y < z < n$ . (Such systems, Schröder’s ‘ $(C_1)$  and ‘ $(C_2)$ ’, are called totally symmetric quasigroups; see S. K. Stein, *Trans. Amer. Math. Soc.* **85** (1957), 228–256, §8. If idempotent, they’re equivalent to Steiner triple systems.)

(c) Omit columns for which  $x = 0$  or  $y = 0$ . Use only the  $2\binom{n-1}{3}$  rows of type (iii) for  $1 \leq x < y, z < n$  and  $y \neq z$ . (Indeed, such systems are equivalent to idempotent gropes on the elements  $\{1, \dots, n-1\}$ .)

**30.** In (a), four pieces change; in (b) the solution is unique:



Notice that the spot patterns  $\blacksquare$ ,  $\blacksquare$ , and  $\blacksquare$  are rotated when a domino is placed vertically; these visual clues, which would disambiguate (a), don’t show up in the matrix.

[Dominosa was invented in Germany by O. S. Adler [Reichs Patent #71539 (1893); see his booklet written with F. Jahn, *Sperr-Domino und Dominosa* (1912), 23–64. Similar problems of “quadrilles” had been studied earlier by E. Lucas and H. Delannoy; see Lucas’s [*Récréations Mathématiques 2* (Paris: Gauthier-Villars, 1883), 52–63].

**31.** Define 28 vertices  $Dxy$  for  $0 \leq x \leq y \leq 6$ ; 28 vertices  $ij$  for  $0 \leq i < 7, 0 \leq j < 8$ , and  $i + j$  even; and 28 similar vertices  $ij$  with  $i + j$  odd. The matching problem has 49 triples of the form  $\{Dxy, ij, i(j+1)\}$  for  $0 \leq i, j < 7$ , as well as 48 of the form  $\{Dxy, ij, (i+1)j\}$  for  $0 \leq i < 6$  and  $0 \leq j < 8$ , corresponding to potential horizontal or vertical placements. For example, the triples for exercise 30(a) are  $\{D00, 00, 01\}$ ,  $\{D05, 01, 02\}$ ,  $\dots$ ,  $\{D23, 66, 67\}$ ;  $\{D01, 00, 10\}$ ,  $\{D04, 01, 11\}$ ,  $\dots$ ,  $\{D12, 57, 67\}$ .

**32.** Model (i) has  $M = 56!/8!^7 \approx 4.10 \times 10^{42}$  equally likely possibilities; model (ii) has  $N = 1292697 \cdot 28! \cdot 2^{21} \approx 8.27 \times 10^{41}$ , because there are 1292697 ways to pack 28 dominoes in a  $7 \times 8$  frame. (Algorithm D will quickly list them all.) The expected number of solutions per trial in model (i) is therefore  $N/M \approx 0.201$ .

Ten thousand random trials with model (i) gave 216 cases with at least one solution, including 26 where the solution was unique. The total number  $\sum x$  of solutions was 2256; and  $\sum x^2 = 95918$  indicated a heavy-tailed distribution whose empirical standard deviation is  $\approx 3.1$ . The total running time was about 250  $M\mu$ .

Ten thousand random trials with model (ii), using random choices from a precomputed list of 1292687 packings, gave 106 cases with a unique solution; one case had 2652 of them! Here  $\sum x = 508506$  and  $\sum x^2 = 144119964$  indicated an empirical mean of  $\approx 51$  solutions per trial, with standard deviation  $\approx 109$ . Total time was about 650  $M\mu$ .

Mendelsohn triples  
Schröder  
totally symmetric quasigroups  
Stein  
Steiner triple systems  
Adler  
Jahn  
quadrilles  
Lucas  
Delannoy  
dimer tilings  
heavy-tailed distribution  
empirical standard deviation

**39.** Each of the 92 solutions to the eight queens problem (see Fig. 68) occupies eight of the 64 cells; so we must find eight disjoint solutions. Only 1897 updates of Algorithm D are needed to show that such a mission is impossible. [In fact no *seven* solutions can be disjoint, because each solution touches at least three of the twenty cells 13, 14, 15, 16, 22, 27, 31, 38, 41, 48, 51, 58, 61, 68, 72, 77, 83, 84, 85, 86. See Thorold Gosset, *Messenger of Mathematics* **44** (1914), 48. Henry E. Dudeney found the illustrated way to occupy all but two cells, in *Tit-Bits* **32** (11 September 1897), 439; **33** (2 October 1897), 3.]

**40.** This is an exact cover problem with  $92 + 312 + 396 + \dots + 312 = 3284$  rows (see exercise 7.2.2-6). Algorithm D needs about 2 million updates to find the solution shown, and about 83 billion to find all 11,092 of them.

**50.** Set  $f_m \leftarrow 0$  and  $f_{k-1} \leftarrow f_k \mid r_k$  for  $m \geq k > 1$ . The bits of  $u_k$  represent columns that are being changed for the last time.

Let  $u_k = u' + u''$ , where  $u' = u_k \& p$ . If  $u_k \neq 0$  at the beginning of step N4, we compress the database as follows: For  $N \geq j \geq 1$ , if  $s_j \& u' \neq u'$ , delete  $(s_j, c_j)$ ; otherwise if  $s_j \& u'' \neq 0$ , delete  $(s_j, c_j)$  and insert  $((s_j \& \bar{u}_k) \mid u', c_j)$ .

To delete  $(s_j, c_j)$ , set  $(s_j, c_j) \leftarrow (s_N, c_N)$  and  $N \leftarrow N - 1$ .

When this improved algorithm terminates in step N2, we always have  $N \leq 1$ . Furthermore, if we let  $p_k = r_1 \mid \dots \mid r_{k-1}$ , the size of  $N$  never exceeds  $2^{\nu_k}$ , where  $\nu_k = \nu(p_k r_k f_k)$  is the size of the “frontier” (see exercise 7.1.4-55).

[In the special case of  $n$  queens, represented as the exact cover problem in  $(\star\star)$ , this algorithm is due to I. Rivin, R. Zabih, and J. Lamping, *Inf. Proc. Letters* **41** (1992), 253–256. They proved that the frontier for  $n$  queens never has more than  $3n$  columns.]

**51.** The author has had reasonably good results using a triply linked binary search tree for the database, with randomized search keys. (Beware: The swapping algorithm used for deletion was difficult to get right.) This implementation was, however, limited to exact cover problems whose matrix has at most 64 columns; hence it could do  $n$  queens via  $(\star\star)$  only when  $n < 12$ . When  $n = 11$  its database reached a maximum size of 75,009, and its running time was about 25 megamems. But Algorithm D was a lot better: It needed only about 780K updates to find all  $Q(11) = 2680$  solutions.

In theory, this method will need only about  $2^{3n}$  steps as  $n \rightarrow \infty$ , times a small polynomial function of  $n$ . A backtracking algorithm such as Algorithm D, which enumerates each solution explicitly, will probably run asymptotically slower (see exercise 7.2.2-15). But in practice, a breadth-first approach needs too much space.

On the other hand, this method did beat Algorithm D on the  $n$  queen bees problem of exercise 7.2.2-16: When  $n = 11$  its database grew to 364,864 items; it computed  $H(11) = 596,483$  in just 30 M $\mu$ , while Algorithm D needed 27 mega-updates.

**52.** The set of solutions for  $s_j$  can be represented as a regular expression  $\alpha_j$  instead of by its size,  $c_j$ . Instead of inserting  $(s_j + t, c_j)$  in step N3, insert  $\alpha_j k$ . If inserting  $(s, \alpha)$ , when  $(s_i, \alpha_i)$  is already present with  $s_i = s$ , change  $\alpha_i \leftarrow \alpha_i \cup \alpha$ . [Alternatively, if only one solution is desired, we could attach a single solution to each  $s_j$  in the database.]

**58.** After uncovering all other columns of CURNODE at level 0, let  $p$  point to the node at the right of CURNODE's row. If  $p \geq \text{SECOND}$ , cover  $\text{COL}(\text{ND}[p])$ . (This extension applies also to Algorithm C, but one should ensure first that  $\text{COLOR}(\text{ND}[p]) = 0$ .)

**60.** Let CUTOFF (initially  $\infty$ ) point to the spacer at the end of the best solution found so far. We'll essentially remove all nodes  $> \text{CUTOFF}$  from further consideration.

Whenever a solution is found, let node PP be the spacer at the end of the option for which  $\text{CHOICE}[k]$  is maximum. If  $\text{PP} \neq \text{CUTOFF}$ , set  $\text{CUTOFF} \leftarrow \text{PP}$ , and for  $0 \leq k \leq \text{LEVEL}$

eight queens problem  
Gosset  
Dudeney  
frontier  
 $n$  queens  
Rivin  
Zabih  
Lamping  
author  
triply linked  
binary search tree  
backtracking algorithm  
asymptotically  
theory vs practice  
practice vs theory  
 $n$  queen bees  
regular expression  
CUTOFF



remove all options  $> \text{CUTOFF}$  from the list for  $\text{COL}(\text{ND}(\text{CHOICE}[k]))$ . (It's easy to do this because the list is sorted.) Minimax solutions follow the last change to  $\text{CUTOFF}$ .

Begin the subroutine 'uncover( $c$ )' by removing all options  $> \text{CUTOFF}$  from column  $c$ 's list. After setting  $\text{DD} \leftarrow \text{DOWN}(\text{ND}(\text{NN}))$  in that routine, set  $\text{DOWN}(\text{ND}(\text{NN})) \leftarrow \text{DD} \leftarrow \text{CC}$  if  $\text{DD} > \text{CUTOFF}$ . Make the same modifications also to the subroutine 'unpurify( $p$ )'.

*Subtle point:* Suppose we're uncovering column  $c$  and encounter an option ' $c x \dots$ ' that should be restored to column  $x$ ; and suppose that the original successor ' $x a \dots$ ' of that option in column  $x$  lies below the cutoff. We know that ' $x a \dots$ ' contains at least one primary column, and that every primary column was covered before we changed the cutoff. Hence ' $x a \dots$ ' was *not* restored, and we needn't worry about removing it. We merely need to correct the  $\text{DOWN}$  link, as stated above.

**61.** Now let  $\text{CUTOFF}$  be the spacer just *before* the best solution known. When resetting  $\text{CUTOFF}$ , backtrack to level  $k - 1$ , where  $k$  maximizes  $\text{CHOICE}[k]$ .

**64.** Use  $2n$  primary columns  $a_i, d_j$  for the "across" and "down" words, together with  $n^2$  secondary columns  $ij$  for the individual cells. Also use  $M$  secondary columns  $w$ , one for each legal word. The cover problem has  $2Mn$  rows, namely ' $a_i i1:c_1 \dots in:c_n c_1 \dots c_n$ ' and ' $d_j 1j:c_1 \dots nj:c_n c_1 \dots c_n$ ' for  $1 \leq i, j \leq n$  and each legal word  $c_1 \dots c_n$ .

We can avoid having both a solution and its transpose by introducing  $M$  further secondary columns  $w'$  and appending  $c_1 \dots c_n'$  at the right of each option for  $a_1$  and  $d_1$ . Then exercise 58's variant of Algorithm C will never choose a word for  $d_1$  that it has already tried for  $a_1$ . (Think about it.)

But this construction is *not* a win for "dancing links," because it causes massive amounts of data to go in and out of the active structure. For example, with the five-letter words of  $\text{WORDS}(5757)$ , it correctly finds all 323,264 of the double word squares but its running time is 15 *teramems*! Much faster is to use the algorithm of exercise 7.2.2-28, which needs only 46 gigamems to discover all of the 1,787,056 unrestricted word squares; the double word squares are easily identified among those solutions.

**65.** One could do a binary search, trying varying values of  $N$ . But the best way is to use the construction of exercise 64 together with the minimax variant of Algorithm C in exercise 60. This works perfectly, when the options for most common words come first.

Indeed, this method finds the double square 'BLAST|EARTH|ANGER|SCOPE|TENSE' and proves it best in just 64  $G\mu$ , almost as fast as the specialized method of exercise 7.2.2-28. (That square contains ARGON, the 1720th most common five-letter word, in its third column; the next-best squares use PEERS, which has rank 1800.)

**66.** The "minimax" method of exercise 65 finds the first five squares of

|     |       |         |           |             |               |               |
|-----|-------|---------|-----------|-------------|---------------|---------------|
|     |       |         |           |             | C H E S T S   | H E R T Z E S |
|     |       |         | S T A R T | L U S T R E | O P E R A T E |               |
|     | M A Y | S H O W | T H R E E | O B T A I N | M I M I C A L |               |
| I S | A G E | N O N E | R O O F S | A R E N A S | A C E R A T E |               |
| T O | N O T | O P E N | A S S E T | C I R C L E | G E N E T I C |               |
|     |       | W E S T | P E E R S | A S S E S S | E N D M O S T |               |
|     |       |         |           |             | R E S E N T S |               |

in respectively 200  $K\mu$ , 15  $M\mu$ , 450  $M\mu$ , 25  $G\mu$ , 25.6  $T\mu$ . It struggles to find the best  $6 \times 6$ , because too few words are cut off from the search; and it thrashes miserably with the 24 thousand 7-letter words, because those words yield only seven extremely esoteric solutions. For those lengths it's best to cull the 2038753 and 14513 *unrestricted* word squares, which the method of exercise 7.2.2-28 finds in respectively 4.6  $T\mu$  and 8.7  $T\mu$ .

**68.** An exact cover problem with colors, as in answer 64, works nicely: There are  $2p$  primary columns  $a_i$  and  $d_i$  for the final words, and  $pn + M$  secondary columns

sorted  
binary search  
minimax  
minimax

$ij$  and  $w$  for the cells and potential words, where  $0 \leq i < p$  and  $1 \leq j \leq n$ . The  $Mp$  rows going across are ' $a_i i1:c_1 i2:c_2 \dots in:c_n c_1 \dots c_n$ '. The  $Mp$  rows going down are ' $d_i i1:c_1 ((i+1) \bmod p)2:c_2 \dots ((i+n-1) \bmod p)n:c_n c_1 \dots c_n$ ' for left-leaning stairs; ' $d_i i1:c_n ((i+1) \bmod p)2:c_{n-1} \dots ((i+n-1) \bmod p)n:c_1 c_1 \dots c_n$ ' for right-leaning stairs. The modification to Algorithm C in exercise 58 saves a factor of  $2p$ ; and the minimax modification in exercise 50 hones in quickly on optimum solutions.

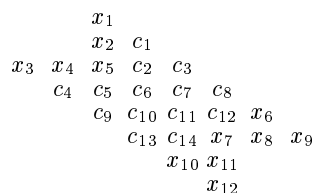
minimax  
Flower Power  
Shortz  
Incroci Concentrici  
Petal Pushers  
kernels  
induced subgraph  
in-degree  
out-degree

There are no left word stairs for  $p = 1$ , since we need two distinct words. The left winners for  $2 \leq p \leq 10$  are: 'WRITE|WHOLE'; 'MAKES|LIVED|WAXES'; 'THERE|SHARE|WHOLE|WHOSE'; 'STOOD|THANK|SHARE|SHIPS|STORE'; 'WHERE|SHEEP|SMALL|STILL|WHOLE|SHARE'; 'MAKES|BASED|TIRED|WORKS|LANDS|LIVES|GIVES'; 'WATER|MAKES|LOVED|GIVES|LAKES|BASED|NOTES|BONES'; 'WHERE|SHEET|STILL|SHALL|WHITE|SHAPE|STARS|WHILE|SHORE'; 'THERE|SHOES|SHIRT|STONE|SHOOK|START|WHILE|SHELL|STEEL|SHARP'. They all belong to WORDS(500), except that  $p = 8$  needs WORDS(504) for NOTED.

The right winners have a bit more variety: 'SPOTS'; 'STALL|SPIES'; 'STOOD|HOLES|LEAPS'; 'MIXED|TEARS|SLEPT|SALAD'; 'YEARS|STEAM|SALES|MARKS|DRIED'; 'STEPS|SEALS|DRAWS|KNOTS|TRAPS|DROPS'; 'TRIED|FEARS|SLIPS|SEAMS|DRAWS|ERECT|TEARS'; 'YEARS|STOPS|HOOKS|FRIED|TEARS|SLANT|SWORD|SWEEP'; 'START|SPEAR|SALES|TESTS|STEER|SPEAK|SKIES|SLEPT|SPORT'; 'YEARS|STOCK|HORNS|FUELS|BEETS|SPEED|TEARS|PLANT|SWORD|SWEEP'. They belong to WORDS(1300) except when  $p$  is 2 or 3.

[Arrangements equivalent to left word stairs were introduced in America under the name "Flower Power" by Will Shortz in *Classic Crossword Puzzles* (Penny Press, February 1976), based on Italian puzzles called "Incroci Concentrici" in *La Settimana Enigmistica*. Shortly thereafter, in *GAMES* magazine and with  $p = 16$ , he called them "Petal Pushers," usually based on six-letter words but occasionally going to seven. Left word stairs are much more common than the right-leaning variety, because the latter mix end-of-word with beginning-of-word letter statistics.]

**69.** Consider all "kernels"  $c_1 \dots c_{14}$  that can appear as illustrated, within a right word stair of 5-letter words. Such kernels arise for a given set of words only if there are letters  $x_1 \dots x_{12}$  such that  $x_3x_4x_5c_2c_3, c_4c_5c_6c_7c_8, c_9c_{10}c_{11}c_{12}x_6, c_{13}c_{14}x_7x_8x_9, x_1x_2x_5c_5c_9, c_1c_2c_6c_{10}c_{13}, c_3c_7c_{11}c_{14}x_{10}$ , and  $c_8c_{12}x_7x_{11}x_{12}$  are all in the set. Thus it's an easy matter to set up an exact cover problem (with colors) that will find the multiset of kernels, after which we can extract the set of *distinct* kernels.



Construct the digraph whose arcs are the kernels, and whose vertices are the 9-tuples that arise when kernel  $c_1 \dots c_{14}$  is regarded as the transition

$$c_1c_2c_3c_4c_5c_6c_7c_9c_{10} \rightarrow c_3c_7c_8c_9c_{10}c_{11}c_{12}c_{13}c_{14}.$$

This transition contributes two words,  $c_4c_5c_6c_7c_8$  and  $c_1c_2c_6c_{10}c_{13}$ , to the word stair. Indeed, *right word stairs of period  $p$  are precisely the  $p$ -cycles in this digraph for which the  $2p$  contributed words are distinct.*

Now we can solve the problem, if the graph isn't too big. For example, WORDS(1000) leads to a digraph with 180524 arcs and 96677 vertices. We're interested only in the oriented cycles of this (very sparse) digraph; so we can reduce it drastically by looking only at the largest induced subgraph for which each vertex has positive in-degree and positive out-degree. (See exercise 7.1.4-234, where a similar reduction was made.) And wow: That subgraph has only 30 vertices and 34 arcs! So it is totally understandable, and we deduce quickly that the longest right word stair belonging to WORDS(1000) has

$p = 5$ . That word stair, which we found directly in answer 68, corresponds to the cycle

SEDYEARST  $\rightarrow$  DRSTEASA  $\rightarrow$  SAMSALEMA  $\rightarrow$  MESMARKDR  $\rightarrow$  SKSDRIEYE  $\rightarrow$  SEDYEARST.

A similar approach applies to left word stairs, but the kernel configurations are reflected left-to-right; transitions then contribute the words  $c_8c_7c_6c_5c_4$  and  $c_1c_2c_6c_{10}c_{13}$ . The digraph from WORDS(500) turns out to have 136771 arcs and 74568 vertices; but this time 6280 vertices and 13677 arcs remain after reduction. Decomposition into strong components makes the task simpler, because very cycle belongs to a strong component. Still, we're stuck with a giant component that has 6150 vertices and 12050 arcs.

The solution is to reduce the current subgraph repeatedly as follows: Find a vertex  $v$  of out-degree 1. Backtrack to discover a simple path, from  $v$ , that contributes only distinct words. If there is no such path (and there usually isn't, and the search usually terminates quickly), remove  $v$  from the graph and reduce it again.

With this method one can rapidly show that the longest left word stair from WORDS(500) has period length 36: 'SHARE|SPENT|SPEED|WHEAT|THANK|CHILD|SHELL|SHORE|STORE|STOOD|CHART|GLORY|FLOWS|CLASS|NOISE|GAMES|TIMES|MOVES|BONES|WAVES|GASES|FIXED|TIRED|FEELS|FALLS|WORLD|ROOMS|WORDS|DOORS|PARTY|WANTS|WHICH|WHERE|SHOES|STILL|STATE', with 36 other words that go down. Incidentally, GLORY and FLOWS have ranks 496 and 498, so they just barely made it into WORDS(500).

Larger values of  $N$  are likely to lead to quite long cycles from WORDS( $N$ ). Their discovery won't be easy, but the search will no doubt be instructive.

**70.** Use  $3p$  primary columns  $a_i, b_i, d_i$  for the final words;  $pn + 2M$  secondary columns  $ij, w, w'$  for the cells and potential words, with  $0 \leq i < p$  and  $1 \leq j \leq n$  (somewhat as in answer 68). The  $Mp$  rows going across are ' $a_i i1:c_1 i2:c_2 \dots in:c_n c_1 \dots c_n c_1 \dots c_n$ '. The  $2Mp$  rows going down in each way are ' $b_i i1:c_1 ((i+1) \bmod p)2:c_2 \dots ((i+n-1) \bmod p)n:c_n c_1 \dots c_n$ ' and ' $d_i i1:c_n ((i+1) \bmod p)2:c_{n-1} \dots ((i+n-1) \bmod p)n:c_1 c_1 \dots c_n$ '. We save a factor of  $p$  because of the items  $w'$  at the right of the  $a_i$  rows.

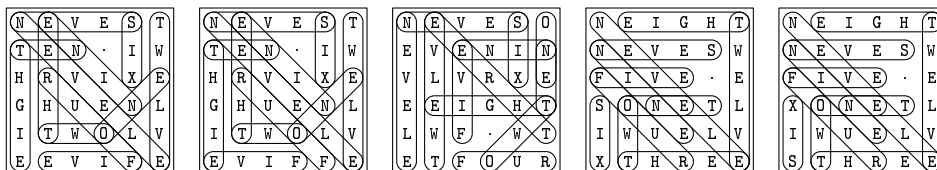
Use Algorithm C (modified). We can't have  $p = 1$ . Then comes 'SPEND|SPIES'; 'WAVES|LINED|LEPER'; 'LOOPS|POUTS|TROTS|TOONS'; 'SPOOL|STROP|STAIT|SNORT|SNOOT'; 'DIMES|MULES|RIPER|SIREN|AIDED|FINED'; 'MILES|LINTS|CARES|LAMED|PIPED|SANER|LIVER'; 'SUPER|ROVED|TILED|LICIT|CODED|ROPED|TIMED|DOMED'; 'FORTH|LURES|MIREN|POLLS|SLATS|SPOTS|SOAPS|PLOTS|LOOTS'; 'TIMES|FUROR|RUNES|MIMED|CAPED|PACED|LAVER|FINES|LIMED|MIREN'. (Lengthy computations were needed for  $p \geq 8$ .)

**71.** Now  $p \leq 2$  is impossible. A construction like the previous one allows us again to save a factor of  $p$ . (There's also top/bottom symmetry, but it is somewhat harder to exploit.) Examples are relatively easy to find, and the winners are 'MILES|GALLS|BULLS'; 'FIRES|PONDS|WALKS|LOCKS'; 'LIVES|FIRED|DIKES|WAVED|TIRES'; 'BIRDS|MARKS|POLES|WAVES|WINES|FONTS'; 'LIKED|WARES|MINES|WINDS|MALES|LOVES|FIVES'; 'WAXES|SITES|MINED|BOXES|CAVES|TALES|WIRED|MALES'; 'CENTS|HOLDS|BOILS|BALLS|MALES|WINES|FINDS|LORDS|CARES'; 'LOOKS|ROADS|BEATS|BEADS|HOLDS|COOLS|FOLKS|WINES|GASES|BOLTS'. [Such patterns were introduced by Harry Mathews in 1975, who gave the four-letter example 'TINE|SALE|MALE|VINE'. See H. Mathews and A. Brotchie, *Oulipo Compendium* (London: Atlas, 1998), 180–181.]

**75.** Given a 3SAT problem with clauses  $(l_{i1} \vee l_{i2} \vee l_{i3})$  for  $1 \leq i \leq m$ , with each  $l_{ij} \in \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\}$ , construct an exact cover problem with  $3m$  primary columns  $ij$  ( $1 \leq i \leq m, 1 \leq j \leq 3$ ) and  $n$  secondary columns  $x_k$  ( $1 \leq k \leq n$ ), having the following rows: (i) ' $l_{i1} l_{i2}$ ', ' $l_{i2} l_{i3}$ ', ' $l_{i3} l_{i1}$ '; (ii) ' $l_{ij} x_k:1$ ' if  $l_{ij} = x_k$ , ' $l_{ij} x_k:0$ ' if  $l_{ij} = \bar{x}_k$ . That problem has a solution if and only if the given clauses are satisfiable.

strong components  
Mathews  
Brotchie  
Oulipo  
3SAT

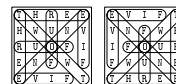
80. There are just five solutions; the latter two are flawed by being disconnected:



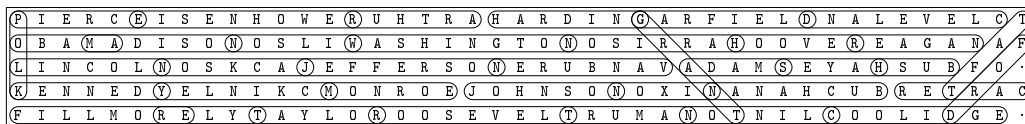
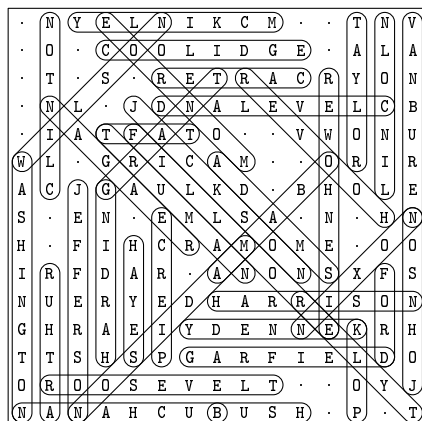
disconnected  
Gibat  
author  
interactive method  
Gordon  
Eckler

*Historical note:* Word search puzzles were invented by Norman E. Gibat in 1968.

81. When Algorithm C is generalized to allow non-unit column sums as in Algorithm M, it needs just 24 megamems to prove that there are exactly eight solutions—which all are rotations of the two shown here.



82. (a, b) The author's best solutions, thought to be minimal (but there is no proof), are below. In both cases, and in Fig. 71, an interactive method was used: After the longest words were placed strategically by hand, Algorithm C packed the others nicely.

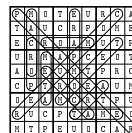


[Solution (b) applies an idea by which Leonard Gordon was able to pack the names of presidents 1–42 with one less column. See A. Ross Eckler, *Word Ways* 27 (1994), 147; see also page 252, where OBAMA miraculously fits into Gordon's 15 × 15 solution!]

**83.** To pack  $w$  given words, use primary columns  $\{P_{ij}, Ric, Cic, Bic, \#k \mid 1 \leq i, j \leq 9, 1 \leq k \leq w, c \in \{A, C, E, M, O, P, R, T, U\}\}$  and secondary columns  $\{ij \mid 1 \leq i, j \leq 9\}$ . There are 729 rows  $'P_{ij} Ric Cjc Bbc ij:c'$ , where  $b = 3\lfloor(i-1)/3\rfloor + \lceil j/3\rceil$ , together with a row  $'\#k i_1j_1:c_1 \dots i_lj_l:c_l'$  for each placement of an  $l$ -letter word  $c_1 \dots c_l$  into cells  $(i_1, j_1), \dots, (i_l, j_l)$ . Furthermore, it's important to *modify* step ?? of the algorithm so that the "best column" always has the form  $\#k$ , unless it has length  $\leq 1$ .

branch, choice of  
choice of column to cover  
best column  
Huang  
Snyder

A brief run then establishes that COMPUTER and CORPORATE cannot both be packed. But all of the words *except* CORPORATE do fit together; the (unique) solution shown is found after only 7.3 megamems, most of which are needed simply to input the problem. [This exercise was inspired by a puzzle in *Sudoku Masterpieces* (2010) by Huang and Snyder.]



**85.** To pack  $w$  given words, use  $w + m(n-1) + (m-1)n$  primary columns  $\{\#k \mid 1 \leq k \leq w\}$  and  $\{H_{ij}, V_{ij} \mid 1 \leq i \leq m, 1 \leq j \leq n\}$ , but with  $H_{in}$  and  $V_{mj}$  omitted;  $H_{ij}$  represents the edge between cells  $(i, j)$  and  $(i, j+1)$ , and  $V_{ij}$  is similar. There also are  $2mn$  secondary columns  $\{ij, ij' \mid 1 \leq i \leq m, 1 \leq j \leq n\}$ . Each horizontal placement of the  $k$ th word  $c_1 \dots c_l$  into cells  $(i, j+1), \dots, (i, j+l)$  generates the option

$$\#k ij: . ij':0 i(j+1):c_1 i(j+1)':1 Hi(j+1) i(j+2):c_2 i(j+2)':1 Hi(j+2) \dots$$

$$Hi(j+l-1) i(j+l):c_l i(j+l)':1 i(j+l+1):. i(j+l+1)':0$$

with  $3l + 4$  items, except that  $'ij: . ij':0'$  is omitted when  $j = 0$  and  $'i(j+l+1):. i(j+l+1)':0'$  is omitted when  $j+l = n$ . Each vertical placement is similar. For example,

$$\#1 11:Z 11':1 V11 21:E 21':1 V21 31:R 31':1 V31 41:0 41':1 51: . 51':0 \quad (*)$$

is the first vertical placement option for ZERO, if ZERO is word #1. When  $m = n$ , however, we save a factor of 2 by omitting all of the vertical placements of word #1.

To enforce the tricky condition (ii), we also introduce  $3m(n-1) + 3(m-1)n$  rows:

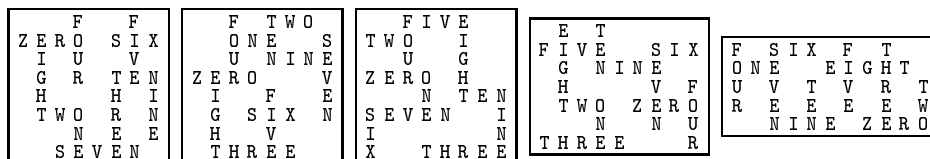
$$H_{ij} ij':0 i(j+1)':1 ij: . \quad V_{ij} ij':0 (i+1)j':1 ij: .$$

$$H_{ij} ij':1 i(j+1)':0 i(j+1):. \quad V_{ij} ij':1 (i+1)j':0 (i+1)j: .$$

$$H_{ij} ij':0 i(j+1)':0 ij: . i(j+1):. \quad V_{ij} ij':0 (i+1)j':0 ij: . (i+1)j: .$$

This construction works nicely because each edge must encounter either a word that crosses it or a space that touches it. (Beware of a slight glitch: A valid solution to the puzzle might have several compatible choices for  $H_{ij}$  and  $V_{ij}$  in "blank" regions.) *Important:* The change to step ?? in answer 83, which branches only on  $\#k$  columns unless an H or V is forced, should be followed here because it gives an enormous speedup.

The cover problem for our 11-word example has 1192 rows,  $123 + 128$  columns, and 9127 solutions, found in  $29 G\mu$ . But only 20 of those solutions are connected; and they yield only the three distinct word placements below. A slightly smaller rectangle,  $7 \times 9$ , also has three valid placements. The smallest rectangle that admits a solution to (i) and (ii) is  $5 \times 11$ ; that placement is *unique*, but it has two components:



Instead of generating all solutions to (i) and (ii) and discarding the disconnected ones, there's a much faster way to guarantee connectedness throughout the search; but



(b) With column 1 first the profile is  $(a_0, a_1, \dots, a_p, a_p a_1, \dots, a_p a_q)$ , where  $a_j = \binom{j+d}{d}$ . We should branch on column 2 first because  $a_{p+1} < a_p a_1, a_{p+2} < a_p a_2, \dots, a_q < a_p a_{q-p}, a_q a_1 < a_p a_{q-p+1}, \dots, a_q a_{p-1} < a_p a_{q-1}$ . (These inequalities follow because the sequence  $\langle a_j \rangle$  is strongly log-concave: It satisfies the condition  $a_j^2 > a_{j-1} a_{j+1}$  for all  $j \geq 1$ . See exercise MPR-125.)

**140.** Let the given shape be specified as a set of integer pairs  $(x, y)$ . These pairs might simply be listed one by one in the input; but it's much more convenient to accept a more compact specification. For example, the utility program with which the author prepared the examples of this book was designed to accept UNIX-like specifications such as '14-7]2 5[0-3]' for the seven pairs  $\{(1, 2), (4, 2), (5, 2), (6, 2), (7, 2), (5, 0), (5, 1), (5, 3)\}$ . The range  $0 \leq x, y < 62$  has proved to be sufficient in almost all instances, with such integers encoded as single "extended hexadecimal digits" 0, 1, ..., 9, a, b, ..., z, A, B, ..., Z. The specification '[1-3][1-k]' is one way to define a  $3 \times 20$  rectangle.

Similarly, each of the given polyominoes is specified by stating its piece name and a set  $T$  of typical positions that it might occupy. Such positions  $(x, y)$  are specified using the same conventions that were used for the shape; they needn't lie within that shape.

The program computes *base placements* by rotating and/or reflecting the elements of that set  $T$ . The first base placement is the shifted set  $T_0 = T - (x_{\min}, y_{\min})$ , whose coordinates are nonnegative and as small as possible. Then it repeatedly applies an elementary transformation, either  $(x, y) \mapsto (y, x_{\max} - x)$  or  $(x, y) \mapsto (y, x)$ , to every existing base placement, until no further placements arise. (That process becomes easy when each base placement is represented as a sorted list of packed integers  $(x \ll 16) + y$ .) For example, the typical positions of the straight tromino might be specified as '1[1-3]'; it will have two base placements,  $\{(0, 0), (0, 1), (0, 2)\}$  and  $\{(0, 0), (1, 0), (2, 0)\}$ .

After digesting the input specifications, the program defines the columns of the exact problem, which are the piece names together with the cells  $xy$  of the given shape.

Finally, it defines the rows: For each piece  $p$  and for each base placement  $T'$  of  $p$ , and for each offset  $(\delta_x, \delta_y)$  such that  $T' + (\delta_x, \delta_y)$  lies fully within the given shape, there's a row that names the columns  $\{p\} \cup \{(x + \delta_x, y + \delta_y) \mid (x, y) \in T'\}$ .

(The output of this program is often edited by hand, to take account of special circumstances. For example, some columns may change from primary to secondary; some rows may be eliminated in order to break symmetry. The author's implementation also allows the specification of secondary columns with color controls, along with base placements that include such controls.)

**148.** RUSTY. [Leigh Mercer posed a similar question to Martin Gardner in 1960.]

**150.** As in the  $3 \times 20$  example considered in the text, we can set up an exact cover problem with  $12 + 60$  columns, and with rows for every potential placement of each piece. This gives respectively (52, 292, 232, 240, 232, 120, 146, 120, 120, 30, 232, 120) rows for pieces (O, P, ..., Z) in Conway's nomenclature, thus 1936 rows in all.

To reduce symmetry, we can insist that the X occurs in the upper left corner; then it contributes just 10 rows instead of 30. But some solutions are still counted twice, when X is centered in the middle row. To prevent this we can add a *secondary column* 's', and append 's' to the five rows that correspond to those centered appearances; we also append 's' to the 60 rows that correspond to placements where the Z is flipped over.

Without those changes, Algorithm D would use 9.76  $G\mu$  to find 4040 solutions; with them, it needs just 2.86  $G\mu$  to find 1010.

This approach to symmetry breaking in pentomino problems is due to Dana Scott [Technical Report No. 1 (Princeton University Dept. of Electrical Engineering, 10 June

log-concave  
author  
UNIX  
extended hexadecimal digits  
hexadecimal notation, extended  
base placements  
sorted  
packed integers  
straight tromino  
secondary  
break symmetry  
author  
color controls  
Mercer  
Gardner  
Conway  
secondary column  
Scott

1958)]. Another way to break symmetry would be to allow X anywhere, but to restrict the W to its 30 *unrotated* placements. That works almost as well: 2.87  $G\mu$ .

**151.** There's a unique way to pack P, Q, R, U, X into a  $5 \times 5$  square, and to pack the other seven into a  $5 \times 7$ . (See below.) With independent reflections, together with rotation of the square, we obtain 16 of the 1010. There's also a unique way to pack P, R, U into a  $5 \times 3$  and the others into a  $5 \times 9$  (noticed by R. A. Fairbairn in 1967), yielding 8 more. And there's a unique way to pack O, Q, T, W, Y, Z into a  $5 \times 6$ , plus two ways to pack the others, yielding another 16. (These paired  $5 \times 6$  patterns were apparently first noticed by J. Pestiau; see answer 169.) Finally, the packings in the next exercise give us 264 decomposable  $5 \times 12$ s altogether.

[Similarly, C. J. Bouwkamp discovered that S, V, T, Y pack uniquely into a  $4 \times 5$ , while the other eight can be put into an  $4 \times 10$  in five ways, thus accounting for 40 of the 368 distinct  $4 \times 15$ s. See *JRM* **3** (1970), 125.]



**152.** Without symmetry reduction, 448 solutions are found in 1.21  $G\mu$ . But we can restrict X to the upper left corner, flagging its placements with 's' when centered in the middle row or middle column (but not both). Again the 's' is appended to flipped Z's. Finally, when X is placed in dead center, we append *another* secondary column 'c', and append 'c' to the 90 rotated placements of W. This yields 112 solutions, after 0.34  $G\mu$ .

Or we could leave X unhindered but curtail W to 1/4 of its placements. That's easier to do (although not *quite* as clever) and it finds those 112 in 0.42  $G\mu$ .

Incidentally, there *aren't* actually any solutions with X in dead center.

**154.** The exact cover problem analogous to that in exercise 150 has  $12 + 60$  columns and (56, 304, 248, 256, 248, 128, 1152, 128, 128, 32, 248, 128) rows. It finds 9356 solutions after 15.93  $G\mu$  of computation, without symmetry reduction. But if we insist that X be centered in the upper left quarter, by removing all but 8 of its placements, we get 2339 solutions after just 3.93  $G\mu$ . (The alternative of restricting W's rotations is *not* as effective in this case: 5.43  $G\mu$ .) These solutions were first enumerated by C. B. and Jenifer Haselgrove [*Eureka: The Archimedean's Journal* **23** (1960), 16–18].

**155.** (a) Obviously only  $k = 5$  is feasible. All such packings can be obtained by omitting all rows of the cover problem that straddle the "cut." That leaves 1507 of the original 2032 rows, and yields 16 solutions after 104  $M\mu$ . (Those 16 boil down to just the two  $5 \times 6$  decompositions that we already saw in answer 151.)

(b) Now we remove the 763 rows for placements that don't touch the boundary, and obtain just the two solutions below, after 100  $M\mu$ . (This result was first noticed by Tony Potts, who posted it to Martin Gardner on 9 February 1960.)

(c) Now there are 1237 placements/rows; the *unique* solution is found after 83  $M\mu$ .

(d) There are respectively (0, 9, 3, 47, 16, 8, 3, 1, 30, 22, 5, 11) solutions for pentominoes (O, P, Q, . . . , Z). (The I/O pentomino can be "framed" by the others in 11 ways; but all of those packings also have at least one other interior pentomino.)

(e) Despite many ways to cover all boundary cells with just seven pentominoes, none of them lead to an overall solution. Thus the minimum is eight; 207 of the 2339 solutions attain it. To find them we might as well generate and examine all 2339.

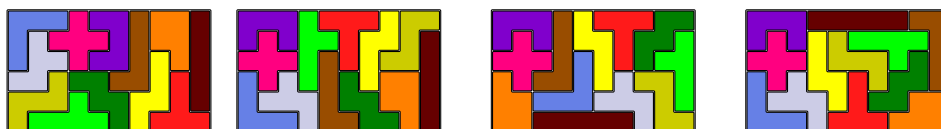
(f) The question is ambiguous: If we're willing to allow the X to touch unnamed pieces at a corner, but not at an edge, there are 25 solutions (8 of which happen to

break symmetry  
Fairbairn  
Pestiau  
Bouwkamp  
Haselgrove, Colin  
Haselgrove, Jenifer  
Potts  
Gardner

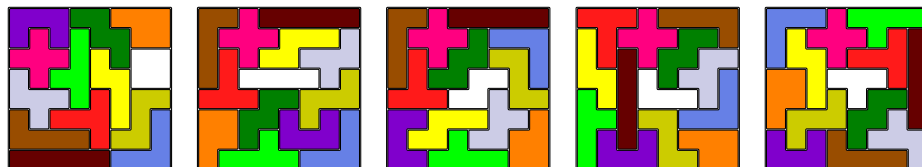


be answers to part (a)). In each of these solutions, X also touches the outer boundary. (The cover and frontispiece of Clarke's book show a packing in which X doesn't touch the boundary, but it *doesn't* solve this problem: There's an edge where X meets I, and there's a point where X meets P.) There also are two packings in which the edges of X touch only F, N, U, and the boundary, but not V.

On the other hand, there are just 6 solutions if we allow only F, N, U, V to touch X's corner points. One of them, shown below, has X touching the short side and seems to match the quotation best. These 6 solutions can be found in just 47  $M\mu$ , by introducing 60 secondary columns as sort of an "upper level" to the board: All placements of X occupy the normal five lower-level cells, plus up to 16 upper-level cells that touch them; all placements of F, N, U, V are unchanged; all placements of the other seven pieces occupy both the lower and the upper level. This nicely forbids them from touching X.




**157.** Restrict X to five essentially different positions; if X is on the diagonal, also keep Z unflipped by using the second column 's' as in answer 152. There are respectively (16146, 24600, 23619, 60608, 25943) solutions, found in (19.8, 35.4, 27.3, 66.6, 34.5)  $G\mu$ .







In each case the tetromino can be placed anywhere that doesn't immediately cut off a region of one or two squares. [The twelve pentominoes first appeared in print when H. E. Dudeney published *The Canterbury Puzzles* in 1907. His puzzle #74, "The Broken Chessboard," presented the first solution shown above, with pieces checkered in black and white. That parity restriction, with the further condition that no piece is turned over, would reduce the number of solutions to only 4, findable in 120  $M\mu$ .]

The 60-element subsets of the chessboard that *can't* be packed with the pentominoes has been characterized by M. Reid in *JRM* **26** (1994), 153–154.

**158.** Yes, in seven essentially different ways. To remove symmetry, we can make the I vertical and put the X in the right half. (The pentominoes will have a total of  $6 \times 2 + 5 \times 3 + 4 = 31$  black squares; therefore the tetromino *must* be )



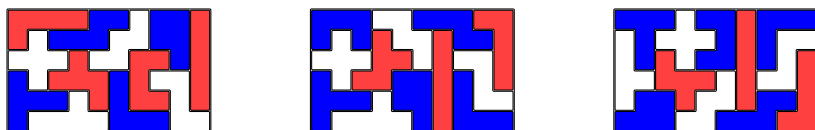
**159.** These shapes can't be packed in a rectangle. But we can use the "supertile"  to make an infinite strip  $\cdots$    $\cdots$ . We can also tile the plane with a supertile like , or even use a generalized torus such as  (see exercise 7–137). That supertile was used in 2009 by George Sicherman to make tetromino wallpaper.

**160.** The 2339 solutions contain 563 that satisfy the "tatami" condition: No four pieces meet at any one point. Each of those 563 leads to a simple 12-vertex graph coloring problem; for example, the SAT methods of Section 7.2.2.2 typically need at most two or three kilomems to decide each case.

secondary columns  
Dudeney  
parity  
one-sided pentominoes  
Reid  
symmetry  
torus  
torus, generalized  
Sicherman  
wallpaper  
tatami  
SAT

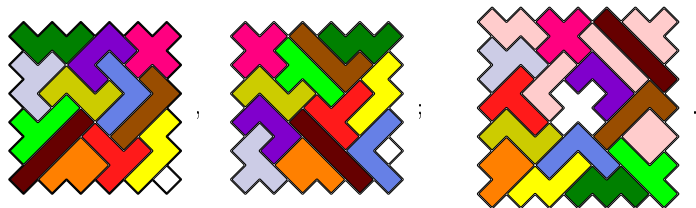
It turns out that exactly 94 are three-colorable, including the second solution to exercise 155(b). Here are the three for which W, X, Y, Z all have the same color:

secondary column  
Gardner  
Hawkins  
Lindon  
Fuhlendorf  
symmetries  
three-colorable



**162.** Both shapes have 8-fold symmetry, so we can save a factor of nearly 8 by placing the X in (say) the north-northwest octant. If X thereby falls on the diagonal, or in the middle column, we can insist that the Z is not flipped, by introducing a secondary column 's' as in answer 152. Furthermore, if X occurs in dead center — this is possible only for shape (i) — we use 'c' as in that answer to prohibit also any rotation of the W.

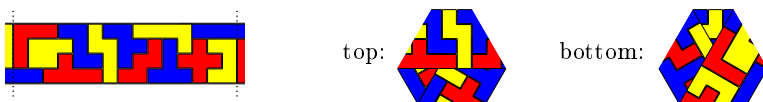
Thus find (a) 10 packings, in  $3.5 G\mu$ ; (b) 7302 packings, in  $353 G\mu$ ; for instance



It turns out that the monomino must appear in or next to a corner, as shown. [The first solution to shape (i) with monomino in the corner was sent to Martin Gardner by H. Hawkins in 1958. The first solution of the other type was published by J. A. Lindon in *Recreational Mathematics Magazine* #6 (December 1961), 22. Shape (ii) was introduced and solved much earlier, by G. Fuhlendorf in *The Problemist: Fairy Chess Supplement* 2, 17 and 18 (April and June, 1936), problem 2410.]

**163.** (Notice that width 3 would be impossible, because every faultfree placement of the V needs width 4 or more.) We can set up an exact cover problem for a  $4 \times 19$  rectangle in the usual way; but then we make cell  $(x, y + 15)$  identical to  $(3 - x, y)$  for  $0 \leq x < 4$  and  $0 \leq y < 5$ , essentially making a half-twist when the pattern begins to wrap around. There are 60 symmetries, and care is needed to remove them properly. The easiest way is to put X into a fixed position, and allow W to rotate at most  $90^\circ$ .

This exact cover problem has 850 solutions, 502 of which are faultfree. Here's one of the 29 strongly three-colorable ones, shown before and after its ends are joined:

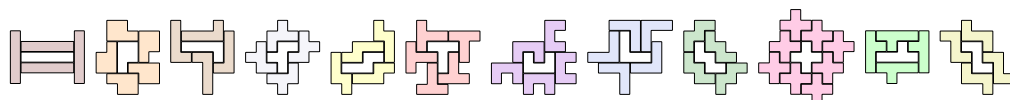


**164.** It's also possible to wrap *two* cubes of size  $\sqrt{5} \times \sqrt{5} \times \sqrt{5}$ , as shown by F. Hansson; see *Fairy Chess Review* 6 (1947–1948), problems 7124 and 7591. A full discussion appears in *FGbook*, pages 685–689.

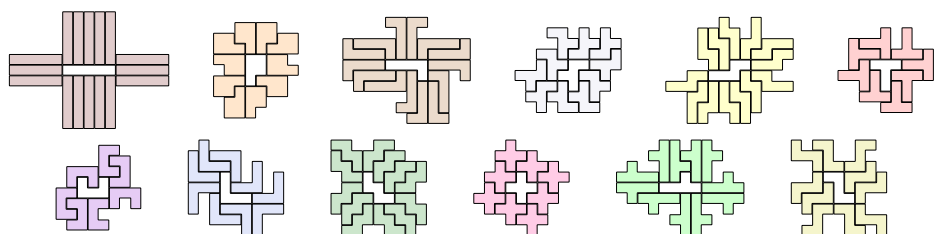


**165.** It's easy to set up an exact cover problem in which the cells touching the polyomino are primary columns, while other cells are secondary, and with rows restricted to placements that contain at least one primary column. Postprocessing can then remove

spurious solutions that contain holes. Typical answers for (a) are

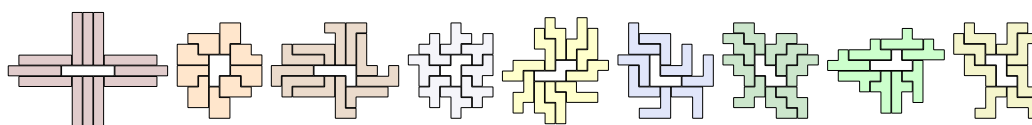


representing respectively (9, 2153, 37, 2, 17, 28, 18, 10, 9, 2, 4, 1) cases. For (b) they're



representing (16, 642, 1, 469, 551, 18, 24, 6, 4, 2, 162, 1). The total number of fences is respectively (3120, 1015033, 8660380, 284697, 1623023, 486, 150, 2914, 15707, 2, 456676, 2074), after weeding out respectively (0, 0, 16387236, 398495, 2503512, 665, 600, 11456, 0, 0, 449139, 5379) cases with holes. (See *MAA Focus* **36**, 3 (June/July 2016), 26; **36**, 4 (August/September 2016), 33.) Of course we can also make fences for one shape by using *other* shapes; for example, there's a beautiful way to fence a Z with 12 Ws, and a unique way to fence one pentomino with only *three* copies of another.

**166.** The small fences of answer 165(a) already meet this condition—except for the X, which has *no* tatami fence. The large fences for T and U in 165(b) are also good. But the other nine fences can no longer be as large:



[The tatami condition can be incorporated into the exact cover problem by using color controls: Introduce a secondary column for every potential edge between tiles, with values *t* and *f*. Also introduce a primary column *p* for every corner point; *p* will appear only in four rows '*p e:f*', one for each edge *e* that touches *p*. In every row for the placement of a piece, include the columns '*e:f*' for every edge *internal* to that piece, and '*e:t*' for every edge at the *boundary* of that piece. Then every edge point will be next to a nonedge. However, for this exercise it's best simply to apply the tatami condition directly to each ordinary solution, before postprocessing for hole-removal.]

**167.** This problem is readily solved with the “second death” algorithm of exercise 10, by letting the four designated piece names be the *only* primary columns. The answers to both (a) and (b) are unique. [See M. Gardner, *Scientific American* **213**, 4 (October 1965), 96–102, for Golomb's conjectures about minimum blocking configurations on larger boards.]



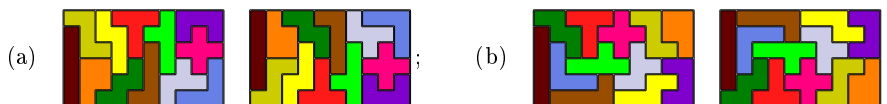
**168.** This exercise, with  $3 \times 30$ ,  $5 \times 18$ ,  $6 \times 15$ , and  $9 \times 10$  rectangles, yields four increasingly difficult benchmarks for the exact cover problem, having respectively (46, 686628, 2562928, 10440433) solutions. Symmetry can be broken as in exercise 152. The  $3 \times 30$  case was first resolved by J. Haselgrove; the  $9 \times 10$  packings were first enumerated by A. Wassermann and P. Östergård, independently. [See *New Scientist*

color controls  
gadget  
second death  
Gardner  
pentominoes, shortest games  
benchmarks  
Haselgrove  
Wassermann  
Östergård

**12** (1962), 260–261; J. Meeus, *JRM* **6** (1973), 215–220; and *FGbook* pages 455, 468–469.] Algorithm D needs (.006, 5.234, 15.576, 63.386) teramems to find them. (I plan to give statistics for improved versions too; please stay tuned.)

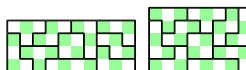
Meeus  
 180° rotation  
 central symmetry  
 Patent  
 Pestiau  
 Guy  
 checkering  
 parity  
 parity  
 exact cover  
 factoring

**169.** Two solutions are now equivalent only when related by 180° rotation. Thus there are  $2 \cdot 2339/64 = 73.09375$  solutions per problem, on average. The minimum (42) and maximum (136) solution counts occur for the cases



[In *U.S. Patent 2900190* (1959, filed 1956), J. Pestiau remarked that these 64 problems would give his pentomino puzzle “unlimited life and utility.”]

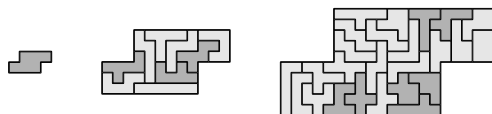
**170.** There are no ways to fill  $2 \times 20$ ;  $4 \times 66$  ways to fill  $4 \times 10$ ;  $4 \times 84$  ways to fill  $5 \times 8$ . None of the solutions are symmetrical. [See R. K. Guy, *Nabla* **7** (1960), 99–101.]



**175.** Most of the hexominoes will have three black cells and three white cells, in any “checkering” of the board. However, eleven of them (shown as darker gray in the illustration) will have a two-to-four split. Thus the total number of black cells will always be an even number between 94 and 116, inclusive. But a 210-cell rectangle always contains exactly 105 black cells. [See *The Problemist: Fairy Chess Supplement* **2**, 9–10 (1934–1935), 92, 104–105; *Fairy Chess Review* **3**, 4–5 (1937), problem 2622.]

Benjamin’s triangular shape, on the other hand, has  $1+3+5+\dots+19 = 10^2 = 100$  cells of one parity and  $\binom{20}{2} - 10^2 = 110$  of the other. It can be packed with the 35 hexominoes in a huge number of ways, probably not feasible to count exactly.

**176.** The parity considerations in answer 175 tell us that this is possible only for the “unbalanced” hexominoes, such as the one shown. And in fact, Algorithm D readily finds solutions for all eleven of those, too numerous to count. Here’s an example:



[See *Fairy Chess Review* **6** (April 1947) through **7** (June 1949), problems 7252, 7326, 7388, 7460, 7592, 7728, 7794, 7865, 7940, 7995, 8080. See also the similar problem 7092.]



**177.** Each castle must contain an odd number of the eleven unbalanced hexominoes (see answer 175). Thus we can begin by finding all sets of seven hexominoes that can be packed into a castle: This amounts to solving  $\binom{11}{1} + \binom{11}{3} + \binom{11}{5} + \binom{11}{7} = 968$  exact cover problems, one for each potential choice of unbalanced elements. Each of those problems is fairly easy; the 24 balanced hexominoes provide secondary columns, while the castle cells and the chosen unbalanced elements are primary. In this way we obtain 39411 suitable sets of seven hexominoes, with only a moderate amount of computation.


That gives us *another* exact cover problem, having 35 columns and 39411 rows. This secondary problem turns out to have exactly 1201 solutions (found in just 115 Gμ), each of which leads to at least one of the desired overall packings. Here’s one:



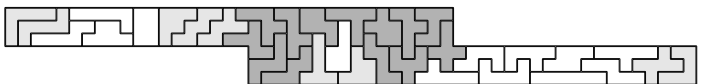
In this example, two of the hexominoes in the rightmost castle can be flipped vertically; and of course the entire contents of each castle can independently be flipped horizontally. Thus we get  $64$  packings from this particular partition of the hexominoes (or maybe  $64 \times 5!$ , by permuting the castles), but only two of them are “really” distinct. Taking multiplicities into account, there are  $1803$  “really” distinct packings altogether.

Hansson  
Povah  
Hansson  
Sicherman  
strongly three-colorable  
dynamic programming

[Frans Hansson found the first way to pack the hexominoes into five equal shapes, using  as the container; see *Fairy Chess Review* **8** (1952–1953), problem 9442. His container admits  $123189$  suitable sets of seven, and  $9298602$  partitions into five suitable sets instead of only  $1201$ . Even more packings are possible with the container , which has  $202289$  suitable sets and  $3767481163$  partitions!]

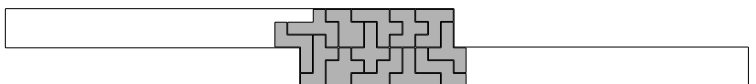
In 1965, M. J. Povah packed all of the hexominoes into containers of shape , using *seven* sets of *five*; see *The Games and Puzzles Journal* **2** (1996), 206.

**178.** By exercise 175,  $m$  must be odd, and less than  $35$ . F. Hansson posed this question in *Fairy Chess Review* **7** (1950), problem 8556. He gave a solution for  $m = 19$ ,



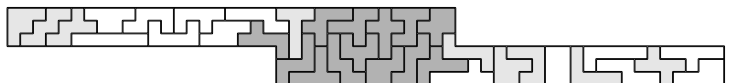
and claimed without proof that  $19$  is optimum. The  $13$  dark gray hexominoes in this diagram cannot be placed in either “arm”; so they must go in the center. (Medium gray indicates pieces that have parity restrictions in the arms.) Thus we cannot have  $m \geq 25$ .

When  $m = 23$ , there are  $39$  ways to place all of the hard hexominoes, such as



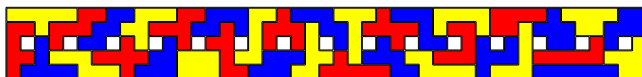
However, none of these is completable with the other  $22$ ; hence  $m \leq 21$ .

When  $m = 21$ , the hard hexominoes can be placed in  $791792$  ways, without creating a region whose size isn't a multiple of  $6$  and without creating more than one region that matches a particular hexomino. Those  $791792$  ways have  $69507$  essentially distinct “footprints” of occupied cells, and the vast majority of those footprints appear to be impossible to fill. But in 2016, George Sicherman found the remarkable packing



which not only solves  $m = 21$ , it yields solutions for  $m = 19, 17, 15, 11, 9, 7, 5$ , and  $3$  by simple modifications. Sicherman also found separate solutions for  $m = 13$  and  $m = 1$ .

**179.** Stead's original solution makes a very pleasant three-colored design:



[See *Fairy Chess Review* **9** (1954), 2–4; also *FGbook*, pages 659–662.]

This problem is best solved via the techniques of dynamic programming (Section 7.7), *not* with Algorithm D, because numerous subproblems are equivalent.

**181.** Make rows for the pentominoes in cells  $xy$  for  $0 \leq x < 8, 0 \leq y < 10$  as in exercise 140, and also for the tetrominoes in cells  $xy$  for  $1 \leq x < 7, 1 \leq y < 9$ . In the latter rows include also columns  $xy':0$  for all cells  $xy$  in the tetromino, as well as  $xy':1$  for

all other cells  $xy$  touching the tetromino, where the columns  $xy'$  for  $0 \leq x < 8$  and  $0 \leq y < 10$  are secondary. We can also assume that the center of the X pentomino lies in the upper left corner. There are 168 solutions, found after  $1.5 T\mu$  of computation. (Another way to keep the tetrominoes from touching would be to introduce secondary columns for the *vertices* of the grid. Such columns are more difficult to implement, however, because they behave differently under the rotations of answer 140.)

Benjamin  
Keller  
Torbijn  
Meeus  
nested parentheses  
forest  
continued fraction  
Bessel functions, gen'lized  
Catalan numbers  
tatami  
strongly three-colorable  
secondary columns

[Many problems that involve placing the tetrominoes and pentominoes together in a rectangle were explored by H. D. Benjamin and others in the *Fairy Chess Review*, beginning already with its predecessor *The Problemist: Fairy Chess Supplement* (1936), problem 2171. But this particular question seems to have been raised first by Michael Keller in *World Game Review* **9**, (1989), xx.]

**182.** At present, not a single solution to this puzzle is known, although intuition suggests that enormously many of them ought to be possible. P. J. Torbijn and J. Meeus [*JRM* **32** (2003), 78–79] have exhibited solutions for rectangles of sizes  $6 \times 45$ ,  $9 \times 30$ ,  $10 \times 27$ , and  $15 \times 18$ .

**198.** (a) Represent the tree as a sequence  $a_0 a_1 \dots a_{2n+1}$  of nested parentheses; then  $a_1 \dots a_{2n}$  will represent the corresponding root-deleted forest, as in Algorithm 7.2.1.6P. The left boundary of the corresponding parallomino is obtained by mapping each ‘(’ into N or E, according as it is immediately followed by ‘(’ or ‘)’. The right boundary, similarly, maps each ‘)’ into N or E according as it is immediately preceded by ‘)’ or ‘(’. For example, the parallomino for forest 7.2.1.6–(2) is shown below with part (d).

(b) This series  $wxy + w^2(xy^2 + x^2y) + w^3(xy^3 + 2x^2y^2 + x^3y) + \dots$  can be written  $wxyH(w, wx, wy)$ , where  $H(w, x, y) = 1/(1 - x - y - G(w, x, y))$  generates a sequence of “atoms” corresponding to places  $x, y, G$  where the juxtaposed boundary paths have the respective forms  $\begin{smallmatrix} E \\ E \end{smallmatrix}, \begin{smallmatrix} N \\ N \end{smallmatrix}$ , or  $\begin{smallmatrix} N \\ E \end{smallmatrix}(\text{inner})\begin{smallmatrix} E \\ N \end{smallmatrix}$ . The area is thereby computed by diagonals between corresponding boundary points. (In the example from (a), the area is  $1+1+1+1+2+2+2+2+2+2+2+2+2+2+1+1$ ; there’s an “outer”  $G$ , whose  $H$  is  $xyxyGy$ , and an “inner”  $G$ , whose  $H$  is  $xyxyxyxy$ .) Thus we can write  $G$  as a continued fraction,

$$G(w, x, y) = wxy / (1 - x - y - wxy / (1 - wx - wy - w^3xy / (1 - w^2x - w^2y - w^5xy / (\dots))))$$

[A completely different form is also possible, namely  $G(w, x, y) = x \frac{J_1(w, x, y)}{J_0(w, x, y)}$ , where

$$J_0(w, x, y) = \sum_{n=0}^{\infty} \frac{(-1)^n y^n w^{n(n+1)/2}}{(1-w)(1-w^2) \dots (1-w^n)(1-xw)(1-xw^2) \dots (1-xw^n)}$$

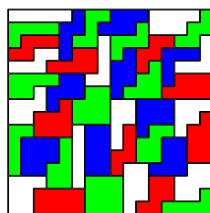
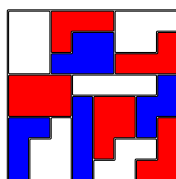
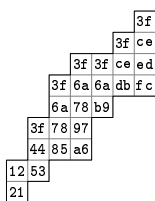
$$J_1(w, x, y) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1} y^n w^{n(n+1)/2}}{(1-w)(1-w^2) \dots (1-w^{n-1})(1-xw)(1-xw^2) \dots (1-xw^n)}$$

This form, derived via *horizontal* slices, disguises the symmetry between  $x$  and  $y$ .]

(c) Let  $G(w, z) = G(w, z, z)$ . We want  $[z^n] G'(1, z)$ , where differentiation is with respect to the first parameter. From the formulas in (b) we know that  $G(1, z) = z(C(z) - 1)$ , where  $C(z) = (1 - \sqrt{1-4z})/(2z)$  generates the Catalan numbers. Partial derivatives  $\partial/\partial w$  and  $\partial/\partial z$  then give  $G'(1, z) = z^2/(1-4z)$  and  $G_z(1, z) = 1/\sqrt{1-4z} - 1$ .

(d) This problem has four symmetries, because we can reflect about either diagonal. When  $n = 5$ , Algorithm D finds  $4 \times 801$  solutions, of which  $4 \times 129$  satisfy the tatami condition, and  $4 \times 16$  are strongly three-colorable. (The tatami condition is easily enforced via secondary columns in this case, because we need only stipulate that the upper right corner of one parallomino doesn’t match the lower left corner of another.)

When  $n = 6$  there are oodles and oodles of solutions. All of the trees/parallominoes thereby appear together in an attractive compact pattern.



[References: D. A. Klarner and R. L. Rivest, *Discrete Math.* **8** (1974), 31–40; E. A. Bender, *Discrete Math.* **8** (1974), 219–226; I. P. Goulden and D. M. Jackson, *Combinatorial Enumeration* (New York: Wiley, 1983), exercise 5.5.2; M.-P. Delest and G. Viennot *Theoretical Comp. Sci.* **34** (1984), 169–206; W.-J. Woan, L. Shapiro, and D. G. Rogers, *AMM* **104** (1997), 926–931; P. Flajolet and R. Sedgewick, *Analytic Combinatorics* (Cambridge Univ. Press, 2009), 660–662.]


geek art  
Klarner  
Rivest  
Bender  
Goulden  
Jackson  
Delest  
Viennot  
Woan  
Shapiro  
Rogers  
Flajolet  
Sedgewick  
pendant vertex: of degree 1  
diameter  
Guy  
Conway  
Guy  
Berlekamp  
Conway  
Guy

**200.** The same ideas apply, but with three coordinates instead of two, and with the elementary transformations  $(x, y, z) \mapsto (y, x_{\max} - x, z)$ ,  $(x, y, z) \mapsto (y, z, x)$ .

Pieces (1, 2, . . . , 7) have respectively (12, 24, 12, 12, 12, 12, 8) base placements, leading to 144 + 144 + 72 + 72 + 96 + 96 + 64 rows for the  $3 \times 3 \times 3$  problem.

**202.** It’s tempting, but wrong, to try to compute the Somap by considering only the 240 solutions that have the tee in a fixed position and the claw restricted; the pairwise semidistances between these special solutions will miss many of the actual adjacencies. To decide if  $u \sim v$ , one must compare  $u$  to the 48 solutions equivalent to  $v$ .

(a) The strong Somap has vertex degrees  $7^1 6^7 5^{19} 4^{31} 3^{59} 2^{63} 1^{45} 0^{15}$ ; so an “average” solution has  $(1 \cdot 7 + 7 \cdot 6 + \dots + 15 \cdot 0)/240 \approx 2.57$  strong neighbors. (The unique vertex of degree 7 has the level-by-level structure  $\begin{smallmatrix} 333 & 114 & 171 \\ 552 & 652 & 662 \end{smallmatrix}$  from bottom to top.)

The full Somap has vertex degrees  $21^2 18^1 16^9 15^{13} 14^{10} 13^{16} 12^{17} 11^{12} 10^{16} 9^{28} 8^{26} 7^{25} 6^{26} 5^{16} 4^{17} 3^3 2^1 1^1 0^1$ , giving an average degree  $\approx 9.14$ . (Its unique isolated vertex is  $\begin{smallmatrix} 333 & 488 & 115 \\ 462 & 762 & 772 \end{smallmatrix}$ , and its only pendant vertex is  $\begin{smallmatrix} 333 & 188 & 111 \\ 222 & 462 & 466 \end{smallmatrix}$ . Two other noteworthy solutions,  $\begin{smallmatrix} 333 & 412 & 115 \\ 466 & 762 & 772 \end{smallmatrix}$  and  $\begin{smallmatrix} 333 & 412 & 112 \\ 466 & 765 & 772 \end{smallmatrix}$ , are the only ones that contain the two-piece substructure )

(b) The Somap has just two components, namely the isolated vertex and the 239 others. The latter has just three bicomponents, namely the pendant vertex, its neighbor, and the 237 others. Its diameter is 8 (or 21, if we use the edge lengths 2 and 3).

The strong Somap has a much sparser and more intricate structure. Besides the 15 isolated vertices, there are 25 components of sizes  $\{8 \times 2, 6 \times 3, 4, 3 \times 5, 2 \times 6, 7, 8, 11, 16, 118\}$ . Using the algorithm of Section 7.4.1, the large component breaks down into nine bicomponents (one of size 2, seven of size 1, the other of size 109); the 16-vertex component breaks into seven; and so on, totalling 58 bicomponents altogether.

[The Somap was first constructed by R. K. Guy, J. H. Conway, and M. J. T. Guy, without computer help. It appears on pages 910–913 of Berlekamp, Conway, and Guy’s *Winning Ways*, where all of the strong links are shown, and where enough other links are given to establish near-connectedness. Each vertex in that illustration has been given a code name; for example, the five special solutions mentioned in part (a) have code names B5f, R7d, LR7g, YR3a, and R3c, respectively.]

**204.** Let the cubic coordinates be  $51z, 41z, 31z, 32z, 33z, 23z, 13z, 14z, 15z$ , for  $z \in \{1, 2, 3\}$ . Replace matrix  $A$  of the exact cover problem by a simplified matrix  $A'$  having only columns (1, 2, 3, 4, 5, 6, 7, S), where S is the sum of all columns  $xyz$  of  $A$  where  $xyz$  is

odd. Any solution to  $A$  yields a solution to  $A'$  with column sums  $(1, 1, 1, 1, 1, 1, 1, 10)$ . But that's impossible, because the rows of  $A'$  all have the forms '1 [S]', '2 [S] [S]', '3 [S] [S]', '4 [S]', '5 [S]', '6 [S]', '7 [S]'. [See the Martin Gardner reference in answer 213.]

**205.** (a) The solution counts, ignoring symmetry reduction, are:  $4 \times 5$  corral (2), gorilla (2), smile (2),  $3 \times 6$  corral (4), face (4), lobster (4), castle (6), bench (16), bed (24), doorway (28), piggybank (80), five-seat bench (104), piano (128), shift 2 (132),  $4 \times 4$  coop (266), shift 1 (284), bathtub (316), shift 0 (408), grand piano (526), tower 4 (552), tower 3 (924), canal (1176), tower 2 (1266), couch (1438), tower 1 (1520), stepping stones (2718). So the  $4 \times 5$  corral, gorilla, and smile are tied for hardest, while stepping stones are the easiest. (The bathtub, canal, bed, and doorway each have four symmetries; the couch, stepping stones, tower 4, shift 0, bench,  $4 \times 4$  coop, castle, five-seat bench, piggybank, lobster, piano, gorilla, face, and smile each have two. To get the number of *essentially distinct* solutions, divide by the number of symmetries.)

(b) Notice that the canal, bed, and doorway appear also in (a), as does the dryer (which is the same as "stepping stones"). The solution counts are: W-wall (0), almost W-wall (12), bed (24), apartments 2 (28), doorway (28), clip (40), tunnel (52), zigzag wall 2 (52), zigzag wall 1 (92), underpass (132), chair (260), stile (328), fish (332), apartments 1 (488), goldfish (608), canal (1176), steps (2346), dryer (2718); hence "almost W-wall" is the hardest of the possible shapes. Notice that the dryer, chair, steps, and zigzag wall 2 each have two symmetries, while the others in Fig. 80(b) all have four. The  $3 \times 3 \times 3$  cube, with its 48 symmetries, probably is the easiest possible shape to make from the Soma pieces.

[Piet Hein himself published the tower 1, shift 2, stile, and zigzag wall 1 in his original patent; he also included the bathtub, bed, canal, castle, chair, steps, stile, stepping stones, shift 1, five-seat bench, tunnel, W-wall, and both apartments in his booklet for Parker Brothers. Parker Brothers distributed four issues of *The SOMA® Addict* in 1970 and 1971, giving credit for new constructions to Noble Carlson (fish, lobster), Mrs. C. L. Hall (clip, underpass), Gerald Hill (towers 2–4), Craig Kenworthy (goldfish), John W. M. Morgan (cot, face, gorilla, smile), Rick Murray (grand piano), and Dan Smiley (doorway, zigzag wall 2). Sivy Farhi published a booklet called *Somacubes* in 1977, containing the solutions to more than one hundred Soma cube problems including the bench, the couch, and the piggybank.]

**206.** By eliminating symmetries, there are (a) 421 distinct cases with cubies omitted on both layers, and (b) 129 with cubies omitted on only one layer. All are possible, except in the one case where the omitted cubies disconnect a corner cell. The easiest of type (a) omits (111, 112, 311) and has 3599 solutions; the hardest omits (211, 222, 231) and has  $45 \times 2$  solutions. The easiest of type (b) omits (111, 151, 311) and has 3050 solutions; the hardest omits (211, 221, 251) and has  $45 \times 2$  solutions. (The two examples illustrated have  $821 \times 2$  and  $68 \times 4$  solutions. Early Soma solvers seem to have overlooked them!)

**207.** (a) The 60 distinct cases are all quite easy. The easiest has 3497 solutions and uses (113, 123, 213) on the top level; the hardest has 268 solutions and uses (113, 223, 313).

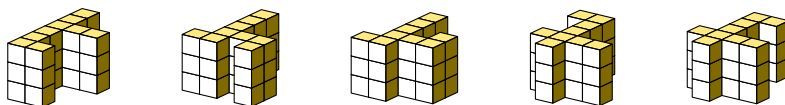
(b) Sixteen of the 60 possibilities are disconnected. Three of the others are also impossible—namely those that omit  $(12z, 24z, 32z)$  or  $(21z, 22z, 23z)$  or  $(21z, 22z, 24z)$ . The easiest has 3554 solutions and omits  $(11z, 12z, 34z)$ ; the hardest of the possibles has only 8 solutions and omits  $(11z, 23z, 24z)$ .

(The two examples illustrated have  $132 \times 2$  and  $270 \times 2$  solutions.)

Gardner  
symmetries  
Hein  
Parker Brothers  
Carlson  
Hall  
Hill  
Kenworthy  
Morgan  
Murray  
Smiley  
Farhi  
symmetries



208. All but 216 are realizable. Five cases have unique (1 × 2) solutions:



secondary column author Skjøde Skjern Knutsen, see Skjøde Skjern Kustes Morgan Schwartz Conway Gardner

210. Every polycube has a minimum enclosing box for which it touches all six faces. If those box dimensions  $a \times b \times c$  aren't too large, we can generate such polycubes uniformly at random in a simple way: First choose 27 of the  $abc$  possible cubies; try again if that choice doesn't touch all faces; otherwise try again if that choice isn't connected.

For example, when  $a = b = c = 4$ , about 99.98% of all choices will touch all faces, and about 0.1% of those will be connected. This means that about  $.001 \binom{64}{27} \approx 8 \times 10^{14}$  of the 27-cubie polycubes have a  $4 \times 4 \times 4$  bounding box. Of these, about 5.8% can be built with the seven Soma pieces.

But most of the relevant polycubes have a larger bounding box; and in such cases the chance of solvability goes down. For example,  $\approx 6.2 \times 10^{18}$  cases have bounding box  $4 \times 5 \times 5$ ;  $\approx 3.3 \times 10^{18}$  cases have bounding box  $3 \times 5 \times 7$ ;  $\approx 1.5 \times 10^{17}$  cases have bounding box  $2 \times 7 \times 7$ ; and only 1% or so of those cases are solvable.

Section 7.2.3 will discuss the enumeration of polycubes by their size.

212. Each interior position of the penthouse and pyramid that might or might not be occupied can be treated as a secondary column in the corresponding exact cover problem. We obtain  $10 \times 2$  solutions for the staircase;  $(223, 286) \times 8$  solutions for the penthouse with hole at the (bottom, middle); and  $32 \times 2$  solutions for the pyramid, of which  $2 \times 2$  have all three holes on the diagonal and  $3 \times 2$  have no adjacent holes.

213. A full simulation of gravity would be quite complex, because pieces can be prevented from tipping with the help of their neighbors above and/or at their side. If we assume a reasonable coefficient of friction and an auxiliary weight at the top, it suffices to define stability by saying that a piece is stable if and only if at least one of its cubies is immediately above either the floor or a stable piece.

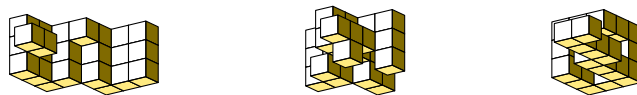
The given shapes can be packed in respectively  $202 \times 2$ ,  $21 \times 2$ ,  $270 \times 2$ ,  $223 \times 8$ , and  $122 \times 2$  ways, of which  $202 \times 2$ ,  $8 \times 2$ ,  $53 \times 2$ ,  $1 \times 8$ , and  $6 \times 2$  are stable. Going from the bottom level to the top, the layers  $\begin{smallmatrix} 1 & 1 & 7 \\ 3 & 3 & 6 \end{smallmatrix}$ ,  $\begin{smallmatrix} 4 & 4 & 7 & 7 \\ 3 & 3 & 6 & 6 \end{smallmatrix}$ ,  $\begin{smallmatrix} 5 & 4 & 1 \\ 3 & 1 & 1 \end{smallmatrix}$  give a decently stable cot; a fragile vulture comes from  $\begin{smallmatrix} 2 & 1 & 7 \\ 3 & 1 & 6 \end{smallmatrix}$ ,  $\begin{smallmatrix} 2 & 4 & 1 & 1 \\ 3 & 3 & 6 & 6 \end{smallmatrix}$ ; a delicate mushroom comes from  $\begin{smallmatrix} 7 & 5 & 2 \\ 3 & 6 & 6 \end{smallmatrix}$ ,  $\begin{smallmatrix} 5 & 2 \\ 3 & 6 \end{smallmatrix}$ ; and a delicate cantilever from  $\begin{smallmatrix} 2 & 2 & 2 \\ 3 & 3 & 6 \end{smallmatrix}$ ,  $\begin{smallmatrix} 5 & 2 \\ 3 & 6 \end{smallmatrix}$ ,  $\begin{smallmatrix} 6 & 2 & 3 \\ 3 & 6 & 6 \end{smallmatrix}$ . The author's cherished set of Skjøde Skjern Soma pieces, made of rosewood and purchased in 1967, includes a small square base that nicely stabilizes both mushroom and cantilever. The vulture needs a book on top.

[The casserole and cot are due respectively to W. A. Kustes and J. W. M. Morgan. The mushroom, which is hollow, is the same as B. L. Schwartz's "penthouse," but turned upside down; John Conway noticed that it then has a unique stable solution. See Martin Gardner, *Knotted Doughnuts* (1986), Chapter 3.]

214. Infinitely many cubies lie behind a wall; but it suffices to consider only the hidden ones whose distance is at most  $27 - v$  from the  $v$  visible ones. For example, if the W-wall has coordinates as in answer 204, we have  $v = 25$  and the two invisible cubies are  $\{332, 331\}$ . We're allowed to use any of  $\{241, 242, 251, 252, 331, 332, 421, 422, 521, 522\}$  at distance 1, and  $\{341, 342, 351, 352, 431, 432, 531, 532, 621, 622\}$  at distance 2. (The stated projection doesn't have left-right symmetry.) The X-wall is similar, but it has  $v = 19$  and potentially  $(9, 7, 6, 3, 3, 2, 1)$  hidden cubies at distances 1 to 7 (omitting cases like 450, which is invisible at distance 2 but "below ground").

Using secondary columns for the optional cubies, we must examine each solution to the exact cover problem and reject those that are disconnected or violate the gravity constraint of exercise 213. Those ground rules yield 282 solutions for the W-wall, 612 for the X-wall, and a whopping 1,130,634 for the cube itself. (These solutions fill respectively 33, 275, and 13842 different sets of cubies.) Here are examples of some of the more exotic shapes that are possible, as seen from behind and below:

gravitationally stable  
 Francillon  
 Hoffmann  
 Mikusinski's Cube  
 Steinhaus  
 pentacubes  
 Reid  
 Sicherman  
 Holy Grail  
 Shindo  
 Neo Diabolical Cube



There also are ten surprising ways to make the cube façade if we allow hidden “underground” cubies: The remarkable construction  $\begin{matrix} \dots & 4\% & 7\% & 33\% \\ \dots & 55 & 2 & 211 \end{matrix}$  raises the entire cube one level *above* the floor, and is gravitationally stable, by exercise 213's criteria! Unfortunately, though, it falls apart — even with a heavy book on top.

[The false-front idea was pioneered by Jean Paul Francillon, whose construction of a fake W-wall was announced in *The SOMA® Addict* 2, 1 (spring 1971).]

**215.** (a) Each of 13 solutions occurs in 48 equivalent arrangements. To remove the symmetry, place piece 7 horizontally, either (i) at the bottom or (ii) in the middle. In case (ii), add a secondary ‘s’ column as in answer 150, and append ‘s’ also to all placements of piece 6 that touch the bottom more than the top. Run time: 400 Kμ.

[This puzzle was number 39 in *Hoffmann's Puzzles Old and New* (1893). Another  $3 \times 3 \times 3$  polycube dissection of historical importance, “Mikusinski's Cube,” was described by Hugo Steinhaus in the 2nd edition of his *Mathematical Snapshots* (1950). That one consists of the ell and the two twist pieces of the Soma cube, plus the pentacubes B, C, and f of exercise 220; it has 24 symmetries and just two solutions.]

(b) Yes: Michael Reid, circa 1995, found the remarkable set



which also makes  $9 \times 3 \times 1$  uniquely(!). George Sicherman carried out an exhaustive analysis of all relevant flat polyominoes in 2016, finding exactly 320 sets that are unique for  $3 \times 3 \times 3$ , of which 19 are unique also for  $9 \times 3 \times 1$ . In fact, one of those 19,



is the long-sought “Holy Grail” of  $3 \times 3 \times 3$  cube decompositions: Its pieces not only have flatness and double uniqueness, they are nested (!!). There's also Yoshiya Shindo's





known as the “Neo Diabolical Cube” (1995); notice that it has 24 symmetries, not 48.

**216.** This piece can be modeled by a polycube with  $20 + 20 + 27 + 3$  cubies, where we want to pack nine of them into a  $9 \times 9 \times 9$  box. Divide that box into 540 primary cells (which must be filled) and 189 secondary cells (which will contain the 27 cubies of the simulated dowels). Answer 200 now yields an exact cover problem with 1536 rows; and Algorithm D needs only 33 Mμ to discover 24 solutions, all equivalent by symmetry. (Or we could modify answer 200 so that all offsets have multiples of 3 in each coordinate; then there would be only 192 rows, and the running time would go down to 8 Mμ.) One packing is  $\begin{matrix} 123 & 567 & 557 \\ 133 & 163 & 867 \\ 443 & 849 & 899 \end{matrix}$ , with dowels at  $\begin{matrix} 010 & 070 & 000 \\ 400 & 520 & 600 \\ 650 & 680 & 600 \end{matrix}$ .

One might be tempted to factor this problem, by first looking at all ways to pack nine solid ell-trominoes into a  $3 \times 3 \times 3$  box. That problem has 5328 solutions, found in about  $5 M\mu$ ; and after removing the 48 symmetries we're left with just 111 solutions, into which we can try to model the holes and dowels. But such a procedure is rather complicated, and it doesn't really save much time, if any.

Ronald Kint-Bruynseels, who designed this remarkable puzzle, also found that it's possible to drill holes in the solid cubies, parallel to the other two, without destroying the uniqueness of the solution(!). [*Cubism For Fun* **75** (2008), 16–19; **77** (2008), 13–18.]


**217.** The straight tetracube  and the square tetracube , together with the size-4 Soma pieces in (30), make a complete set.

We can fix the tee's position in the twin towers, saving a factor of 32; and each of the resulting 40 solutions has just one twist with the tee. Hence there are five inequivalent solutions, and  $5 \times 256$  altogether.

The double claw has  $63 \times 6$  solutions. But the cannon, with  $1 \times 4$  solutions, can be formed in essentially only one way. (*Hint*: Both twists are in the barrel.)

There are no solutions to 'up 3'. But 'up 4' and 'up 5' each have  $218 \times 8$  solutions (related by turning them upside down). Gravitationally, four of those 218 are stable for 'up 5'; the stable solution for 'up 4' is unique, and unrelated to those four.

*References*: Jean Meeus, *JRM* **6** (1973), 257–265; Nob Yoshigahara, *Puzzle World* No. 1 (San Jose: Ishi Press International, 1992), 36–38.

**218.** All but 48 are realizable. The unique “hardest” realizable case, , has  $2 \times 2$  solutions. The “easiest” case is the  $2 \times 4 \times 4$  cuboid, with  $11120 = 695 \times 16$  solutions.

**220.** (a) A, B, C, D, E, F, a, b, c, d, e, f, j, k, l, . . . , z. (It's a little hard to see why reflection doesn't change piece 'l'. In fact, S. S. Besley once patented the pentacubes under the impression that there were 30 different kinds! See *U.S. Patent 3065970* (1962), where Figs. 22 and 23 illustrate the same piece in slight disguise.)

*Historical notes*: R. J. French, in *Fairy Chess Review* **4** (1940), problem 3930, was first to show that there are 23 different pentacube shapes, if mirror images are considered to be identical. The full count of 29 was established somewhat later by F. Hansson and others [*Fairy Chess Review* **6** (1948), 141–142]; Hansson also counted the  $35 + 77 = 112$  mirror-inequivalent hexacubes. Complete counts of hexacubes (166) and heptacubes (1023) were first established soon afterwards by J. Niemann, A. W. Baillie, and R. J. French [*Fairy Chess Review* **7** (1948), 8, 16, 48].

(b) The cuboids  $1 \times 3 \times 20$ ,  $1 \times 4 \times 15$ ,  $1 \times 5 \times 12$ , and  $1 \times 6 \times 10$  have of course already been considered. The  $2 \times 3 \times 10$  and  $2 \times 5 \times 6$  cuboids can be handled by restricting X to the bottom upper left, and sometimes also restricting Z, as in answers 150 and 152; we obtain 12 solutions (in  $350 M\mu$ ) and 264 solutions (in  $2.5 G\mu$ ), respectively.

The  $3 \times 4 \times 5$  cuboid is more difficult. Without symmetry-breaking, we obtain  $3940 \times 8$  solutions in about  $200 G\mu$ . To do better, notice that X can appear in 11 essentially different positions:  $(1 + 1^*)(1 + 1^*)$  in a  $4 \times 5$  plane,  $2^* + 2^{**}$  in a  $3 \times 5$  plane, and  $2^* + 1^{**}$  in a  $3 \times 4$  plane, where '\*' denotes a case where symmetry needs to be broken down further because X is fixed by some symmetry. With 11 separate runs we can find  $(923 + 558/2 + 402/2 + 376/4) + (1268/2 + 656/2 + 420/4 + 752/4) + (1480/2 + 720/2 + 352/4) = 3940$  solutions, in  $4.9 + 3.3 + 3.1 + 2.4 + \dots + 2.1 \approx 50 G\mu$ .

[The fact that solid pentominoes will fill these cuboids was first demonstrated by D. Nixon and F. Hansson, *Fairy Chess Review* **6** (1948), problem 7560 and page 142.

factor  
solid ell-trominoes  
L-cube puzzle  
Kint-Bruynseels  
Meeus  
Yoshigahara  
Besley  
Patent  
French  
Hansson  
hexacubes  
heptacubes  
Niemann  
Baillie  
symmetry-breaking  
Nixon  
Hansson

Exact enumeration was first performed by C. J. Bouwkamp in 1967; see *J. Combinatorial Theory* **7** (1969), 278–280, and *Indagationes Math.* **81** (1978), 177–186.]

(c) Almost *any* subset of 25 pentacubes can probably do the job. But a particularly nice one is obtained if we simply omit o, q, s, and y, namely those that don't fit in a  $3 \times 3 \times 3$  box. R. K. Guy proposed this subset in *Nabla* **7** (1960), 150, although he wasn't able to pack a  $5 \times 5 \times 5$  at that time. The same idea occurred independently to J. E. Dorie, who trademarked the name “Dorian cube” [*U.S. Trademark 1,041,392* (1976)].

An amusing way to form such a cube is to make 5-level prisms in the shapes of the P, Q, R, U, and X pentominoes, using pieces {a, e, j, m, w}, {f, k, l, p, r}, {A, d, D, E, n}, {c, C, F, u, v}, {b, B, t, x, z}; then use the packing in answer 151(!). This solution can be found with six very short runs of Algorithm D, taking only 300 megamems overall.

Another nice way, due to Torsten Sillke, is more symmetrical: There are 70,486 ways to partition the pieces into five sets of five that allow us to build an X-prism in the center (with piece x on top), surrounded by four P-prisms.

One can also assemble a Dorian cube from five cuboids, using one  $1 \times 3 \times 5$ , one  $2 \times 2 \times 5$ , and three  $2 \times 3 \times 5$ s. Indeed, there are zillions more ways, too many to count.

**221.** (a) Make an exact cover problem in which a and A, b and B, . . . , f and F are required to be in symmetrical position; there are respectively (86, 112, 172, 112, 52, 26) placements for such 10-cubie “super-pieces.” Furthermore, the author decided to force piece m to be in the middle of the top wall. Solutions were found immediately! So piece x was placed in the exact center, as an additional desirable constraint. Then there were exactly 20 solutions; the one below has also n, o, and u in mirror-symmetrical locations.

(b) The super-pieces now have (59, 84, 120, 82, 42, 20) placements; the author also optimistically forced j, k, and m to be symmetrical about the diagonal, with m in the northwest corner. A long and apparently fruitless computation (34.3 teramems) ensued; but — hurrah — two closely related solutions were discovered at the last minute.

(c) This computation, due to Torsten Sillke [see *Cubism For Fun* **27** (1991), 15], goes much faster: The quarter-of-a-box shown here can be packed with seven non-x pentacubes in 55356 ways, found in 1.3 Gμ. As in answer 177, this yields a new exact cover problem, with 33412 different rows. Then 11.8 Gμ more computation discovers seven suitable partitions into four sets of seven, one of which is illustrated here.

Bouwkamp  
Guy  
Dorie  
Dorian cube  
pentominoes  
Sillke  
partition  
author  
geek art  
Sillke



|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| l | l | l | q | q | q | q |
| l | o | o | o | o | o | q |
| f | f | u | u | F | F |   |
| D | f | u | m | u | F | d |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| m | o | o | o | o | o | s |
| m | m | x | q | q | q | q |
| m | x | x | x | b | b | b |
| r | n | x | e | e | b | a |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| v | E | z | z | t | A | A |
| E | E | z | s | t | t | t |
| E | z | z | s | t | F | F |
| f | f | s | s | a | F | k |

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| l | l | f | D | D | m | m | d | d | d | F | q | q |   |   |
| l | f | f | C | C | D | D | m | d | d | c | c | F | F | r |
| v | v | v | B | C | C | C | x | c | c | c | b | r | r | r |
| v | w | B | B | B | x | x | b | b | b | b | b | r | r | z |
| v | w | w | A | A | A | A | x | a | a | a | a | z | z | z |
| k | k | w | w | E | E | A | n | a | e | e | y | z | j | j |
| k | k | s | s | s | E | E | n | e | e | y | y | y | j | j |

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| m | m | m | r | r | n | e | e | a | a | a | a | a | b | q | s |
| t | m | r | r | n | n | n | e | a | D | D | D | b | q | s |   |
| t | t | t | r | E | p | p | p | v | C | C | D | D | s | s |   |
| t | w | w | E | E | p | p | v | F | C | C | C | s | z |   |   |
| v | w | B | E | A | A | v | v | v | F | F | F | z | z | z |   |
| w | l | B | B | A | d | c | f | f | k | k | F | z | j | j |   |
| l | l | B | A | A | d | c | c | f | k | k | k | u | u | j |   |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| e | C | q | b | b | c | d |
| e | q | b | r | n | n |   |
| p | e | q | r | r | r | l |
| p | q | q | r | l | l | l |

(a)
(b)
(c)

**222.** As in previous exercises, the key is to reduce the search space drastically, by asking for solutions of a special form. (Such solutions aren't unlikely, because pentacubes are so versatile.) Here we can break the given shape into four pieces: Three modules of size  $3^3 + 2^3$  to be packed with seven pentacubes, and one of size  $4^3 - 3 \cdot 2^3$  to be packed with eight pentacubes. The first problem has 13,587,963 solutions, found with  $2.5 T\mu$  of computation; they involve 737,695 distinct sets of seven pentacubes. The larger problem has 15,840 solutions, found with  $400 M\mu$  and involving 2075 sets of eight. Exactly covering those sets yields 1,132,127,589 suitable partitions; the first one found,  $\{a, A, b, c, j, q, t, y\}$ ,  $\{B, C, d, D, e, k, o\}$ ,  $\{E, f, l, n, r, v, x\}$ ,  $\{F, m, p, s, u, w, z\}$ , works fine. (We need only one partition, so we needn't have computed more than a thousand or so solutions to the smaller problem.)



Künzell  
Farhi  
X pentomino



*Pentacubes galore:* Since the early 1970s, Ekkehard Künzell and Sivy Farhi have independently published booklets that contain hundreds of solved pentacube problems.

**239.** First we realize that every edge of the square must touch at least three pieces; hence the pieces must in fact form a  $3 \times 3$  arrangement. Consequently any correct placement would also lead to a placement for nine pieces of sizes  $(17 - k) \times (20 - k)$ ,  $\dots$ ,  $(24 - k) \times (25 - k)$ , into a  $(65 - 3k) \times (65 - 3k)$  box. Unfortunately, however, if we try, say,  $k = 16$ , Algorithm D quickly gives a contradiction.

But aha—a closer look shows that the pieces have *rounded corners*. Indeed, there's just enough room for pieces to get close enough together so that, if they truly were rectangles, they'd make a  $1 \times 1$  overlap at a corner.

So we can take  $k = 13$  and make nine pieces of sizes  $4 \times 7, \dots, 11 \times 12$ , consisting of rectangles *minus* their corners. Those pieces can be packed into a  $26 \times 26$  square, as if they were polyominoes (see exercise 140), but with the individual cells of the enclosing rectangle treated as secondary columns because they needn't be covered. (Well, the eight cells adjacent to corners can be primary.) We can save a factor of 8 by insisting that the  $9 \times 11$  piece appear in the upper left quarter, with its long side horizontal.

Algorithm D solves that problem in 620 gigamems—but it finds 43 solutions, most of which are unusable, because the missing corners give too much flexibility. The unique correct solution is easily identified, because a  $1 \times 1$  overlap between rectangles in one place must be compensated by a  $1 \times 1$  empty cell between rectangles in another. The resulting cross pattern (like the X pentomino) occurs in just one of the 43.

**240.** Let there be  $mn$  primary columns  $p_{ij}$  for  $0 \leq i < m$  and  $0 \leq j < n$ , one for each cell that should be covered exactly once. Also introduce  $m$  primary columns  $x_i$  for  $0 \leq i < m$ , as well as  $n$  primary columns  $y_j$  for  $0 \leq j < n$ . The exact cover problem has  $\binom{m+1}{2} \cdot \binom{n+1}{2}$  rows, one for each subrectangle  $[a..b) \times [c..d)$  with  $0 \leq a < b \leq m$  and  $0 \leq c < d \leq n$ . The row for that subrectangle contains  $2 + (b - a)(d - c)$  columns, namely  $x_a, y_c$ , and  $p_{ij}$  for  $a \leq i < b, c \leq j < d$ . The solutions correspond to reduced decompositions when we insist that each  $x_i$  be covered  $[1..n]$  times and that each  $y_j$  be covered  $[1..m]$  times. (We can save a little time by omitting  $x_0$  and  $y_0$ .)

The  $3 \times 5$  problem has 20165 solutions, found in  $18 M\mu$ . They include respectively (1071, 3816, 5940, 5266, 2874, 976, 199, 22, 1) cases with (7, 8,  $\dots$ , 15) subrectangles.


**241.** The minimum is  $m + n - 1$ . Proof (by induction): The result is obvious when  $m = 1$  or  $n = 1$ . Otherwise, given a decomposition into  $t$  subrectangles,  $k \geq 1$  of them must be confined to the  $n$ th column. If two of those  $k$  are contiguous, we can combine them; the resulting dissection of order  $t - 1$  reduces to either  $(m - 1) \times n$  or  $m \times n$ , hence  $t - 1 \geq (m - 1) + n - 1$ . On the other hand if none of them are contiguous, the reduction of the first  $n - 1$  columns is  $m \times (n - 1)$ ; hence  $t \geq m + (n - 1) - 1 + k$ .

Close examination of this proof shows that a reduced decomposition has minimum order  $t$  if and only if its boundary edges form  $m - 1$  horizontal lines and  $n - 1$  vertical lines that don't cross each other. (In particular, the "tatami condition" is satisfied; see exercise 7.1.4–215.)

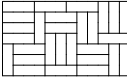
tatami condition  
Scherer  
Graham

**242.** Simply remove the offending subrectangles, so that the cover problem has only  $\binom{m+1}{2} - 1$  rows. Now there are 13731  $3 \times 5$  solutions, found in 11  $M\mu$ , and (410, 1974, 3830, 3968, 2432, 900, 194, 22, 1) cases with (7, 8, ..., 15) subrectangles.

**243.** Introduce additional primary columns  $X_i$  for  $0 < i < m$ , to be covered  $[1..n-1]$  times, as well as  $Y_j$  for  $0 < j < n$ , to be covered  $[1..m-1]$  times. Then add columns  $X_i$  for  $a < i < b$  and  $Y_j$  for  $c < j < d$  to the constraint for subrectangle  $[a..b] \times [c..d]$ .

Now the  $3 \times 5$  problem has just 216 solutions, found in 1.9 megamems. They include (66, 106, 44) instances with (7, 8, 9) subrectangles. Just two of the solutions are symmetric under left-right reflection, namely  and its top-bottom reflection.

**244.** We can delete non-tromino options from the exact cover problem, thereby getting all faultfree tromino tilings that are reduced. If we also delete the constraints on  $x_i$  and  $y_j$ —and if we require  $X_i$  and  $Y_j$  to be covered  $[1..n]$  and  $[1..m]$  times instead of  $[1..n-1]$  and  $[1..m-1]$ —we obtain *all* of the  $m \times n$  faultfree tromino tilings.

It is known that such nontrivial tilings exist if and only if  $m, n \geq 7$  and  $mn$  is a multiple of 3. [See K. Scherer, *JRM* **13** (1980), 4–6; R. L. Graham, *The Mathematical Gardner* (1981), 120–126.] So we look at the smallest cases in order of  $mn$ : When  $(m, n) = (7, 9), (8, 9), (9, 9), (7, 12), (9, 10)$ , we get respectively (32, 32), (48, 48), (16, 16), (706, 1026), (1080, 1336) solutions. Hence the assertion is false; a smallest counterexample is shown. 

**247.** Augment the exact cover problem of answer 242 by introducing  $\binom{m+1}{2} + \binom{n+1}{2} - 2$  secondary columns  $x_{ab}$  and  $y_{cd}$ , for  $0 < a < b \leq m$  and  $0 \leq c < d \leq n$ ,  $(a, b) \neq (0, m)$ ,  $(c, d) \neq (0, n)$ . Include column  $x_{ab}$  and  $y_{cd}$  in the row for subrectangle  $[a..b] \times [c..d]$ . Furthermore, cover  $x_i$   $[1..m-i]$  times, not  $[1..n]$ ; cover  $y_j$   $[1..n-j]$  times.

**248.** The hint follows because  $[a..b] \times [0..d]$  cannot coexist motleywise with its left-right reflection  $[a..b] \times [n-d..n]$ . Thus we can forbid half of the solutions.

Consider, for example, the case  $(m, n) = (7, 7)$ . Every solution will include  $x_{67}$  with some  $y_{cd}$ . If it's  $y_{46}$ , say, left-right reflection would produce an equivalent solution with  $y_{13}$ ; therefore we disallow the option  $(a, b, c, d) = (6, 7, 4, 6)$ . Similarly, we disallow  $(a, b, c, d) = (6, 7, c, d)$  whenever  $7 - d < c$ .

Reflection doesn't change the bottom-row rectangle when  $c + d = 7$ , so we haven't broken all the symmetry. But we can complete the job by looking also at the top-row rectangle, namely the option where  $x_{01}$  occurs with some  $y_{c'd'}$ . Let's introduce new secondary columns  $t_1, t_2, t_3$ , and include  $t_c$  in the option that has  $x_{67}$  with  $y_{c(7-c)}$ . Then we include  $t_1, t_2$ , and  $t_3$  in the option that has  $x_{01}$  with  $y_{c'd'}$  for  $c' + d' > 7$ . We also add  $t_1$  to the option with  $x_{01}$  and  $y_{25}$ ; and we add both  $t_1$  and  $t_2$  to the option with  $x_{01}$  and  $y_{34}$ . This works beautifully, because no solution can have  $c = c'$  and  $d = d'$ .

In general, we introduce new secondary columns  $t_c$  for  $1 \leq c < n/2$ , and we disallow all options  $x_{(m-1)m} y_{cd}$  for which  $c + d > n$ . We put  $t_c$  into the option that contains  $x_{(m-1)m} y_{c(n-c)}$ . We put  $t_1$  thru  $t_{\lfloor (n-1)/2 \rfloor}$  into the option that contains  $x_{01} y_{c'd'}$  when  $c' + d' > n$ . And we put  $t_1$  thru  $t_{c'-1}$  into the option that contains  $x_{01} y_{c'(n-c')}$ . (Think about it.)

For example, when  $m = n = 7$  there now are 717 options instead of 729, 57 secondary columns instead of 54. We now find 352546 solutions after only 13.2

gigamems of computation, instead of 705092 solutions after 26.4. The search tree now has just 7.8 meganodes instead of 15.7.

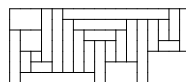
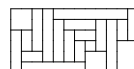
faultfree

(It's tempting to believe that the same idea will break top-bottom symmetry too. But that would be fallacious: Once we've fixed attention on the bottommost row while breaking left-right symmetry, we've lost all symmetry between top and bottom.)

**249.** From any  $m \times n$  dissection of order  $t$  we get two  $(m+2) \times (n+2)$  dissections of order  $t+4$ , by enclosing it within two  $1 \times m$  tiles and two  $1 \times n$  tiles. So the claim follows by induction and the examples in exercise 247, together with a  $5 \times 6$  example of order 10 —of which there are 8 symmetrical instances such as the one shown here. (This construction is faultfree, and it's also “tight”: The order of every  $m \times n$  dissection is at least  $m+n-1$ , by exercise 241.)



**250.** All 214 of the  $5 \times 7$  motley dissections have order 11, which is far short of  $\binom{6}{2} - 1 = 14$ ; and there are no  $5 \times 8$ s,  $5 \times 9$ s, or  $5 \times 10$ s. Surprisingly, however, 424 of the 696 dissections of size  $6 \times 12$  do have the optimum order 20, and  $7 \times 17$  dissections with the optimum order 27 also exist. Examples of these remarkable patterns are shown. (The case  $m=7$  is still not fully explored except for small  $n$ . For example, the total number of motley  $7 \times 17$  dissections is unknown, and no  $7 \times 18$ s have yet been found. If we restrict attention to *symmetrical* dissections, the maximum orders for  $5 \leq m \leq 8$  are 11 ( $5 \times 7$ ); 19 ( $6 \times 11$ ); 25 ( $7 \times 15$ ); 33 ( $8 \times 21$ .)



**252.** The basic idea is to combine complementary options into a single option whenever possible. More precisely: (i) If  $a+b=m$  and  $c+d=n$ , we retain the option as usual; it is self-complementary. (ii) Otherwise, if  $a+b=m$  or  $c+d=n$ , reject the option; merging would be non-motley. (iii) Otherwise, if  $a+b > m$ , reject the option; we've already considered its complement. (iv) Otherwise, if  $b=1$  and  $c+d < n$ , reject the option; its complement is illegal. (v) Otherwise, if  $b > m/2$  and  $c < n/2$  and  $d > n/2$ , reject the option; it intersects its complement. (vi) Otherwise merge the option with its complement. For example, when  $(m,n) = (4,5)$ , case (i) arises when  $(a,b,c,d) = (1,3,2,3)$ ; the option is ' $x_1 y_2 p_{12} p_{22} x_{13} y_{23}$ ' as in answer 248. Case (ii) arises when  $(a,b,c,d) = (1,3,0,1)$ . Case (iii) arises when  $(a,b) = (2,3)$ . Case (iv) arises when  $(a,b,c,d) = (0,1,0,1)$ ; the complement  $(3,4,4,5)$  isn't a valid subrectangle in answer 248. Case (v) arises when  $(a,b,c,d) = (1,3,1,3)$ ; cells  $p_{22}$  and  $p_{23}$  occur also in the complement  $(1,3,2,4)$ . And case (vi) arises when  $(a,b,c,d) = (0,1,4,5)$ ; the merged option is the union of ' $x_0 y_4 p_{04} x_{01} y_{45} t_1 t_2$ ' and ' $x_3 y_0 p_{30} x_{34} y_{01}$ '. (Well,  $x_0$  and  $y_0$  are actually omitted, as suggested in answer 240.)

Size  $8 \times 16$  has (6703, 1984, 10132, 1621, 47) solutions, of orders (26, ..., 30).

**253.** (a) Again we merge compatible options, as in answer 252. But now  $(a,b,c,d) \rightarrow (c,d,n-b,n-a) \rightarrow (n-b,n-c,n-b,n-a) \rightarrow (n-b,n-a,c,d)$ , so we typically must merge *four* options instead of two. The rules are: Reject if  $a=n-1$  and  $c+d > n$ , or  $c=n-1$  and  $a+b < n$ , or  $b=1$  and  $c+d < n$ , or  $d=1$  and  $a+b > n$ . Also reject if  $(a,b,c,d)$  is lexicographically greater than any of its three successors. But accept, without merging, if  $(a,b,c,d) = (c,d,n-b,n-a)$ . Otherwise reject if  $b > c$  and  $b+d > n$ , or if  $b > n/2$  and  $c < n/2$  and  $d > n/2$ , because of intersection. Also reject if  $a+b=n$  or  $c+d=n$ , because of the motley condition. Otherwise merge four options into one.

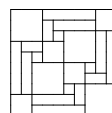
For example, the merged option when  $n=4$  and  $(a,b,c,d) = (0,1,2,4)$  is ' $x_0 y_2 p_{02} p_{03} x_{01} y_{24} t_1 x_2 y_3 p_{23} p_{33} x_{24} y_{34} x_3 y_0 p_{30} p_{31} x_{34} y_{24} p_{00} p_{10} x_{02} y_{01}$ ', except that  $x_0$  and  $y_0$  are omitted. Notice that it's important not to include an item  $x_i$  or  $y_j$  twice, when merging in cases that have  $a=c$  or  $b=d$  or  $a=n-d$  or  $b=n-c$ .

(b) With bidirectional symmetry it's possible to have  $(a, b, c, d) = (c, d, a, b)$  but  $(a, b, c, d) \neq (n - d, n - c, n - b, n - a)$ , or vice versa. Thus we'll sometimes merge two options, we'll sometimes merge four, and we'll sometimes accept without merging. In detail: Reject if  $a = n - 1$  and  $c + d > n$ , or  $c = n - 1$  and  $a + b > n$ , or  $b = 1$  and  $c + d < n$ , or  $d = 1$  and  $a + b < n$ . Also reject if  $(a, b, c, d)$  is lexicographically greater than any of its three successors. But accept, without merging, if  $a = c = n - d = n - b$ . Otherwise reject if  $b > c$  or  $b > n - d$  or  $a + b = n$  or  $c + d = n$ . Otherwise merge two or four distinct options into one.

pinwheel  
factored  
backtrack  
van Hertog  
Gardner  
Cutler

Examples when  $n = 4$  are:  $\langle x_1 y_1 p_{11} p_{12} p_{21} p_{22} x_{13} y_{13} \rangle$ ;  $\langle x_0 y_3 p_{03} x_{01} y_{34} t_1 x_3 y_0 p_{30} x_{34} y_{01} \rangle$ ;  $\langle x_0 y_2 p_{02} x_{34} y_{23} t_1 x_1 y_3 p_{13} x_{12} y_{34} x_3 y_1 p_{31} x_{34} y_{12} x_2 y_0 p_{20} x_{23} y_{01} \rangle$ ; again with  $x_0$  and  $y_0$  suppressed.

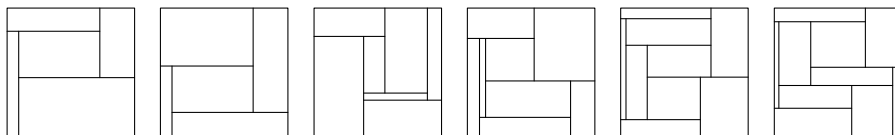
(c) The unique solution for  $n = 10$  is shown. [The total number of such patterns for  $n = (10, 11, \dots, 16)$  turns out to be  $(1, 0, 3, 6, 28, 20, 354)$ . All 354 of the  $16 \times 16$  solutions are found in only 560 megamems; they have orders 34, 36, and 38–44. Furthermore the number of  $n \times n$  motley dissections with symmetry (a), for  $n = (3, 4, 5, \dots, 16)$ , turns out to be  $(1, 0, 2, 2, 8, 18, 66, 220, 1024, 4178, 21890, 102351, 598756, 3275503)$ , respectively. Algorithm M needs 3.3 teramems when  $n = 16$ ; those patterns have orders  $4k$  and  $4k + 1$  for  $k = 8, 9, \dots, 13$ .]



**255.** The reduction of a perfectly decomposed rectangle is a motley dissection. Thus we can find all perfectly decomposed rectangles by “unreducing” all motley dissections.

For example, the only motley dissection of order 5 is the  $3 \times 3$  pinwheel. Thus the perfectly decomposed  $m \times n$  rectangles of order 5 with integer dimensions are the positive integer solutions to  $x_1 + x_2 + x_3 = m$ ,  $y_1 + y_2 + y_3 = n$  such that the ten values  $x_1, x_2, x_3, x_1 + x_2, x_2 + x_3, y_1, y_2, y_3, y_1 + y_2, y_2 + y_3$  are distinct. Those equations are readily factored into two easy backtrack problems, one for  $m$  and one for  $n$ , each producing a list of five-element sets  $\{x_1, x_2, x_3, x_1 + x_2, x_2 + x_3\}$ ; then we search for all pairs of disjoint solutions to the two subproblems. In this way we quickly see that the equations have just two essentially different solutions when  $m = n = 11$ , namely  $(x_1, x_2, x_3) = (1, 7, 3)$  and  $(y_1, y_2, y_3) = (2, 4, 5)$  or  $(5, 4, 2)$ . The smallest perfectly decomposed squares of order 5 are therefore have size  $11 \times 11$ , and there are two of them (shown below); they were discovered by M. van Hertog, who reported them to Martin Gardner in May 1979. (Incidentally, a  $12 \times 12$  square can also be perfectly decomposed.)

There are no solutions of order 6. Those of orders 7, 8, 9, 10 must come respectively from motley dissections of sizes  $4 \times 4$ ,  $4 \times 5$ ,  $5 \times 5$ , and  $5 \times 6$ . By looking at them all, we find that the smallest  $n \times n$  squares respectively have  $n = 18, 21, 24$ , and 28. Each of the order- $t$  solutions shown here uses rectangles of dimensions  $\{1, 2, \dots, 2t\}$ , except in the case  $t = 9$ : There's a *unique* perfectly decomposed  $24 \times 24$  square of order 9, and it uses the dimensions  $\{1, 2, \dots, 17, 19\}$ .



[W. H. Cutler introduced perfectly decomposed rectangles in *JRM* **12** (1979), 104–111.]

**256.** (a) False (but close). Let the individual dimensions be  $z_1, \dots, z_{2t}$ , where  $z_1 \leq \dots \leq z_{2t}$ . Then we have  $\{w_1, h_1\} = \{z_1, z_{2t}\}$ ,  $\{w_2, h_2\} = \{z_2, z_{2t-1}\}$ ,  $\dots$ ,  $\{w_t, h_t\} = \{z_t, z_{t+1}\}$ ; consequently  $z_1 < \dots < z_t \leq z_{t+1} < \dots < z_{2t}$ . But  $z_t = z_{t+1}$  is possible.



(b) False (but close). If the reduced rectangle is  $m \times n$ , one of its subrectangles might be  $1 \times n$  or  $m \times 1$ ; a motley dissection must be *strict*.

(c) True. Label the rectangles  $\{a, b, c, d, e\}$  as shown. Then there's a contradiction:  $w_b > w_d \iff w_e > w_c \iff h_e < h_c \iff h_d < h_b \iff w_b < w_d$ .



(d) The order can't be 6, because the reduction would then have to be a pinwheel together with a  $1 \times 3$  subrectangle, and the argument in (c) would still apply. Thus the order must be 7, and we must show that the second dissection of exercise 247 doesn't work. Labeling its regions  $\{a, \dots, g\}$  as shown, we have  $h_d > h_a$ ; hence  $w_a > w_d$ . Also  $h_e > h_b$ ; so  $w_b > w_e$ . Oops:  $w_f > w_g$  and  $h_f > h_g$ .



In the other motley  $4 \times 4$  dissection of exercise 247 we obviously have

$$w_4 < w_5, \quad w_4 < w_6, \quad w_6 < w_7, \quad h_4 < h_3, \quad h_3 < h_1, \quad h_4 < h_2;$$

therefore  $h_4 > h_5, h_4 > h_6, h_6 > h_7, w_4 > w_3, w_3 > w_1, w_4 > w_2$ . Now  $h_5 < h_6 \iff w_5 > w_6 \iff w_2 > w_3 \iff h_2 < h_3 \iff h_6 + h_7 < h_5$ . Hence  $h_5 < h_6$  implies  $h_5 > h_6$ ; we must have  $h_5 > h_6$ , thus also  $h_2 > h_3$ . Finally  $h_2 < h_1$ , because  $h_7 < h_5$ .

(e) The condition is clearly necessary. Conversely, given any such pair of solutions, the rectangles  $w_1 \times ah_1, \dots, w_t \times ah_t$  are incomparable for all large enough  $\alpha$ .

[Many questions remain unanswered: Is it NP-hard to determine whether or not a given motley dissection supports an incomparable dissection? Is there a motley dissection that supports incomparable dissections having two different permutation labels? Can a *symmetric* motley dissection ever support an incomparable dissection?]

**257.** (a) By exercise 256(d), the widths and heights must satisfy

$$\begin{aligned} w_5 &= w_2 + w_4, & w_6 &= w_3 + w_4, & w_7 &= w_1 + w_3 + w_4; \\ h_3 &= h_4 + h_5, & h_2 &= h_4 + h_6 + h_7, & h_1 &= h_4 + h_5 + h_6. \end{aligned}$$

To prove the hint, consider answer 256(a). Each  $z_j$  for  $1 \leq j \leq t$  can be either  $w$  or  $h$ ; then  $z_{2t+1-j}$  is the opposite. So there are  $2^t$  ways to shuffle the  $w$ 's and  $h$ 's together.

For example, suppose all the  $h$ 's come first, namely  $h_7 < \dots < h_1 \leq w_1 < \dots < w_t$ :

$$\begin{aligned} 1 &\leq h_7, & h_7 + 1 &\leq h_6, & h_6 + 1 &\leq h_5, & h_5 + 1 &\leq h_4, & h_4 + 1 &\leq h_4 + h_5, \\ &h_4 + h_5 + 1 &\leq h_4 + h_6 + h_7, & h_4 + h_6 + h_7 + 1 &\leq h_4 + h_5 + h_6, \\ &h_4 + h_5 + h_6 &\leq w_1, & w_1 + 1 &\leq w_2, & w_2 + 1 &\leq w_3, & w_3 + 1 &\leq w_4, \\ &w_4 + 1 &\leq w_2 + w_4, & w_2 + w_4 + 1 &\leq w_3 + w_4, & w_3 + w_4 + 1 &\leq w_1 + w_3 + w_4. \end{aligned}$$

The least perimeter in this case is the smallest value of  $w_1 + w_2 + w_3 + w_4 + h_7 + h_6 + h_5 + h_4$ , subject to those inequalities; and one easily sees that the minimum is 68, achieved when  $h_7 = 2, h_6 = 3, h_5 = 4, h_4 = 5, w_1 = 12, w_2 = 13, w_3 = 14, w_4 = 15$ .

Consider also the alternating case,  $w_1 < h_7 < w_2 < h_6 < w_3 < h_5 < w_4 \leq h_4 < w_2 + w_4 < h_4 + h_5 < w_3 + w_4 < h_4 + h_6 + h_7 < w_1 + w_3 + w_4 < h_4 + h_5 + h_6$ . This case turns out to be infeasible. (Indeed, any case with  $h_6 < w_3 < h_5$  requires  $h_4 + h_5 < w_3 + w_4$ , hence it needs  $h_4 < w_4$ .) Only 52 of the 128 cases are actually feasible.

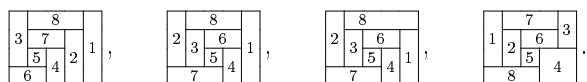
Each of the 128 subproblems is a classic example of linear programming, and a decent LP solver will resolve it almost instantly. The minimum perimeter with seven subrectangles is 35, obtained uniquely in the case  $w_1 < w_2 < w_3 < h_7 < h_6 < h_5 < h_4 \leq w_4 < w_5 < w_6 < w_7 < h_3 < h_2 < h_1$  (or the same case with  $w_4 \leftrightarrow h_4$ ) by setting  $w_1 = 1, w_2 = 2, w_3 = 3, h_7 = 4, h_6 = 5, h_5 = 6, h_4 = w_4 = 7$ . The next-best case has perimeter 43. In one case the best-achievable perimeter is 103!

To find the smallest square, we simply add the constraint  $w_1 + w_2 + w_3 + w_4 = h_7 + h_6 + h_5 + h_4$  to each subproblem. Now only four of the 128 are feasible. The minimum side, 34, occurs uniquely when  $(w_1, w_2, w_3, w_4, h_7, h_6, h_5, h_4) = (3, 7, 10, 14, 6, 8, 9, 11)$ .

strict  
NP-hard  
linear programming

(b) With eight subrectangles the reduced pattern is  $4 \times 5$ . We can place a  $4 \times 1$  column at the right of either the  $4 \times 4$  pattern or its transpose; or we can use one of the first two  $4 \times 5$ 's in exercise 247. (The other six patterns can be ruled out, using arguments similar to those of answer 256.) The labeled diagrams are

integer programming  
Yao  
Reingold  
Sands  
Jepsen  
Nuij



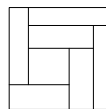
For each of these four choices there are 256 easy subproblems to consider. The best perimeters are respectively (44, 44, 44, 56); the best square sizes are respectively — and surprisingly — (27, 36, 35, 35). [With eight subrectangles we can dissect a significantly smaller square than we can with seven! Furthermore, no smaller square can be incomparably dissected, integerwise, because nine subrectangles would be too many.] One way to achieve perimeter 44 is with  $(w_1, w_2, w_3, w_4, w_5, h_8, h_7, h_6, h_4) = (4, 5, 6, 7, 8, 1, 2, 3, 8)$  in the third diagram. The only way to achieve a square of side 27 is with  $(w_1, w_2, w_3, w_4, w_5, h_8, h_7, h_6, h_4) = (1, 3, 5, 7, 11, 4, 6, 8, 9)$  in the first diagram.

These linear programs usually have integer solutions; but sometimes they don't. For example, the optimum perimeter for the second diagram in the case  $h_8 < h_7 < w_1 < h_6 < w_2 < w_3 < w_4 < h_5$  turns out to be  $97/2$ , achievable when  $(w_1, w_2, w_3, w_4, w_5, h_8, h_7, h_6, h_4) = (7, 11, 13, 15, 17, 3, 5, 9, 17)/2$ . The minimum rises to 52, if we restrict to integer solutions, achieved by  $(w_1, w_2, w_3, w_4, w_5, h_8, h_7, h_6, h_4) = (4, 6, 7, 8, 9, 1, 3, 5, 9)$ .

[The theory of incomparable dissections was developed by A. C. C. Yao, E. M. Reingold, and B. Sands in *JRM* **8** (1976), 112–119. For generalizations to three dimensions, see C. H. Jepsen, *Mathematics Magazine* **59** (1986), 283–292.]

**258.** This is an incomparable dissection in which exercise 256(d) applies. Let's try first to solve the equations  $a(x+y+z) = bx = c(w+x) = d(w+x+y) = (a+b)w = (b+c)y = (b+c+d)z = 1$ , by setting  $b = x = 1$ . We find successively  $c = 1/(w+1)$ ,  $a = (1-w)/w$ ,  $y = (w+1)/(w+2)$ ,  $d = (w+1)/(w(w+2))$ ,  $z = (w+1)(w+3)/((w+2)(w+4))$ . Therefore  $x+y+z-1/a = (2w+3)(2w^2+6w-5)/((w-1)(w+2)(w+4))$ , and we must have  $2w^2+6w = 5$ . The positive root of this quadratic is  $w = (\sqrt{-3})/2$ , where  $\sqrt{\phantom{x}} = \sqrt{19}$ .

Having decomposed the rectangle  $(a+b+c+d) \times (w+x+y+z)$  into seven different rectangles of area 1, we normalize it, dividing  $(a, b, c, d)$  by  $a+b+c+d = \frac{7}{15}(\sqrt{+1})$  and dividing  $(w, x, y, z)$  by  $w+x+y+z = \frac{5}{6}(\sqrt{-1})$ . This gives the desired tiling (shown), with rectangles of dimensions  $\frac{1}{14}(7-\sqrt{+}) \times \frac{1}{15}(7+\sqrt{+})$ ,  $\frac{5}{42}(-1+\sqrt{+}) \times \frac{1}{15}(1+\sqrt{+})$ ,  $\frac{5}{21} \times \frac{3}{5}$ ,  $\frac{1}{21}(8-\sqrt{+}) \times \frac{1}{15}(8+\sqrt{+})$ ,  $\frac{1}{21}(8+\sqrt{+}) \times \frac{1}{15}(8-\sqrt{+})$ ,  $\frac{5}{42}(1+\sqrt{+}) \times \frac{1}{15}(-1+\sqrt{+})$ ,  $\frac{1}{14}(7+\sqrt{+}) \times \frac{1}{15}(7-\sqrt{+})$ .



[See W. A. A. Nuij, *AMM* **81** (1974), 665–666. To get eight different rectangles of area  $1/8$ , we can shrink one dimension by  $7/8$  and attach a rectangle  $(1/8) \times 1$ . Then to get nine of area  $1/9$ , we can shrink the *other* dimension by  $8/9$  and attach a  $(1/9) \times 1$  sliver. And so on. The eight-rectangle problem also has two other solutions, supported by the third and fourth  $4 \times 5$  patterns in exercise 257(b).]

**260.** Let the back corner in the illustration be the point 777, and write just ‘*abcdef*’ instead of  $[a..b] \times [c..d] \times [e..f]$ . The subcuboids are 670517 (270601) 176705 (012706) 051767 (060127), 561547 (260312) 475615 (122603) 154756 (031226), 351446 (361324) 463514 (243613) 144635 (132436), 575757 (020202), 454545 (232323)—with the 11 mirror images in parentheses—plus the central cubie 343434. Notice that each of the 28 possible intervals is used in each dimension, except  $[0..4)$ ,  $[1..6)$ ,  $[2..5)$ ,  $[3..7)$ ,  $[0..7)$ .

Hilbert  
KIM  
Gardner  
puzzle  
23 and me  
author

*I started from a central cube and built outwards, all the while staring at the 24-cell in Hilbert’s Geometry and the Imagination.*

— SCOTT KIM, letter to Martin Gardner (December 1975)

**261.** One solution is obtained by using the 7-tuples  $(1, 2, 5, 44, 9, 43, 4)$ ,  $(6, 15, 10, 23, 22, 19, 13)$ ,  $(14, 12, 16, 11, 18, 17, 20)$ , to “unreduce” the 1st, 2nd, 3rd coordinates. For example, subcuboid 670617 becomes  $4 \times (6+15+10+23+22) \times (12+16+11+18+17+20)$ . The resulting dissection, into blocks of sizes  $1 \times 87 \times 88$ ,  $2 \times 42 \times 74$ ,  $3 \times 21 \times 26$ ,  $4 \times 76 \times 94$ ,  $5 \times 10 \times 16$ ,  $6 \times 82 \times 104$ ,  $7 \times 33 \times 46$ ,  $8 \times 15 \times 62$ ,  $9 \times 18 \times 22$ ,  $11 \times 23 \times 44$ ,  $12 \times 31 \times 101$ ,  $13 \times 71 \times 107$ ,  $14 \times 95 \times 105$ ,  $17 \times 54 \times 60$ ,  $19 \times 56 \times 57$ ,  $20 \times 61 \times 102$ ,  $25 \times 27 \times 96$ ,  $28 \times 49 \times 64$ ,  $29 \times 41 \times 51$ ,  $32 \times 37 \times 47$ ,  $35 \times 48 \times 53$ ,  $39 \times 45 \times 52$ ,  $43 \times 55 \times 70$ , makes a fiendishly difficult puzzle.

How were those magic 7-tuples discovered? An exhaustive search such as that of exercise 256 was out of the question. The author first looked for 7-tuples that would not lead to any dimensions in the “popular” ranges  $[11..25]$  and  $[30..42]$ ; there are 1130, of which the fourth was  $(1, 2, 5, 44, 9, 43, 4)$ . A subsequent search found 25112 7-tuples that don’t conflict with this one, in the 23 relevant places; and those 7-tuples included 26 that don’t conflict with each other.

(The cube size 108 can probably be decreased, but probably not by much.)

**262.** The exact cover problem of answer 247 is readily extended to 3D: The option for every admissible subcuboid  $[a..b] \times [c..d] \times [e..f]$  has  $6 + (b-a)(d-c)(f-e)$  items, namely  $x_a y_c z_e x_{ab} y_{cd} z_{ef}$  and the cells  $p_{ijk}$  that are covered.

We can do somewhat better, as in exercise 248: Most of the improvement in that answer can be achieved also 3Dwise, if we simply omit cases where  $a = l - 1$  and either  $c + d > m$  or  $e + f > n$ . Furthermore, if  $m = n$  we can omit cases with  $(e, f) < (c, d)$ .

Without those omissions, Algorithm M handles the case  $l = m = n = 7$  in 98 teramems, producing 2432 solutions. With them, the running time is reduced to 43 teramems, and 397 solutions are found.

(The  $7 \times 7 \times 7$  problem can be factored into subproblems, based on the patterns that appear on the cube’s six visible faces. These patterns reduce to  $5 \times 5$  pinwheels, and it takes only about  $40 M\mu$  to discover all 152 possibilities. Furthermore, those possibilities reduce to only 5 cases, under the 48 symmetries of a cube. Each of those cases can then be solved by embedding the  $5 \times 5$  reduced patterns into  $7 \times 7$  unreduced patterns, considering  $15^3 = 3375$  possibilities for the three faces adjacent to vertex 000. Most of those possibilities are immediately ruled out. Hence each of the five cases can be solved by Algorithm C in about  $70 G\mu$ —making the total running time about  $350 G\mu$ . However, this 120-fold increase in speed cost the author two man-days of work.)

All three methods showed that, up to isomorphism, exactly 56 distinct motley cubes of size  $7 \times 7 \times 7$  are possible. Each of those 56 dissections has exactly 23 cuboids. Nine of them are symmetric under the mapping  $xyz \mapsto (7-x)(7-y)(7-z)$ ; and one of those nine, namely the one in exercise 260, has six automorphisms.

[These runs confirm and slightly extend the work of W. H. Cutler in *JRM* **12** (1979), 104–111. His computer program found exactly 56 distinct possibilities, when restricting the search to solutions that have exactly 23 cuboids.]

Cutler  
author

**263.** The author has confirmed this conjecture only for  $l, m, n \leq 8$  and  $l + m + n \leq 22$ .

**999.** ...

## INDEX AND GLOSSARY

Pope  
Homer  
WHEATLEY

*There is a curious poetical index to the Iliad in Pope's Homer, referring to all the places in which similes are used.*

— HENRY B. WHEATLEY, *What is an Index?* (1878)

When an index entry refers to a page containing a relevant exercise, see also the *answer* to that exercise for further information. An answer page is not indexed here unless it refers to a topic not included in the statement of the exercise.

Barris, Harry, 1.

*DIMACS: DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, inaugurated in 1990.

Fields, Dorothy, 1.

MPR: Mathematical Preliminaries Redux, v.

Short, Robert Allen, iii.

Nothing else is indexed yet (sorry).

Preliminary notes for indexing appear in the upper right corner of most pages.

If I've mentioned somebody's name and forgotten to make such an index note, it's an error (worth \$2.56).

Note to readers:  
Please ignore these  
sidenotes; they're just  
hints to myself for  
preparing the index,  
and they're often flaky!

KNUTH

# THE ART OF COMPUTER PROGRAMMING

VOLUME 4    PRE-FASCICLE 6A

## A DRAFT OF SECTION 7.2.2.2: SATISFIABILITY

DONALD E. KNUTH *Stanford University*

ADDISON-WESLEY



September 23, 2015

Internet  
Stanford GraphBase  
MMIX

Internet page <http://www-cs-faculty.stanford.edu/~knuth/taocp.html> contains current information about this book and related books.

See also <http://www-cs-faculty.stanford.edu/~knuth/sgb.html> for information about *The Stanford GraphBase*, including downloadable software for dealing with the graphs used in many of the examples in Chapter 7.

See also <http://www-cs-faculty.stanford.edu/~knuth/mmixware.html> for downloadable software to simulate the MMIX computer.

Copyright © 2015 by Addison–Wesley

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher, except that the official electronic file may be used to print single copies for personal (not commercial) use.

Zeroth printing (revision 5), 15 September 2015

September 23, 2015

## PREFACE

*These unforeseen stoppages,  
which I own I had no conception of when I first set out;  
— but which, I am convinced now, will rather increase than diminish as I advance,  
— have struck out a hint which I am resolved to follow;  
— and that is, — not to be in a hurry;  
— but to go on leisurely, writing and publishing two volumes of my life every year;  
— which, if I am suffered to go on quietly, and can make a tolerable bargain  
with my bookseller, I shall continue to do as long as I live.*  
— LAURENCE STERNE, *The Life and Opinions of  
Tristram Shandy, Gentleman* (1759)

THIS BOOKLET contains draft material that I'm circulating to experts in the field, in hopes that they can help remove its most egregious errors before too many other people see it. I am also, however, posting it on the Internet for courageous and/or random readers who don't mind the risk of reading a few pages that have not yet reached a very mature state. *Beware:* This material has not yet been proofread as thoroughly as the manuscripts of Volumes 1, 2, 3, and 4A were at the time of their first printings. And those carefully-checked volumes, alas, were subsequently found to contain thousands of mistakes.

Given this caveat, I hope that my errors this time will not be so numerous and/or obtrusive that you will be discouraged from reading the material carefully. I did try to make the text both interesting and authoritative, as far as it goes. But the field is vast; I cannot hope to have surrounded it enough to corral it completely. So I beg you to let me know about any deficiencies that you discover.

To put the material in context, this long pre-fascicle contains Section 7.2.2.2 of a long, long chapter on combinatorial algorithms. Chapter 7 will eventually fill at least four volumes (namely Volumes 4A, 4B, 4C, and 4D), assuming that I'm able to remain healthy. It began in Volume 4A with a short review of graph theory and a longer discussion of "Zeros and Ones" (Section 7.1); that volume concluded with Section 7.2.1, "Generating Basic Combinatorial Patterns," which was the first part of Section 7.2, "Generating All Possibilities." Volume 4B will begin with Section 7.2.2, about backtracking in general, and Section 7.2.2.1 will discuss a family of methods called "dancing links," for updating data structures while backtracking. That sets the scene for the present section, which applies those ideas to the important problem of Boolean satisfiability.

\* \* \*



\* \* \*

After working on this material for more than three years, I've finally gotten most of the major topics into place. Each time I finish drafting a small piece of the final big picture, I've been testing it by adding it to these pages; hence there now are quite a few scraps of text and exercises, which I plan to refine and polish (if the FORCE stays with me).

I hope you will see why I've found this topic to be such a fascinating story; and I hope you'll not get too lost as I move through different parts of the tale. I've tried to explain things in a natural order.

\* \* \*

My notes on combinatorial algorithms have been accumulating for more than fifty years, yet I fear that in many respects my knowledge is woefully behind the times. Please look, for example, at the exercises that I've classed as research problems (rated with difficulty level 46 or higher), namely exercises 5, 39, 112, 193, 194, 236, 283, 516, . . . ; exercise 223 is also currently unsolved, although I've rated it only '40' because I once thought of an answer (which I have since forgotten!). I've also implicitly mentioned or posed additional unsolved questions in the answers to exercises 18, 19, 68, 84, 105(c,e), 111, 132, 183, 194, 204, 205, 316, 335, 351, 360, 365, 372, 397, 409(c), 476, 480, 486, 487, 488, 501, 511, 515, . . . . And I still haven't solved exercise 68. Are those problems still open? Please inform me if you know of a solution to any of those intriguing enigmas. And of course if no solution is known today but you do make progress on any of them in the future, I hope you'll let me know.

I urgently need your help also with respect to some exercises that I made up as I was preparing this material. I certainly don't like to receive credit for things that have already been published by others, and most of these results are quite natural "fruits" that were just waiting to be "plucked." Therefore please tell me if you know who deserves to be credited, with respect to the ideas found in exercises 18, 19, 20, 21, 22, 24, 29, 38(b), 62, 63, 65(b), 74, 84(c,d,e), 101, 108, 132, 133, 149, 151, 161, 162, 177, 180, 181, 188, 191, 204(b,c,d), 206, 207, 208, 220, 228, 229, 232, 239, 242, 252, 259, 270, 272, 273, 279, 280, 282, 300, 305, 310, 311, 312, 327, 328, 329, 334, 335, 336(b), 337, 343, 349, 357, 358, 361, 390, 396, 399(c), 404, 406, 410, 411, 414, 419, 423, 427, 432, 433, 435, 439, 462, 463, 464, 465, 470, 472, 473, 475(d,e,f,g), 476, 479, 495, 498, . . . , and/or the answers to exercises . . . . Furthermore I've credited exercise 170 to unpublished work of Heule. Have any of those results ever appeared in print, to your knowledge?

I also wonder if Eq. 7.2.2.2–(169) is "well known."

\* \* \*

Special thanks are due to Armin Biere, Randy Bryant, Sam Buss, Niklas Eén, Ian Gent, Marijn Heule, Holger Hoos, Svante Janson, Peter Jeavons, Daniel Kroening, Oliver Kullmann, Massimo Lauria, Wes Pegden, Will Shortz, Carsten Sinz, Niklas Sörensson, Udo Wermuth, Ryan Williams, and . . . for their detailed comments on my early attempts at exposition, as well as to numerous other correspondents who have contributed crucial corrections. Thanks also to Stanford’s Information Systems Laboratory for providing extra computer power when my laptop machine was inadequate.

Biere  
 Bryant  
 Buss  
 Eén  
 Gent  
 Heule  
 Hoos  
 Janson  
 Jeavons  
 Kroening  
 Kullmann  
 Lauria  
 Pegden  
 Shortz  
 Sinz  
 Sörensson  
 Wermuth  
 Williams  
 Internet  
 MPR  
 Internet

\* \* \*

Wow — Section 7.2.2.2 has turned out to be the longest section, by far, in *The Art of Computer Programming*. The SAT problem is evidently a “killer app,” because it is key to the solution of so many other problems. Consequently I can only hope that my lengthy treatment does not also kill off my faithful readers! As I wrote this material, one topic always seemed to flow naturally into another, so there was no neat way to break this section up into separate subsections. (And anyway the format of *TAOCP* doesn’t allow for a Section 7.2.2.2.1.)

I’ve tried to ameliorate the reader’s navigation problem by adding subheadings at the top of each right-hand page. Furthermore, as in other sections, the exercises appear in an order that roughly parallels the order in which corresponding topics are taken up in the text. Numerous cross-references are provided between text, exercises, and illustrations, so that you have a fairly good chance of keeping in sync. I’ve also tried to make the index as comprehensive as possible.

I wrote more than three hundred computer programs while preparing this material, because I find that I don’t understand things unless I try to program them. Most of those programs were quite short, of course; but several of them are rather substantial, and possibly of interest to others. Therefore I’ve made a selection available by listing some of them on the following webpage:

<http://www-cs-faculty.stanford.edu/~knuth/programs.html>

I happily offer a “finder’s fee” of \$2.56 for each error in this draft when it is first reported to me, whether that error be typographical, technical, or historical. The same reward holds for items that I forgot to put in the index. And valuable suggestions for improvements to the text are worth 32¢ each. (Furthermore, if you find a better solution to an exercise, I’ll actually do my best to give you immortal glory, by publishing your name in the eventual book:—)

Volume 4B will begin with a special tutorial and review of probability theory, in an unnumbered section entitled “Mathematical Preliminaries Redux.” References to its equations and exercises use the abbreviation ‘MPR’. (Think of the word “improvement.”) A preliminary version of that section can be found online, as pre-fascicle 5a to Volume 4, if you knew how to find this one.

Cross references to yet-unwritten material sometimes appear as ‘00’; this impossible value is a placeholder for the actual numbers to be supplied later.

The notational conventions that I've used in the mathematical formulas of this section are summarized either in the Index to Notations of Volume 4A (Appendix B on pages 822–827) or under the heading 'Notational conventions' in the index below.

notational conventions  
Knuth

Happy reading!

*Stanford, California*  
*69 Umbruary 2015*

D. E. K.

Here's an exercise for Section 7.2.2.1 that I plan to put eventually into fascicle 5:

**00.** [20] The problem of Langford pairs on  $\{1, 1, \dots, n, n\}$  can be represented as an exact cover problem using columns  $\{d_1, \dots, d_n\} \cup \{s_1, \dots, s_{2n}\}$ ; the rows are  $d_i s_j s_k$  for  $1 \leq i \leq n$  and  $1 \leq j < k \leq 2n$  and  $k = i + j + 1$ , meaning "put digit  $i$  into slots  $j$  and  $k$ ."

However, that construction essentially gives us every solution twice, because the left-right reversal of any solution is also a solution. Modify it so that we get only half as many solutions; the others will be the reversals of these.

And here's its cryptic answer (needed in exercise 7.2.2.2–13):

**00.** Omit the rows with  $i = n - [n \text{ even}]$  and  $j > n/2$ .

(Other solutions are possible. For example, we could omit the rows with  $i = 1$  and  $j \geq n$ ; that would omit  $n - 1$  rows instead of only  $[n/2]$ . However, the suggested rule turns out to make the dancing links algorithm run about 10% faster.)

Langford pairs  
exact cover problem  
symmetry breaking  
DEFOE  
Crusoe

*Now I saw, tho' too late, the Folly of  
beginning a Work before we count the Cost,  
and before we judge rightly of our own Strength to go through with it.*

— DANIEL DEFOE, *Robinson Crusoe* (1719)

## CONTENTS

|  |     |
|--|-----|
| <b>Chapter 7 — Combinatorial Searching</b> . . . . .     | 0   |
| 7.2. Generating All Possibilities . . . . .              | 0   |
| 7.2.1. Generating Basic Combinatorial Patterns . . . . . | 0   |
| 7.2.2. Basic Backtrack . . . . .                         | 0   |
| 7.2.2.1. Dancing links . . . . .                         | 0   |
| 7.2.2.2. Satisfiability . . . . .                        | 1   |
| Example applications . . . . .                           | 4   |
| Backtracking algorithms . . . . .                        | 27  |
| Random clauses . . . . .                                 | 47  |
| Resolution of clauses . . . . .                          | 54  |
| Clause-learning algorithms . . . . .                     | 60  |
| Monte Carlo algorithms . . . . .                         | 77  |
| The Local Lemma . . . . .                                | 81  |
| *Message-passing algorithms . . . . .                    | 90  |
| *Preprocessing of clauses . . . . .                      | 95  |
| Encoding constraints into clauses . . . . .              | 97  |
| Unit propagation and forcing . . . . .                   | 103 |
| Symmetry breaking . . . . .                              | 105 |
| Satisfiability-preserving maps . . . . .                 | 107 |
| One hundred test cases . . . . .                         | 113 |
| Tuning the parameters . . . . .                          | 124 |
| Exploiting parallelism . . . . .                         | 128 |
| History . . . . .  | 129 |
| Exercises . . . . .                                      | 133 |
| <b>Answers to Exercises</b> . . . . .                    | 185 |
| <b>Index to Algorithms and Theorems</b> . . . . .        | 292 |
| <b>Index and Glossary</b> . . . . .                      | 293 |

*That your book has been delayed I am glad,  
since you have gained an opportunity of being more exact.*  
— SAMUEL JOHNSON, letter to Charles Burney (1 November 1784)

HUME  
 JAGGER  
 RICHARDS  
 Boolean formula  
 unsatisfiable  
 satisfying assignment  
 consistent, see satisfiable  
 inconsistent, see unsatisfiable  
 $P = NP$   
 NP-complete problem  
 Knuth  
 SAT solvers

*He reaps no satisfaction but from low and sensual objects,  
 or from the indulgence of malignant passions.*

— DAVID HUME, *The Sceptic* (1742)

*I can't get no ...*

— MICK JAGGER and KEITH RICHARDS, *Satisfaction* (1965)

**7.2.2.2. Satisfiability.** We turn now to one of the most fundamental problems of computer science: Given a Boolean formula  $F(x_1, \dots, x_n)$ , expressed in so-called “conjunctive normal form” as an AND of ORs, can we “satisfy”  $F$  by assigning values to its variables in such a way that  $F(x_1, \dots, x_n) = 1$ ? For example, the formula

$$F(x_1, x_2, x_3) = (x_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \quad (1)$$

is satisfied when  $x_1 x_2 x_3 = 001$ . But if we rule that solution out, by defining

$$G(x_1, x_2, x_3) = F(x_1, x_2, x_3) \wedge (x_1 \vee x_2 \vee \bar{x}_3), \quad (2)$$

then  $G$  is unsatisfiable: It has no satisfying assignment.

Section 7.1.1 discussed the embarrassing fact that nobody has ever been able to come up with an efficient algorithm to solve the general satisfiability problem, in the sense that the satisfiability of any given formula of size  $N$  could be decided in  $N^{O(1)}$  steps. Indeed, the famous unsolved question “does  $P = NP$ ?” is equivalent to asking whether such an algorithm exists. We will see in Section 7.9 that satisfiability is a natural progenitor of every NP-complete problem.\*

On the other hand enormous technical breakthroughs in recent years have led to amazingly good ways to approach the satisfiability problem. We now have algorithms that are much more efficient than anyone had dared to believe possible before the year 2000. These so-called “SAT solvers” are able to handle industrial-strength problems, involving millions of variables, with relative ease, and they’ve had a profound impact on many areas of research such as computer-aided verification. In this section we shall study the principles that underlie modern SAT-solving procedures.

---

\* At the present time very few people believe that  $P = NP$  [see *SIGACT News* **43**, 2 (June 2012), 53–77]. In other words, almost everybody who has studied the subject thinks that satisfiability cannot be decided in polynomial time. The author of this book, however, suspects that  $N^{O(1)}$ -step algorithms do exist, yet that they’re unknowable. Almost all polynomial time algorithms are so complicated that they lie beyond human comprehension, and could never be programmed for an actual computer in the real world. Existence is different from embodiment.

To begin, let's define the problem carefully and simplify the notation, so that our discussion will be as efficient as the algorithms that we'll be considering. Throughout this section we shall deal with *variables*, which are elements of any convenient set. Variables are often denoted by  $x_1, x_2, x_3, \dots$ , as in (1); but any other symbols can also be used, like  $a, b, c$ , or even  $d''_{74}$ . We will in fact often use the numerals 1, 2, 3,  $\dots$  to stand for variables; and in many cases we'll find it convenient to write just  $j$  instead of  $x_j$ , because it takes less time and less space if we don't have to write so many  $x$ 's. Thus '2' and ' $x_2$ ' will mean the same thing in many of the discussions below.

A *literal* is either a variable or the complement of a variable. In other words, if  $v$  is a variable, both  $v$  and  $\bar{v}$  are literals. If there are  $n$  possible variables in some problem, there are  $2n$  possible literals. If  $l$  is the literal  $\bar{x}_2$ , which is also written  $\bar{2}$ , then the complement of  $l$ ,  $\bar{l}$ , is  $x_2$ , which is also written 2.

The variable that corresponds to a literal  $l$  is denoted by  $|l|$ ; thus we have  $|v| = |\bar{v}| = v$  for every variable  $v$ . Sometimes we write  $\pm v$  for a literal that is either  $v$  or  $\bar{v}$ . We might also denote such a literal by  $\sigma v$ , where  $\sigma$  is  $\pm 1$ . The literal  $l$  is called *positive* if  $|l| = l$ ; otherwise  $|l| = \bar{l}$ , and  $l$  is said to be *negative*.

Two literals  $l$  and  $l'$  are *distinct* if  $l \neq l'$ . They are *strictly distinct* if  $|l| \neq |l'|$ . A set of literals  $\{l_1, \dots, l_k\}$  is strictly distinct if  $|l_i| \neq |l_j|$  for  $1 \leq i < j \leq k$ .

The satisfiability problem, like all good problems, can be understood in many equivalent ways, and we will find it convenient to switch from one viewpoint to another as we deal with different aspects of the problem. Example (1) is an AND of clauses, where every clause is an OR of literals; but we might as well regard every clause as simply a *set* of literals, and a formula as a set of clauses. With that simplification, and with ' $x_j$ ' identical to ' $j$ ', Eq. (1) becomes

$$F = \{\{1, \bar{2}\}, \{2, 3\}, \{\bar{1}, \bar{3}\}, \{\bar{1}, \bar{2}, 3\}\}.$$

And we needn't bother to represent the clauses with braces and commas either; we can simply write out the literals of each clause. With that shorthand we're able to perceive the real essence of (1) and (2):

$$F = \{1\bar{2}, 23, \bar{1}\bar{3}, \bar{1}\bar{2}3\}, \quad G = F \cup \{12\bar{3}\}. \quad (3)$$

Here  $F$  is a set of four clauses, and  $G$  is a set of five.

In this guise, the satisfiability problem is equivalent to a *covering problem*, analogous to the exact cover problems that we considered in Section 7.2.2.1: Let

$$T_n = \{\{x_1, \bar{x}_1\}, \{x_2, \bar{x}_2\}, \dots, \{x_n, \bar{x}_n\}\} = \{1\bar{1}, 2\bar{2}, \dots, n\bar{n}\}. \quad (4)$$

"Given a set  $F = \{C_1, \dots, C_m\}$ , where each  $C_i$  is a clause and each clause consists of literals based on the variables  $\{x_1, \dots, x_n\}$ , find a set  $L$  of  $n$  literals that 'covers'  $F \cup T_n$ , in the sense that every clause contains at least one element of  $L$ ." For example, the set  $F$  in (3) is covered by  $L = \{\bar{1}, \bar{2}, 3\}$ , and so is the set  $T_3$ ; hence  $F$  is satisfiable. The set  $G$  is covered by  $\{1, \bar{1}, 2\}$  or  $\{1, \bar{1}, 3\}$  or  $\dots$  or  $\{\bar{2}, 3, \bar{3}\}$ , but not by any three literals that also cover  $T_3$ ; so  $G$  is unsatisfiable.

Similarly, a family  $F$  of clauses is satisfiable if and only if it can be covered by a set  $L$  of *strictly distinct* literals.

variables  
literal  
notation:  $|v|$   
positive  
negative  
distinct  
strictly distinct  
covering problem  
exact cover problems

If  $F'$  is any formula obtained from  $F$  by complementing one or more variables, it's clear that  $F'$  is satisfiable if and only if  $F$  is satisfiable. For example, if we replace 1 by  $\bar{1}$  and 2 by  $\bar{2}$  in (3) we obtain

$$F' = \{\bar{1}2, \bar{2}3, 1\bar{3}, 123\}, \quad G = F \cup \{\bar{1}\bar{2}\bar{3}\}.$$

In this case  $F'$  is *trivially* satisfiable, because each of its clauses contains a positive literal: Every such formula is satisfied by simply letting  $L$  be the set of positive literals. Thus the satisfiability problem is the same as the problem of switching signs (or “polarities”) so that no all-negative clauses remain.

Another problem equivalent to satisfiability is obtained by going back to the Boolean interpretation in (1) and complementing both sides of the equation. By De Morgan's laws 7.1.1–(11) and (12) we have

$$\overline{F}(x_1, x_2, x_3) = (\bar{x}_1 \wedge x_2) \vee (\bar{x}_2 \wedge \bar{x}_3) \vee (x_1 \wedge x_3) \vee (x_1 \wedge x_2 \wedge \bar{x}_3); \quad (5)$$

and  $F$  is unsatisfiable  $\iff F = 0 \iff \overline{F} = 1 \iff \overline{F}$  is a tautology. Consequently  $F$  is satisfiable if and only if  $\overline{F}$  is not a tautology: The tautology problem and the satisfiability problem are essentially the same.\*

Since the satisfiability problem is so important, we simply call it SAT. And instances of the problem such as (1), in which there are no clauses of length greater than 3, are called 3SAT. In general,  $k$ SAT is the satisfiability problem restricted to instances where no clause has more than  $k$  literals.

Clauses of length 1 are called *unit clauses*, or unary clauses. Binary clauses, similarly, have length 2; then come ternary clauses, quaternary clauses, and so forth. Going the other way, the *empty clause*, or nullary clause, has length 0 and is denoted by  $\epsilon$ ; it is always unsatisfiable. Short clauses are very important in algorithms for SAT, because they are easier to deal with than long clauses. But long clauses aren't necessarily bad; they're much easier to satisfy than the short ones.

A slight technicality arises when we consider clause length: The binary clause  $(x_1 \vee \bar{x}_2)$  in (1) is equivalent to the ternary clause  $(x_1 \vee x_1 \vee \bar{x}_2)$  as well as to  $(x_1 \vee \bar{x}_2 \vee \bar{x}_2)$  and to longer clauses such as  $(x_1 \vee x_1 \vee x_1 \vee \bar{x}_2)$ ; so we can regard it as a clause of *any* length  $\geq 2$ . But when we think of clauses as *sets* of literals rather than ORs of literals, we usually rule out multisets such as  $11\bar{2}$  or  $1\bar{2}\bar{2}$  that aren't sets; in that sense a binary clause is *not* a special case of a ternary clause. On the other hand, every binary clause  $(x \vee y)$  is equivalent to *two* ternary clauses,  $(x \vee y \vee z) \wedge (x \vee y \vee \bar{z})$ , if  $z$  is another variable; and every  $k$ -ary clause is equivalent to two  $(k+1)$ -ary clauses. Therefore we can assume, if we like, that  $k$ SAT deals only with clauses whose length is exactly  $k$ .

A clause is tautological (always satisfied) if it contains both  $v$  and  $\bar{v}$  for some variable  $v$ . Tautological clauses can be denoted by  $\wp$  (see exercise 7.1.4–222). They never affect a satisfiability problem; so we usually assume that the clauses input to a SAT-solving algorithm consist of strictly distinct literals.

When we discussed the 3SAT problem briefly in Section 7.1.1, we took a look at formula 7.1.1–(32), “the shortest interesting formula in 3CNF.” In our new

trivially SAT  
polarities  
De Morgan's laws  
TAUT  
coNP-complete  
SAT  
kSAT  
3SAT  
unit clauses  
unary clauses  
Binary clauses  
ternary clauses  
empty clause  
nullary clause  
 $\epsilon$   
multisets  
tautological  
 $\wp$   
strictly distinct literals

\* Strictly speaking, TAUT is coNP-complete, while SAT is NP-complete; see Section 7.9.



shorthand, it consists of the following eight unsatisfiable clauses:

$$R = \{12\bar{3}, 23\bar{4}, 341, 4\bar{1}2, \bar{1}\bar{2}3, \bar{2}\bar{3}4, \bar{3}\bar{4}\bar{1}, \bar{4}\bar{1}\bar{2}\}. \quad (6)$$

This set makes an excellent little test case, so we will refer to it frequently below. (The letter  $R$  reminds us that it is based on R. L. Rivest's associative block design 6.5-(13).) The first seven clauses of  $R$ , namely

$$R' = \{12\bar{3}, 23\bar{4}, 341, 4\bar{1}2, \bar{1}\bar{2}3, \bar{2}\bar{3}4, \bar{3}\bar{4}\bar{1}\}, \quad (7)$$

also make nice test data; they are satisfied only by choosing the complements of the literals in the omitted clause, namely  $\{4, \bar{1}, 2\}$ . More precisely, the literals 4,  $\bar{1}$ , and 2 are necessary and sufficient to cover  $R'$ ; we can also include either 3 or  $\bar{3}$  in the solution. Notice that (6) is symmetric under the cyclic permutation  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \bar{1} \rightarrow \bar{2} \rightarrow \bar{3} \rightarrow \bar{4} \rightarrow 1$  of literals; thus, omitting *any* clause of (6) gives a satisfiability problem equivalent to (7).

**A simple example.** SAT solvers are important because an enormous variety of problems can readily be formulated Booleanwise as ANDs of ORs. Let's begin with a little puzzle that leads to an instructive family of example problems: *Find a binary sequence  $x_1 \dots x_8$  that has no three equally spaced 0s and no three equally spaced 1s.* For example, the sequence 01001011 almost works; but it doesn't qualify, because  $x_2$ ,  $x_5$ , and  $x_8$  are equally spaced 1s.

If we try to solve this puzzle by backtracking manually through all 8-bit sequences in lexicographic order, we see that  $x_1x_2 = 00$  forces  $x_3 = 1$ . Then  $x_1x_2x_3x_4x_5x_6x_7 = 0010011$  leaves us with no choice for  $x_8$ . A minute or two of further hand calculation reveals that the puzzle has just six solutions, namely

$$00110011, 01011010, 01100110, 10011001, 10100101, 11001100. \quad (8)$$

Furthermore it's easy to see that none of these solutions can be extended to a suitable binary sequence of length 9. We conclude that *every binary sequence  $x_1 \dots x_9$  contains three equally spaced 0s or three equally spaced 1s.*

Notice now that the condition  $x_2x_5x_8 \neq 111$  is the same as the Boolean clause  $(\bar{x}_2 \vee \bar{x}_5 \vee \bar{x}_8)$ , namely  $\bar{2}\bar{5}\bar{8}$ . Similarly  $x_2x_5x_8 \neq 000$  is the same as  $258$ . So we have just verified that the following 32 clauses are unsatisfiable:

$$\begin{aligned} &123, 234, \dots, 789, 135, 246, \dots, 579, 147, 258, 369, 159, \\ &\bar{1}\bar{2}\bar{3}, \bar{2}\bar{3}\bar{4}, \dots, \bar{7}\bar{8}\bar{9}, \bar{1}\bar{3}\bar{5}, \bar{2}\bar{4}\bar{6}, \dots, \bar{5}\bar{7}\bar{9}, \bar{1}\bar{4}\bar{7}, \bar{2}\bar{5}\bar{8}, \bar{3}\bar{6}\bar{9}, \bar{1}\bar{5}\bar{9}. \end{aligned} \quad (9)$$

This result is a special case of a general fact that holds for any given positive integers  $j$  and  $k$ : *If  $n$  is sufficiently large, every binary sequence  $x_1 \dots x_n$  contains either  $j$  equally spaced 0s or  $k$  equally spaced 1s.* The smallest such  $n$  is denoted by  $W(j, k)$  in honor of B. L. van der Waerden, who proved an even more general result (see exercise 2.3.4.3-6): *If  $n$  is sufficiently large, and if  $k_0, \dots, k_{b-1}$  are positive integers, every  $b$ -ary sequence  $x_1 \dots x_n$  contains  $k_a$  equally spaced  $a$ 's for some digit  $a$ ,  $0 \leq a < b$ .* The least such  $n$  is  $W(k_0, \dots, k_{b-1})$ .

Let us accordingly define the following set of clauses when  $j, k, n > 0$ :

$$\begin{aligned} \text{waerden}(j, k; n) = &\{(x_i \vee x_{i+d} \vee \dots \vee x_{i+(j-1)d}) \mid 1 \leq i \leq n - (j-1)d, d \geq 1\} \\ &\cup \{(\bar{x}_i \vee \bar{x}_{i+d} \vee \dots \vee \bar{x}_{i+(k-1)d}) \mid 1 \leq i \leq n - (k-1)d, d \geq 1\}. \end{aligned} \quad (10)$$

The 32 clauses in (9) are  $waerden(3, 3; 9)$ ; and in general  $waerden(j, k; n)$  is an appealing instance of SAT, satisfiable if and only if  $n < W(j, k)$ .

It's obvious that  $W(1, k) = k$  and  $W(2, k) = 2k - [k \text{ even}]$ ; but when  $j$  and  $k$  exceed 2 the numbers  $W(j, k)$  are quite mysterious. We've seen that  $W(3, 3) = 9$ , and the following nontrivial values are currently known:

|             |    |    |     |      |     |     |     |    |     |     |     |     |     |     |     |     |     |
|-------------|----|----|-----|------|-----|-----|-----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $k =$       | 3  | 4  | 5   | 6    | 7   | 8   | 9   | 10 | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  |
| $W(3, k) =$ | 9  | 18 | 22  | 32   | 46  | 58  | 77  | 97 | 114 | 135 | 160 | 186 | 218 | 238 | 279 | 312 | 349 |
| $W(4, k) =$ | 18 | 35 | 55  | 73   | 109 | 146 | 309 | ?  | ?   | ?   | ?   | ?   | ?   | ?   | ?   | ?   | ?   |
| $W(5, k) =$ | 22 | 55 | 178 | 206  | 260 | ?   | ?   | ?  | ?   | ?   | ?   | ?   | ?   | ?   | ?   | ?   | ?   |
| $W(6, k) =$ | 32 | 73 | 206 | 1132 | ?   | ?   | ?   | ?  | ?   | ?   | ?   | ?   | ?   | ?   | ?   | ?   | ?   |

Chvátal  
Kouril  
Paul  
Kouril  
Ahmed  
Kullmann  
Snevily  
monotonic clauses  
exact cover problems  
dancing links  
Langford pairs  
symmetry

V. Chvátal inaugurated the study of  $W(j, k)$  by computing the values for  $j+k \leq 9$  as well as  $W(3, 7)$  [*Combinatorial Structures and Their Applications* (1970), 31–33]. Most of the large values in this table have been calculated by state-of-the-art SAT solvers [see M. Kouril and J. L. Paul, *Experimental Math.* **17** (2008), 53–61; M. Kouril, *Integers* **12** (2012), A46:1–A46:13]. The table entries for  $j = 3$  suggest that we might have  $W(3, k) < k^2$  when  $k > 4$ , but that isn't true: SAT solvers have also been used to establish the lower bounds

$$k = 20 \quad 21 \quad 22 \quad 23 \quad 24 \quad 25 \quad 26 \quad 27 \quad 28 \quad 29 \quad 30$$

$$W(3, k) \geq 389 \quad 416 \quad 464 \quad 516 \quad 593 \quad 656 \quad 727 \quad 770 \quad 827 \quad 868 \quad 903$$

(which might in fact be the true values for this range of  $k$ ); see T. Ahmed, O. Kullmann, and H. Snevily [*Discrete Applied Math.* **174** (2014), 27–51].

Notice that the literals in every clause of  $waerden(j, k; n)$  have the same sign: They're either all positive or all negative. Does this “monotonic” property make the SAT problem any easier? Unfortunately, no: Exercise 10 proves that *any* set of clauses can be converted to an equivalent set of monotonic clauses.

**Exact covering.** The exact cover problems that we solved with “dancing links” in Section 7.2.2.1 can easily be reformulated as instances of SAT and handed off to SAT solvers. For example, let's look again at Langford pairs, the task of placing two 1s, two 2s, . . . , two  $n$ 's into  $2n$  slots so that exactly  $k$  slots intervene between the two appearances of  $k$ , for each  $k$ . The corresponding exact cover problem when  $n = 3$  has nine columns and eight rows (see 7.2.2.1–(oo)):

$$d_1 s_1 s_3, \quad d_1 s_2 s_4, \quad d_1 s_3 s_5, \quad d_1 s_4 s_6, \quad d_2 s_1 s_4, \quad d_2 s_2 s_5, \quad d_2 s_3 s_6, \quad d_3 s_1 s_5. \quad (11)$$

The columns are  $d_i$  for  $1 \leq i \leq 3$  and  $s_j$  for  $1 \leq j \leq 6$ ; the row ' $d_i s_j s_k$ ' means that digit  $i$  is placed in slots  $j$  and  $k$ . Left-right symmetry allows us to omit the row ' $d_3 s_2 s_6$ ' from this specification.

We want to select rows of (11) so that each column appears just once. Let the Boolean variable  $x_j$  mean ‘select row  $j$ ’, for  $1 \leq j \leq 8$ ; the problem is then to satisfy the nine constraints

$$S_1(x_1, x_2, x_3, x_4) \wedge S_1(x_5, x_6, x_7) \wedge S_1(x_8)$$

$$\wedge S_1(x_1, x_5, x_8) \wedge S_1(x_2, x_6) \wedge S_1(x_1, x_3, x_7)$$

$$\wedge S_1(x_2, x_4, x_5) \wedge S_1(x_3, x_6, x_8) \wedge S_1(x_4, x_7), \quad (12)$$

one for each column. (Here, as usual,  $S_1(y_1, \dots, y_p)$  denotes the symmetric function  $[y_1 + \dots + y_p = 1]$ .) For example, we must have  $x_5 + x_6 + x_7 = 1$ , because column  $d_2$  appears in rows 5, 6, and 7 of (11).

One of the simplest ways to express the symmetric Boolean function  $S_1$  as an AND of ORs is to use  $1 + \binom{p}{2}$  clauses:

$$S_1(y_1, \dots, y_p) = (y_1 \vee \dots \vee y_p) \wedge \bigwedge_{1 \leq j < k \leq p} (\bar{y}_j \vee \bar{y}_k). \quad (13)$$

“At least one of the  $y$ ’s is true, but not two.” Then (12) becomes, in shorthand,

$$\{1234, \bar{1}\bar{2}, \bar{1}\bar{3}, \bar{1}\bar{4}, \bar{2}\bar{3}, \bar{2}\bar{4}, \bar{3}\bar{4}, 567, \bar{5}\bar{6}, \bar{5}\bar{7}, \bar{6}\bar{7}, 8, \\ 158, \bar{1}\bar{5}, \bar{1}\bar{8}, \bar{5}\bar{8}, 26, \bar{2}\bar{6}, 137, \bar{1}\bar{3}, \bar{1}\bar{7}, \bar{3}\bar{7}, \\ 245, \bar{2}\bar{4}, \bar{2}\bar{5}, \bar{4}\bar{5}, 368, \bar{3}\bar{6}, \bar{3}\bar{8}, \bar{6}\bar{8}, 47, \bar{4}\bar{7}\}; \quad (14)$$

we shall call these clauses *langford*(3). (Notice that only 30 of them are actually distinct, because  $\bar{1}\bar{3}$  and  $\bar{2}\bar{4}$  appear twice.) Exercise 13 defines *langford*( $n$ ); we know from exercise 7-1 that *langford*( $n$ ) is satisfiable  $\iff n \bmod 4 = 0$  or  $3$ .

The unary clause 8 in (14) tells us immediately that  $x_8 = 1$ . Then from the binary clauses  $\bar{1}\bar{8}, \bar{5}\bar{8}, \bar{3}\bar{8}, \bar{6}\bar{8}$  we have  $x_1 = x_5 = x_3 = x_6 = 0$ . The ternary clause 137 then implies  $x_7 = 1$ ; finally  $x_4 = 0$  (from  $\bar{4}\bar{7}$ ) and  $x_2 = 1$  (from 1234). Rows 8, 7, and 2 of (11) now give us the desired Langford pairing 3 1 2 1 3 2.

Incidentally, the function  $S_1(y_1, y_2, y_3, y_4, y_5)$  can also be expressed as

$$(y_1 \vee y_2 \vee y_3 \vee y_4 \vee y_5) \wedge (\bar{y}_1 \vee \bar{y}_2) \wedge (\bar{y}_1 \vee \bar{y}_3) \wedge (\bar{y}_1 \vee \bar{t}) \\ \wedge (\bar{y}_2 \vee \bar{y}_3) \wedge (\bar{y}_2 \vee \bar{t}) \wedge (\bar{y}_3 \vee \bar{t}) \wedge (t \vee \bar{y}_4) \wedge (t \vee \bar{y}_5) \wedge (\bar{y}_4 \vee \bar{y}_5),$$

where  $t$  is a new variable. In general, if  $p$  gets big, it’s possible to express  $S_1(y_1, \dots, y_p)$  with only  $3p - 5$  clauses instead of  $\binom{p}{2} + 1$ , by using  $\lfloor (p-3)/2 \rfloor$  new variables as explained in exercise 12. When this alternative encoding is used to represent Langford pairs of order  $n$ , we’ll call the resulting clauses *langford'*( $n$ ).

Do SAT solvers do a better job with the clauses *langford*( $n$ ) or *langford'*( $n$ )? Stay tuned: We’ll find out later.

**Coloring a graph.** The classical problem of coloring a graph with at most  $d$  colors is another rich source of benchmark examples for SAT solvers. If the graph has  $n$  vertices  $V$ , we can introduce  $nd$  variables  $v_j$ , for  $v \in V$  and  $1 \leq j \leq d$ , signifying that  $v$  has color  $j$ ; the resulting clauses are quite simple:

$$(v_1 \vee v_2 \vee \dots \vee v_d) \text{ for } v \in V \text{ (“every vertex has at least one color”)}; \quad (15)$$

$$(\bar{u}_j \vee \bar{v}_j) \text{ for } u - v, 1 \leq j \leq d \text{ (“adjacent vertices have different colors”)}. \quad (16)$$

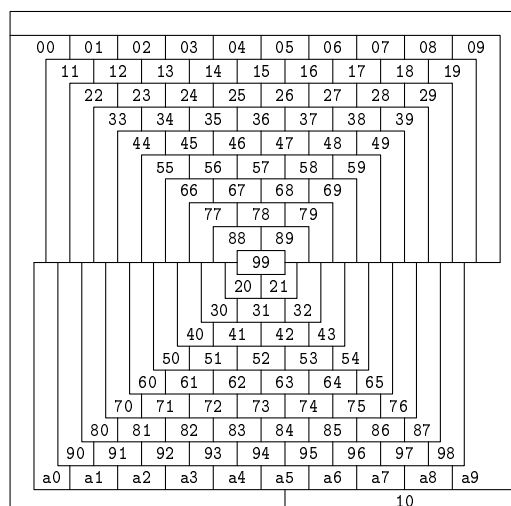
We could also add  $n\binom{d}{2}$  additional so-called *exclusion clauses*

$$(\bar{v}_i \vee \bar{v}_j) \text{ for } v \in V, 1 \leq i < j \leq d \text{ (“every vertex has at most one color”)}; \quad (17)$$

but they’re optional, because vertices with more than one color are harmless. Indeed, if we find a solution with  $v_1 = v_2 = 1$ , we’ll be extra happy, because it gives us *two* legal ways to color vertex  $v$ . (See exercise 14.)

$S_1(y_1, \dots, y_p)$   
symmetric Boolean function  
*langford*( $n$ )  
unary clause  
binary clauses  
ternary clause  
encoding  
Langford pairs  
*langford'*( $n$ )  
coloring a graph  
exclusion clauses  
at-most-one

**Fig. 33.** The McGregor graph of order 10. Each region of this “map” is identified by a two-digit hexadecimal code. Can you color the regions with four colors, never using the same color for two adjacent regions?



Gardner  
chess  
da Vinci  
Ripoff  
April Fool  
Four Color Theorem  
McGregor  
Bryant  
independent

Martin Gardner astonished the world in 1975 when he reported [*Scientific American* **232**, 4 (April 1975), 126–130] that a proper coloring of the planar map in Fig. 33 requires *five* distinct colors, thereby disproving the longstanding four-color conjecture. (In that same column he also cited several other “facts” supposedly discovered in 1974: (i)  $e^{\pi\sqrt{163}}$  is an integer; (ii) pawn-to-king-rook-4 (‘h4’) is a winning first move in chess; (iii) the theory of special relativity is fatally flawed; (iv) Leonardo da Vinci invented the flush toilet; and (v) Robert Ripoff devised a motor that is powered entirely by psychic energy. Thousands of readers failed to notice that they had been April Fooled!)

The map in Fig. 33 actually *can* be 4-colored; you are hereby challenged to discover a suitable way to do this, before turning to the answer of exercise 18. Indeed, the four-color conjecture became the Four Color Theorem in 1976, as mentioned in Section 7. Fortunately that result was still unknown in April of 1975; otherwise this interesting graph would probably never have appeared in print. McGregor’s graph has 110 vertices (regions) and 324 edges (adjacencies between regions); hence (15) and (16) yield  $110 + 1296 = 1406$  clauses on 440 variables, which a modern SAT solver can polish off quickly.

We can also go much further and solve problems that would be extremely difficult by hand. For example, we can add constraints to limit the number of regions that receive a particular color. Randal Bryant exploited this idea in 2010 to discover that there’s a four-coloring of Fig. 33 that uses one of the colors only 7 times (see exercise 17). His coloring is, in fact, unique, and it leads to an explicit way to 4-color the McGregor graphs of all orders  $n \geq 3$  (exercise 18).

Such additional constraints can be generated in many ways. We could, for instance, append  $\binom{110}{8}$  clauses, one for every choice of 8 regions, specifying that those 8 regions aren’t all colored 1. But no, we’d better scratch that idea:  $\binom{110}{8} = 409,705,619,895$ . Even if we restricted ourselves to the 74,792,876,790 sets of 8 regions that are *independent*, we’d be dealing with far too many clauses.

An interesting SAT-oriented way to ensure that  $x_1 + \dots + x_n$  is at most  $r$ , which works well when  $n$  and  $r$  are rather large, was found by C. Sinz [LNCS **3709** (2005), 827–831]. His method introduces  $(n - r)r$  new variables  $s_j^k$  for  $1 \leq j \leq n - r$  and  $1 \leq k \leq r$ . If  $F$  is any satisfiability problem and if we add the  $(n - r - 1)r + (n - r)(r + 1)$  clauses

$$(\bar{s}_j^k \vee s_{j+1}^k), \quad \text{for } 1 \leq j < n - r \text{ and } 1 \leq k \leq r, \quad (18)$$

$$(\bar{x}_{j+k} \vee \bar{s}_j^k \vee s_j^{k+1}), \quad \text{for } 1 \leq j \leq n - r \text{ and } 0 \leq k \leq r, \quad (19)$$

where  $s_j^k$  is omitted when  $k = 0$  and  $s_j^{k+1}$  is omitted when  $k = r$ , then the new set of clauses is satisfiable if and only if  $F$  is satisfiable with  $x_1 + \dots + x_n \leq r$ . (See exercise 26.) With this scheme we can limit the number of red-colored regions of McGregor’s graph to at most 7 by appending 1538 clauses in 721 new variables.

Another way to achieve the same goal, which turns out to be even better, has been proposed by O. Bailleux and Y. Boufkhad [LNCS **2833** (2003), 108–122]. Their method is a bit more difficult to describe, but still easy to implement: Consider a complete binary tree that has  $n - 1$  internal nodes numbered 1 through  $n - 1$ , and  $n$  leaves numbered  $n$  through  $2n - 1$ ; the children of node  $k$ , for  $1 \leq k < n$ , are nodes  $2k$  and  $2k + 1$  (see 2.3.4.5–(5)). We form new variables  $b_j^k$  for  $1 < k < n$  and  $1 \leq j \leq t_k$ , where  $t_k$  is the minimum of  $r$  and the number of leaves below node  $k$ . Then the following clauses, explained in exercise 27, do the job:

$$(\bar{b}_i^{2k} \vee \bar{b}_j^{2k+1} \vee b_{i+j}^k), \quad \text{for } 0 \leq i \leq t_{2k}, 0 \leq j \leq t_{2k+1}, 1 \leq i + j \leq t_k + 1, 1 < k < n; \quad (20)$$

$$(\bar{b}_i^2 \vee \bar{b}_j^3), \quad \text{for } 0 \leq i \leq t_2, 0 \leq j \leq t_3, i + j = r + 1. \quad (21)$$

In these formulas we let  $t_k = 1$  and  $b_1^k = x_{k-n+1}$  for  $n \leq k < 2n$ ; all literals  $\bar{b}_0^k$  and  $b_{r+1}^k$  are to be omitted. Applying (20) and (21) to McGregor’s graph, with  $n = 110$  and  $r = 7$ , yields just 1216 new clauses in 399 new variables.

The same ideas apply when we want to ensure that  $x_1 + \dots + x_n$  is at least  $r$ , because of the identity  $S_{\geq r}(x_1, \dots, x_n) = S_{\leq n-r}(\bar{x}_1, \dots, \bar{x}_n)$ . And exercise 30 considers the case of equality, when our goal is to make  $x_1 + \dots + x_n = r$ . We’ll discuss other encodings of such cardinality constraints below.

**Factoring integers.** Next on our agenda is a family of SAT instances with quite a different flavor. Given an  $(m + n)$ -bit binary integer  $z = (z_{m+n} \dots z_2 z_1)_2$ , do there exist integers  $x = (x_m \dots x_1)_2$  and  $y = (y_n \dots y_1)_2$  such that  $z = x \times y$ ? For example, if  $m = 2$  and  $n = 3$ , we want to invert the binary multiplication

$$\begin{array}{r} y_3 \ y_2 \ y_1 \\ \times \ x_2 \ x_1 \\ \hline a_3 \ a_2 \ a_1 \\ b_3 \ b_2 \ b_1 \\ \hline c_3 \ c_2 \ c_1 \\ \hline z_5 \ z_4 \ z_3 \ z_2 \ z_1 \end{array} \quad \begin{array}{l} z_1 = a_1 \\ (c_1 z_2)_2 = a_2 + b_1 \\ (c_2 z_3)_2 = a_3 + b_2 + c_1 \\ (c_3 z_4)_2 = b_3 + c_2 \\ z_5 = c_3 \end{array} \quad (22)$$

when the  $z$  bits are given. This problem is satisfiable when  $z = 21 = (10101)_2$ , in the sense that suitable binary values  $x_1, x_2, y_1, y_2, y_3, a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3$  do satisfy these equations. But it’s unsatisfiable when  $z = 19 = (10011)_2$ .

Arithmetical calculations like (22) are easily expressed in terms of clauses that can be fed to a SAT solver: We first specify the computation by constructing a Boolean chain, then we encode each step of the chain in terms of a few clauses. One such chain, if we identify  $a_1$  with  $z_1$  and  $c_3$  with  $z_5$ , is

$$\begin{aligned} z_1 \leftarrow x_1 \wedge y_1, & \quad b_1 \leftarrow x_2 \wedge y_1, & \quad z_2 \leftarrow a_2 \oplus b_1, & \quad s \leftarrow a_3 \oplus b_2, & \quad z_3 \leftarrow s \oplus c_1, & \quad z_4 \leftarrow b_3 \oplus c_2, \\ a_2 \leftarrow x_1 \wedge y_2, & \quad b_2 \leftarrow x_2 \wedge y_2, & \quad c_1 \leftarrow a_2 \wedge b_1, & \quad p \leftarrow a_3 \wedge b_2, & \quad q \leftarrow s \wedge c_1, & \quad z_5 \leftarrow b_3 \wedge c_2, \\ a_3 \leftarrow x_1 \wedge y_3, & \quad b_3 \leftarrow x_2 \wedge y_3, & & & & & c_2 \leftarrow p \vee q, \end{aligned} \quad (23)$$

using a “full adder” to compute  $c_2 z_3$  and “half adders” to compute  $c_1 z_2$  and  $c_3 z_4$  (see 7.1.2–(23) and (24)). And that chain is equivalent to the 49 clauses

$$(x_1 \vee \bar{z}_1) \wedge (y_1 \vee \bar{z}_1) \wedge (\bar{x}_1 \vee \bar{y}_1 \vee z_1) \wedge \cdots \wedge (\bar{b}_3 \vee \bar{c}_2 \vee \bar{z}_4) \wedge (b_3 \vee \bar{z}_5) \wedge (c_2 \vee \bar{z}_5) \wedge (\bar{b}_3 \vee \bar{c}_2 \vee z_5)$$

obtained by expanding the elementary computations according to simple rules:

$$\begin{aligned} t \leftarrow u \wedge v & \text{ becomes } (u \vee \bar{t}) \wedge (v \vee \bar{t}) \wedge (\bar{u} \vee \bar{v} \vee t); \\ t \leftarrow u \vee v & \text{ becomes } (\bar{u} \vee t) \wedge (\bar{v} \vee t) \wedge (u \vee v \vee \bar{t}); \\ t \leftarrow u \oplus v & \text{ becomes } (\bar{u} \vee v \vee t) \wedge (u \vee \bar{v} \vee t) \wedge (u \vee v \vee \bar{t}) \wedge (\bar{u} \vee \bar{v} \vee \bar{t}). \end{aligned} \quad (24)$$

To complete the specification of this factoring problem when, say,  $z = (10101)_2$ , we simply append the unary clauses  $(z_5) \wedge (\bar{z}_4) \wedge (z_3) \wedge (\bar{z}_2) \wedge (z_1)$ .

Logicians have known for a long time that computational steps can readily be expressed as conjunctions of clauses. Rules such as (24) are now called *Tseytin encoding*, after Gregory Tseytin (1966). Our representation of a small five-bit factorization problem in  $49+5$  clauses may not seem very efficient; but we will see shortly that  $m$ -bit by  $n$ -bit factorization corresponds to a satisfiability problem with fewer than  $6mn$  variables, and fewer than  $20mn$  clauses of length 3 or less.

*Even if the system has hundreds or thousands of formulas,  
it can be put into conjunctive normal form “piece by piece,”  
without any “multiplying out.”*

— MARTIN DAVIS and HILARY PUTNAM (1958)

Suppose  $m \leq n$ . The easiest way to set up Boolean chains for multiplication is probably to use a scheme that goes back to John Napier’s *Rabdologiæ* (Edinburgh, 1617), pages 137–143, as modernized by Luigi Dadda [*Alta Frequenza* **34** (1964), 349–356]: First we form all  $mn$  products  $x_i \wedge y_j$ , putting every such bit into  $\text{bin}[i+j]$ , which is one of  $m+n$  “bins” that hold bits to be added for a particular power of 2 in the binary number system. The bins will contain respectively  $(0, 1, 2, \dots, m, m, \dots, m, \dots, 2, 1)$  bits at this point, with  $n-m+1$  occurrences of “ $m$ ” in the middle. Now we look at  $\text{bin}[k]$  for  $k = 2, 3, \dots$ . If  $\text{bin}[k]$  contains a single bit  $b$ , we simply set  $z_{k-1} \leftarrow b$ . If it contains two bits  $\{b, b'\}$ , we use a half adder to compute  $z_{k-1} \leftarrow b \oplus b'$ ,  $c \leftarrow b \wedge b'$ , and we put the carry bit  $c$  into  $\text{bin}[k+1]$ . Otherwise  $\text{bin}[k]$  contains  $t \geq 3$  bits; we choose any three of them, say  $\{b, b', b''\}$ , and remove them from the bin. With a full adder we then compute  $r \leftarrow b \oplus b' \oplus b''$  and  $c \leftarrow \langle bb'b'' \rangle$ , so that  $b+b'+b'' = r+2c$ ; and we put  $r$  into  $\text{bin}[k]$ ,  $c$  into  $\text{bin}[k+1]$ . This decreases  $t$  by 2, so eventually we will have computed  $z_{k-1}$ . Exercise 41 quantifies the exact amount of calculation involved.

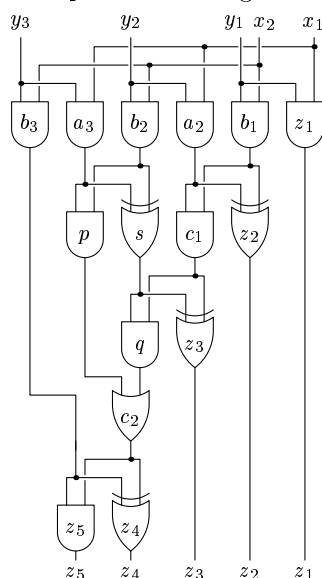
Boolean chain  
full adder  
half adders  
unary clauses  
Tseytin encoding  
conjunctive normal form  
DAVIS  
PUTNAM  
Napier  
Dadda  
binary number system  
half adder  
carry bit  
full adder  
notation  $\langle abc \rangle$   
median operation  
ternary operations

This method of encoding multiplication into clauses is quite flexible, since we're allowed to choose *any* three bits from  $bin[k]$  whenever four or more bits are present. We could use a first-in-first-out strategy, always selecting bits from the "rear" and placing their sum at the "front"; or we could work last-in-first-out, essentially treating  $bin[k]$  as a stack instead of a queue. We could also select the bits randomly, to see if this makes our SAT solver any happier. Later in this section we'll refer to the clauses that represent the factoring problem by calling them  $factor\_fifo(m, n, z)$ ,  $factor\_lifo(m, n, z)$ , or  $factor\_rand(m, n, z, s)$ , respectively, where  $s$  is a seed for the random number generator used to generate them.

It's somewhat mind-boggling to realize that numbers can be factored without using any number theory! No greatest common divisors, no applications of Fermat's theorems, etc., are anywhere in sight. We're providing no hints to the solver except for a bunch of Boolean formulas that operate almost blindly at the bit level. Yet factors are found.

Of course we can't expect this method to compete with the sophisticated factorization algorithms of Section 4.5.4. But the problem of factoring does demonstrate the great versatility of clauses. And its clauses can be combined with *other* constraints that go well beyond any of the problems we've studied before.

**Fault testing.** Lots of things can go wrong when computer chips are manufactured in the "real world," so engineers have long been interested in constructing test patterns to check the validity of a particular circuit. For example, suppose that all but one of the logical elements are functioning properly in some chip; the bad one, however, is stuck: Its output is constant, always the same regardless of the inputs that it is given. Such a failure is called a *single-stuck-at fault*.



**Fig. 34.** A circuit that corresponds to (23).

Figure 34 illustrates a typical digital circuit in detail: It implements the 15 Boolean operations of (23) as a network that produces five output signals  $z_5 z_4 z_3 z_2 z_1$  from the five inputs  $y_3 y_2 y_1 x_2 x_1$ . In addition to having 15 AND, OR, and XOR gates, each of which transforms two inputs into one output, it has 15 "fanout" gates (indicated by dots at junction points), each of which splits one input into two outputs. As a result it comprises 50 potentially distinct logical signals, one for each internal "wire." Exercise 47 shows that a circuit with  $m$  outputs,  $n$  inputs, and  $g$  conventional 2-to-1 gates will have  $g + m - n$  fanout gates and  $3g + 2m - n$  wires. A circuit with  $w$  wires has  $2w$  possible single-stuck-at faults, namely  $w$  faults in which the signal on a wire is stuck at 0 and  $w$  more on which it is stuck at 1.

Table 1 shows 101 scenarios that are possible when the 50 wires of Fig. 34 are activated by one particular sequence of inputs, assuming that at

first-in-first-out  
 FIFO: first in first out  
 last-in-first-out  
 stack  
 queue  
 $factor\_fifo(m, n, z)$   
 $factor\_lifo(m, n, z)$   
 $factor\_rand(m, n, z, s)$   
 Fermat  
 Fault testing  
 ATPG: Automatic test pattern generation,  
 test patterns  
 circuit  
 circuit: see also Boolean chain  
 single-stuck-at fault  
 fanout





inspection shows, for instance, that the pattern considered in Table 1 doesn't detect an error when  $q$  is stuck at 1, even though  $q$  should be 0, because all five output bits  $z_5 z_4 z_3 z_2 z_1$  are correct in spite of that error. In fact, the value of  $c_1 \leftarrow p \vee q$  is unaffected by a bad  $q$ , because  $p = 1$  in this example. Similarly, the fault " $x_1^2$  stuck at 0" doesn't propagate into  $z_1 \leftarrow x_1^2 \wedge y_1^1$  because  $y_1^1 = 0$ . Altogether 44 faults, not 50, are discovered by this particular test pattern.

All of the relevant repeatable faults, whether they're single-stuck-at or wildly complicated, could obviously be discovered by testing all  $2^n$  possible patterns. But that's out of the question unless  $n$  is quite small. Fortunately, testing isn't hopeless, because satisfactory results are usually obtained in practice if we do have enough test patterns to detect all of the detectable single-stuck-at faults. Exercise 49 shows that just five patterns suffice to certify Fig. 34 by this criterion.

The detailed analysis in exercise 49 also shows, surprisingly, that one of the faults, namely " $s^2$  stuck at 1," cannot be detected! Indeed, an erroneous  $s^2$  can propagate to an erroneous  $q$  only if  $c_1^2 = 1$ , and that forces  $x_1 = x_2 = y_1 = y_2 = 1$ ; only two possibilities remain, and neither  $y_3 = 0$  nor  $y_3 = 1$  reveals the fault. Consequently we can simplify the circuit by removing gate  $q$ ; the chain (23) becomes shorter, with " $q \leftarrow s \wedge c_1, c_2 \leftarrow p \vee q$ " replaced by " $c_2 \leftarrow p \vee c_1$ ."

Of course Fig. 34 is just a tiny little circuit, intended only to introduce the concept of stuck-at faults. Test patterns are needed for the much larger circuits that arise in real computers; and we will see that SAT solvers can help us to find them. Consider, for example, the generic multiplier circuit *prod*( $m, n$ ), which is part of the Stanford GraphBase. It multiplies an  $m$ -bit number  $x$  by an  $n$ -bit number  $y$ , producing an  $(m + n)$ -bit product  $z$ . Furthermore, it's a so-called "parallel multiplier," with delay time  $O(\log(m + n))$ ; thus it's much more suited to hardware design than methods like the *factor\_fifo* schemes that we considered above, because those circuits need  $\Omega(m + n)$  time for carries to propagate.

Let's try to find test patterns that will smoke out all of the single-stuck-at faults in *prod*(32, 32), which is a circuit of depth 33 that has 64 inputs, 64 outputs, 3660 AND gates, 1203 OR gates, 2145 XOR gates, and (therefore) 7008 fan-out gates and 21,088 wires. How can we guard it against 42,176 different faults?

Before we construct clauses to facilitate that task, we should realize that most of the single-stuck-at faults are easily detected by choosing patterns at random, since faults usually cause big trouble and are hard to miss. Indeed, choosing  $x = \#3243F6A8$  and  $y = \#885A308D$  more or less at random already eliminates 14,733 cases; and  $(x, y) = (\#2B7E1516, \#28AED2A6)$  eliminates 6,918 more. We might as well keep doing this, because bitwise operations such as those in Table 1 are fast. Experience with the smaller multiplier in Fig. 34 suggests that we get more effective tests if we bias the inputs, choosing each bit to be 1 with probability .9 instead of .5 (see exercise 49). A million such random inputs will then generate, say, 243 patterns that detect all but 140 of the faults.

Our remaining job, then, is essentially to find 140 needles in a haystack of size  $2^{64}$ , after having picked  $42,176 - 140 = 42,036$  pieces of low-hanging fruit. And that's where a SAT solver is useful. Consider, for example, the analogous but simpler problem of finding a test pattern for " $q$  stuck at 0" in Fig. 34.

Boolean evaluation  
*prod*  
 Stanford GraphBase  
 multiplies  
 parallel multiplier  
 factor\_fifo  
 fanout gates  
 wires  
 random  
 pi, as source of random data  
 e, as source of random data  
 bitwise operations

We can use the 49 clauses  $F$  derived from (23) to represent the well-behaved circuit; and we can imagine corresponding clauses  $F'$  that represent the faulty computation, using “primed” variables  $z'_1, a'_2, \dots, z'_5$ . Thus  $F'$  begins with  $(x_1 \vee \bar{z}'_1) \wedge (y_1 \vee \bar{z}'_1)$  and ends with  $(\bar{b}'_3 \vee \bar{c}'_2 \vee z'_5)$ ; it’s like  $F$  except that the clauses representing  $q' \leftarrow s' \wedge c'_1$  in (23) are changed to simply  $\bar{q}'$  (meaning that  $q'$  is stuck at 0). Then the clauses of  $F$  and  $F'$ , together with a few more clauses to state that  $z_1 \neq z'_1$  or  $\dots$  or  $z_5 \neq z'_5$ , will be satisfiable only by variables for which  $(y_3 y_2 y_1)_2 \times (x_2 x_1)_2$  is a suitable test pattern for the given fault.

This construction of  $F'$  can obviously be simplified, because  $z'_1$  is identical to  $z_1$ ; any signal that differs from the correct value must be located “downstream” from the one-and-only fault. Let’s say that a wire is *tarnished* if it is the faulty wire or if at least one of its input wires is tarnished. We introduce new variables  $g'$  only for wires  $g$  that are tarnished. Thus, in our example, the only clauses  $F'$  that are needed to extend  $F$  to a faulty companion circuit are  $\bar{q}'$  and the clauses that correspond to  $c'_2 \leftarrow p \vee q'$ ,  $z'_4 \leftarrow b_3 \oplus c'_2$ ,  $z'_5 \leftarrow b_3 \wedge c'_2$ .

Moreover, any fault that is revealed by a test pattern must have an *active path* of wires, leading from the fault to an output; all wires on this path must carry a faulty signal. Therefore Tracy Larrabee [IEEE Trans. CAD-11 (1992), 4–15] decided to introduce additional “sharped” variables  $g^\sharp$  for each tarnished wire, meaning that  $g$  lies on the active path. The two clauses

$$(\bar{g}^\sharp \vee g \vee g') \wedge (\bar{g}^\sharp \vee \bar{g} \vee \bar{g}') \quad (25)$$

ensure that  $g \neq g'$  whenever  $g$  is part of that path. Furthermore we have  $(\bar{v}^\sharp \vee g^\sharp)$  whenever  $g$  is an AND, OR, or XOR gate with tarnished input  $v$ . Fanout gates are slightly tricky in this regard: When wires  $g^1$  and  $g^2$  fan out from a tarnished wire  $g$ , we need variables  $g^{1\sharp}$  and  $g^{2\sharp}$  as well as  $g^\sharp$ ; and we introduce the clause

$$(\bar{g}^\sharp \vee g^{1\sharp} \vee g^{2\sharp}) \quad (26)$$

to specify that the active path takes at least one of the two branches.

According to these rules, our example acquires the new variables  $q^\sharp, c_2^\sharp, c_2^{1\sharp}, c_2^{2\sharp}, z_4^\sharp, z_5^\sharp$ , and the new clauses

$$(\bar{q}^\sharp \vee q \vee q') \wedge (\bar{q}^\sharp \vee \bar{q} \vee \bar{q}') \wedge (\bar{q}^\sharp \vee c_2^\sharp) \wedge (\bar{c}_2^\sharp \vee c_2 \vee c'_2) \wedge (\bar{c}_2^\sharp \vee \bar{c}_2 \vee \bar{c}'_2) \wedge (\bar{c}_2^\sharp \vee c_2^{1\sharp} \vee c_2^{2\sharp}) \wedge (\bar{c}_2^{1\sharp} \vee z_4^\sharp) \wedge (\bar{c}_2^{2\sharp} \vee z_4^\sharp \vee z'_4) \wedge (\bar{z}_4^\sharp \vee \bar{z}_4 \vee \bar{z}'_4) \wedge (\bar{z}_2^\sharp \vee z_5^\sharp) \wedge (\bar{z}_5^\sharp \vee z_5 \vee z'_5) \wedge (\bar{z}_5^\sharp \vee \bar{z}_5 \vee \bar{z}'_5).$$

The active path begins at  $q$ , so we assert the unit clause  $(q^\sharp)$ ; it ends at a tarnished output, so we also assert  $(z_4^\sharp \vee z_5^\sharp)$ . The resulting set of clauses will find a test pattern for this fault if and only if the fault is detectable. Larrabee found that such active-path variables provide important clues to a SAT solver and significantly speed up the solution process.

Returning to the large circuit *prod* (32, 32), one of the 140 hard-to-test faults is “ $W_{21}^{26}$  stuck at 1,” where  $W_{21}^{26}$  denotes the 26th extra wire that fans out from the OR gate called  $W_{21}$  in §75 of the Stanford GraphBase program GB\_GATES;  $W_{21}^{26}$  is an input to gate  $b_{40}^{40} \leftarrow d_{40}^{19} \wedge W_{21}^{26}$  in §80 of that program. Test patterns for that fault can be characterized by a set of 23,194 clauses in 7,082 variables

tarnished  
active path  
Larrabee  
AND  
OR  
XOR  
Fanout gates  
unit clause  
SAT solver  
Stanford GraphBase  
GB\_GATES

(of which only 4 variables are “primed” and 4 are “sharped”). Fortunately the solution  $(x, y) = (\#7F13FEDD, \#5FE57FFE)$  was found rather quickly in the author’s experiments; and this pattern also killed off 13 of the other cases, so the score was now “14 down and 126 to go”!

The next fault sought was “ $A_5^{36,2}$  stuck at 1,” where  $A_5^{36,2}$  is the second extra wire to fan out from the AND gate  $A_5^{36}$  in §72 of GB\_GATES (an input to  $R_{11}^{36} \leftarrow A_5^{36,2} \wedge R_1^{35,2}$ ). This fault corresponds to 26,131 clauses on 8,342 variables; but the SAT solver took a quick look at those clauses and decided almost instantly that they are *unsatisfiable*. Therefore the fault is undetectable, and the circuit *prod*(32, 32) can be simplified by setting  $R_{11}^{36} \leftarrow R_1^{35,2}$ . A closer look showed, in fact, that clauses corresponding to the Boolean equations

$$x = y \wedge z, \quad y = v \wedge w, \quad z = t \wedge u, \quad u = v \oplus w$$

were present (where  $t = R_{13}^{44}$ ,  $u = A_{58}^{45}$ ,  $v = R_4^{44}$ ,  $w = A_{14}^{45}$ ,  $x = R_{23}^{46}$ ,  $y = R_{13}^{45}$ ,  $z = R_{19}^{45}$ ); these clauses *force*  $x = 0$ . Therefore it was not surprising to find that the list of unresolved faults also included  $R_{23}^{46}$ ,  $R_{23}^{46,1}$  and  $R_{23}^{46,2}$  stuck at 0. Altogether 26 of the 140 faults undetected by random inputs turned out to be *absolutely* undetectable; and only one of these, namely “ $Q_{26}^{46}$  stuck at 0,” required a nontrivial proof of undetectability.

Some of the  $126 - 26 = 100$  faults remaining on the to-do list turned out to be significant challenges for the SAT solver. While waiting, the author therefore had time to take a look at a few of the previously found solutions, and noticed that those patterns themselves were forming a pattern! Sure enough, the extreme portions of this large and complicated circuit actually have a fairly simple structure, stuck-at-fault-wise. Hence number theory came to the rescue: The factorization  $\#87FBC059 \times \#F0F87817 = 2^{63} - 1$  solved many of the toughest challenges, some of which occur with probability less than  $2^{-34}$  when 32-bit numbers are multiplied; and the “Aurifeuillian” factorization  $(2^{31} - 2^{16} + 1)(2^{31} + 2^{16} + 1) = 2^{62} + 1$ , which the author had known for more than forty years (see Eq. 4.5.4–(15)), polished off most of the others.

The bottom line (see exercise 51) is that all 42,150 of the detectable single-stuck-at faults of the parallel multiplication circuit *prod*(32, 32) can actually be detected with at most 196 well-chosen test patterns.

**Learning a Boolean function.** Sometimes we’re given a “black box” that evaluates a Boolean function  $f(x_1, \dots, x_N)$ . We have no way to open the box, but we suspect that the function is actually quite simple. By plugging in various values for  $x = x_1 \dots x_N$ , we can observe the box’s behavior and possibly learn the hidden rule that lies inside. For example, a secret function of  $N = 20$  Boolean variables might take on the values shown in Table 2, which lists 16 cases where  $f(x) = 1$  and 16 cases where  $f(x) = 0$ .

Suppose we assume that the function has a DNF (disjunctive normal form) with only a few terms. We’ll see in a moment that it’s easy to express such an assumption as a satisfiability problem. And when the author constructed clauses corresponding to Table 2 and presented them to a SAT solver, he did in fact learn

Knuth  
GB\_GATES  
number theory  
Aurifeuillian  
learning a Boolean function–  
Boolean function–  
DNF  
disjunctive normal form

**Table 2**  
VALUES TAKEN ON BY AN UNKNOWN FUNCTION

| Cases where $f(x) = 1$ |       |       |       |       |       |       |       |       |     |          | Cases where $f(x) = 0$ |       |       |       |       |       |       |       |       |     |          |   |   |   |
|------------------------|-------|-------|-------|-------|-------|-------|-------|-------|-----|----------|------------------------|-------|-------|-------|-------|-------|-------|-------|-------|-----|----------|---|---|---|
| $x_1$                  | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | ... | $x_{20}$ | $x_1$                  | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | ... | $x_{20}$ |   |   |   |
| 1                      | 1     | 0     | 0     | 1     | 0     | 0     | 1     | 0     | 0   | 0        | 0                      | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 0     | 1   | 1        | 0 | 1 |   |
| 1                      | 0     | 1     | 0     | 1     | 0     | 1     | 0     | 0     | 0   | 1        | 0                      | 0     | 0     | 1     | 0     | 0     | 0     | 1     | 0     | 0   | 0        | 1 | 0 | 1 |
| 0                      | 1     | 1     | 0     | 1     | 0     | 0     | 0     | 1     | 1   | 0        | 0                      | 0     | 0     | 1     | 0     | 0     | 0     | 1     | 0     | 0   | 0        | 1 | 1 | 1 |
| 0                      | 1     | 0     | 0     | 1     | 1     | 0     | 0     | 0     | 1   | 0        | 0                      | 1     | 1     | 0     | 0     | 0     | 1     | 1     | 0     | 0   | 1        | 1 | 0 | 1 |
| 0                      | 0     | 0     | 0     | 1     | 0     | 1     | 0     | 0     | 0   | 1        | 0                      | 1     | 1     | 1     | 0     | 0     | 0     | 1     | 1     | 1   | 0        | 0 | 0 | 1 |
| 0                      | 0     | 0     | 0     | 1     | 1     | 0     | 1     | 1     | 0   | 0        | 0                      | 0     | 0     | 0     | 1     | 1     | 1     | 0     | 0     | 1   | 1        | 1 | 0 | 0 |
| 1                      | 1     | 0     | 1     | 0     | 0     | 0     | 1     | 0     | 0   | 1        | 0                      | 1     | 0     | 0     | 1     | 0     | 0     | 1     | 0     | 0   | 0        | 0 | 0 | 0 |
| 0                      | 0     | 1     | 0     | 0     | 1     | 0     | 0     | 1     | 1   | 1        | 0                      | 0     | 0     | 0     | 0     | 0     | 1     | 0     | 0     | 0   | 1        | 0 | 0 | 0 |
| 1                      | 0     | 0     | 1     | 0     | 1     | 0     | 0     | 1     | 1   | 0        | 0                      | 1     | 1     | 1     | 1     | 1     | 0     | 0     | 1     | 1   | 1        | 0 | 0 | 1 |
| 1                      | 1     | 0     | 0     | 0     | 1     | 1     | 0     | 1     | 1   | 1        | 0                      | 1     | 1     | 1     | 1     | 0     | 0     | 0     | 0     | 0   | 0        | 1 | 0 | 0 |
| 1                      | 0     | 0     | 0     | 1     | 0     | 1     | 1     | 0     | 0   | 1        | 0                      | 0     | 1     | 1     | 0     | 0     | 0     | 0     | 0     | 0   | 0        | 1 | 0 | 0 |
| 1                      | 1     | 0     | 0     | 0     | 1     | 1     | 0     | 1     | 1   | 0        | 0                      | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0   | 0        | 1 | 0 | 0 |
| 0                      | 0     | 0     | 0     | 1     | 0     | 1     | 1     | 0     | 1   | 1        | 1                      | 1     | 1     | 0     | 1     | 0     | 1     | 0     | 1     | 0   | 1        | 0 | 1 | 0 |
| 0                      | 1     | 1     | 0     | 0     | 0     | 1     | 1     | 1     | 0   | 1        | 1                      | 0     | 0     | 0     | 1     | 0     | 0     | 1     | 0     | 0   | 1        | 1 | 0 | 1 |
| 1                      | 0     | 0     | 1     | 1     | 0     | 1     | 1     | 0     | 0   | 1        | 0                      | 0     | 0     | 1     | 0     | 0     | 1     | 0     | 1     | 0   | 1        | 0 | 1 | 0 |
| 0                      | 0     | 0     | 1     | 0     | 1     | 0     | 0     | 1     | 0   | 1        | 0                      | 0     | 0     | 0     | 0     | 1     | 0     | 0     | 0     | 1   | 0        | 0 | 0 | 0 |
| 0                      | 0     | 1     | 1     | 0     | 0     | 1     | 0     | 0     | 1   | 0        | 0                      | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0   | 1        | 0 | 0 | 0 |
| 0                      | 1     | 1     | 1     | 1     | 0     | 0     | 1     | 1     | 0   | 0        | 0                      | 1     | 1     | 1     | 0     | 0     | 0     | 1     | 1     | 0   | 0        | 0 | 1 | 1 |
| 0                      | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 1   | 0        | 0                      | 1     | 1     | 0     | 1     | 1     | 0     | 1     | 1     | 1   | 0        | 1 | 1 | 0 |
| 1                      | 0     | 1     | 1     | 0     | 0     | 0     | 0     | 0     | 1   | 0        | 0                      | 1     | 1     | 1     | 1     | 0     | 1     | 1     | 1     | 1   | 1        | 0 | 0 | 1 |

almost immediately that a very simple formula is consistent with all of the data:

$$f(x_1, \dots, x_{20}) = \bar{x}_2 \bar{x}_3 \bar{x}_{10} \vee \bar{x}_6 \bar{x}_{10} \bar{x}_{12} \vee x_8 \bar{x}_{13} \bar{x}_{15} \vee \bar{x}_8 x_{10} \bar{x}_{12}. \quad (27)$$

This formula was discovered by constructing clauses in  $2MN$  variables  $p_{i,j}$  and  $q_{i,j}$  for  $1 \leq i \leq M$  and  $1 \leq j \leq N$ , where  $M$  is the maximum number of terms allowed in the DNF (here  $M = 4$ ) and where

$$p_{i,j} = [\text{term } i \text{ contains } x_j], \quad q_{i,j} = [\text{term } i \text{ contains } \bar{x}_j]. \quad (28)$$

If the function is constrained to equal 1 at  $P$  specified points, we also use auxiliary variables  $z_{i,k}$  for  $1 \leq i \leq M$  and  $1 \leq k \leq P$ , one for each term at every such point.

Table 2 says that  $f(1, 1, 0, 0, \dots, 1) = 1$ , and we can capture this specification by constructing the clause

$$(z_{1,1} \vee z_{2,1} \vee \dots \vee z_{M,1}) \quad (29)$$

together with the clauses

$$(\bar{z}_{i,1} \vee \bar{q}_{i,1}) \wedge (\bar{z}_{i,1} \vee \bar{q}_{i,2}) \wedge (\bar{z}_{i,1} \vee \bar{p}_{i,3}) \wedge (\bar{z}_{i,1} \vee \bar{p}_{i,4}) \wedge \dots \wedge (\bar{z}_{i,1} \vee \bar{q}_{i,20}) \quad (30)$$

for  $1 \leq i \leq M$ . Translation: (29) says that at least one of the terms in the DNF must evaluate to true; and (30) says that, if term  $i$  is true at the point  $1100\dots 1$ , it cannot contain  $\bar{x}_1$  or  $\bar{x}_2$  or  $x_3$  or  $x_4$  or  $\dots$  or  $\bar{x}_{20}$ .

Table 2 also tells us that  $f(1, 0, 1, 0, \dots, 1) = 0$ . This specification corresponds to the clauses

$$(q_{i,1} \vee p_{i,2} \vee q_{i,3} \vee p_{i,4} \vee \dots \vee q_{i,20}) \quad (31)$$

for  $1 \leq i \leq M$ . (Each term of the DNF must be zero at the given point; thus either  $\bar{x}_1$  or  $x_2$  or  $\bar{x}_3$  or  $x_4$  or  $\dots$  or  $\bar{x}_{20}$  must be present for each value of  $i$ .)

In general, every case where  $f(x) = 1$  yields one clause like (29) of length  $M$ , plus  $MN$  clauses like (30) of length 2. Every case where  $f(x) = 0$  yields  $M$  clauses like (31) of length  $N$ . We use  $q_{i,j}$  when  $x_j = 1$  at the point in question,

and  $p_{i,j}$  when  $x_j = 0$ , for both (30) and (31). This construction is due to A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende [*Mathematical Programming* **57** (1992), 215–238], who presented many examples. From Table 2, with  $M = 4$ ,  $N = 20$ , and  $P = 16$ , it generates 1360 clauses of total length 3904 in 224 variables; a SAT solver then finds a solution with  $p_{1,1} = q_{1,1} = p_{1,2} = 0$ ,  $q_{1,2} = 1, \dots$ , leading to (27).

The simplicity of (27) makes it plausible that the SAT solver has indeed psyched out the true nature of the hidden function  $f(x)$ . The chance of agreeing with the correct value 32 times out of 32 is only 1 in  $2^{32}$ , so we seem to have overwhelming evidence in favor of that equation.

But no: Such reasoning is fallacious. The numbers in Table 2 actually arose in a completely different way, and Eq. (27) has essentially *no* credibility as a predictor of  $f(x)$  for any other values of  $x$ ! (See exercise 53.) The fallacy comes from the fact that short-DNF Boolean functions of 20 variables are not at all rare; there are many more than  $2^{32}$  of them.

On the other hand, when we *do* know that the hidden function  $f(x)$  has a DNF with at most  $M$  terms (although we know nothing else about it), the clauses (29)–(31) give us a nice way to discover those terms, provided that we also have a sufficiently large and unbiased “training set” of observed values.

For example, let’s assume that (27) actually *is* the function in the box. If we examine  $f(x)$  at 32 random points  $x$ , we don’t have enough data to make any deductions. But 100 random training points will almost always home in on the correct solution (27). This calculation typically involves 3942 clauses in 344 variables; yet it goes quickly, needing only about 100 million accesses to memory.

One of the author’s experiments with a 100-element training set yielded

$$\hat{f}(x_1, \dots, x_{20}) = \bar{x}_2 \bar{x}_3 \bar{x}_{10} \vee x_3 \bar{x}_6 \bar{x}_{10} \bar{x}_{12} \vee x_8 \bar{x}_{13} \bar{x}_{15} \vee \bar{x}_8 x_{10} \bar{x}_{12}, \quad (32)$$

which is close to the truth but not quite exact. (Exercise 59 proves that  $\hat{f}(x)$  is equal to  $f(x)$  more than 97% of the time.) Further study of this example showed that another nine training points were enough to deduce  $f(x)$  uniquely, thus obtaining 100% confidence (see exercise 61).

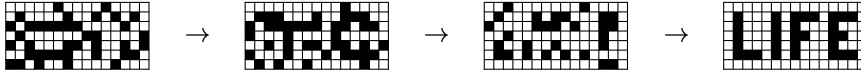
**Bounded model checking.** Some of the most important applications of SAT solvers in practice are related to the verification of hardware or software, because designers generally want some kind of assurance that particular implementations correctly meet their specifications.

A typical design can usually be modeled as a *transition relation* between Boolean vectors  $X = x_1 \dots x_n$  that represent the possible states of a system. We write  $X \rightarrow X'$  if state  $X$  at time  $t$  can be followed by state  $X'$  at time  $t + 1$ . The task in general is to study sequences of state transitions

$$X_0 \rightarrow X_1 \rightarrow X_2 \rightarrow \dots \rightarrow X_r, \quad (33)$$

and to decide whether or not there are sequences that have special properties. For example, we hope that there’s no such sequence for which  $X_0$  is an “initial state” and  $X_r$  is an “error state”; otherwise there’d be a bug in the design.

Kamath  
Karmarkar  
Ramakrishnan  
Resende  
fallacious  
training set  
author  
bounded model checking-  
verification  
model checking+  
transition relation  
dynamical system, discrete  
bug



**Fig. 35.** Conway's rule (35) defines these three successive transitions.

Questions like this are readily expressed as satisfiability problems: Each state  $X_t$  is a vector of Boolean variables  $x_{t1} \dots x_{tn}$ , and each transition relation can be represented by a set of  $m$  clauses  $T(X_t, X_{t+1})$  that must be satisfied. These clauses  $T(X, X')$  involve  $2n$  variables  $\{x_1, \dots, x_n, x'_1, \dots, x'_n\}$ , together with  $q$  auxiliary variables  $\{y_1, \dots, y_q\}$  that might be needed to express Boolean formulas in clause form as we did with the Tseytin encodings in (24). Then the existence of sequence (33) is equivalent to the satisfiability of  $mr$  clauses

$$T(X_0, X_1) \wedge T(X_1, X_2) \wedge \dots \wedge T(X_{r-1}, X_r) \quad (34)$$

in the  $n(r+1) + qr$  variables  $\{x_{tj} \mid 0 \leq t \leq r, 1 \leq j \leq n\} \cup \{y_{tk} \mid 0 \leq t < r, 1 \leq k \leq q\}$ . We've essentially "unrolled" the sequence (33) into  $r$  copies of the transition relation, using variables  $x_{tj}$  for state  $X_t$  and  $y_{tk}$  for the auxiliary quantities in  $T(X_t, X_{t+1})$ . Additional clauses can now be added to specify constraints on the initial state  $X_0$  and/or the final state  $X_r$ , as well as any other conditions that we want to impose on the sequence.

This general setup is called "bounded model checking," because we're using it to check properties of a model (a transition relation), and because we're considering only sequences that have a bounded number of transitions,  $r$ .

John Conway's fascinating *Game of Life* provides a particularly instructive set of examples that illustrate basic principles of bounded model checking. The states  $X$  of this game are two-dimensional bitmaps, corresponding to arrays of square cells that are either alive (1) or dead (0). Every bitmap  $X$  has a unique successor  $X'$ , determined by the action of a simple  $3 \times 3$  cellular automaton: Suppose cell  $x$  has the eight neighbors  $\{x_{NW}, x_N, x_{NE}, x_W, x_E, x_{SW}, x_S, x_{SE}\}$ , and let  $\nu = x_{NW} + x_N + x_{NE} + x_W + x_E + x_{SW} + x_S + x_{SE}$  be the number of neighbors that are alive at time  $t$ . Then  $x$  is alive at time  $t + 1$  if and only if either (a)  $\nu = 3$ , or (b)  $\nu = 2$  and  $x$  is alive at time  $t$ . Equivalently, the transition rule

$$x' = [2 < x_{NW} + x_N + x_{NE} + x_W + \frac{1}{2}x + x_E + x_{SW} + x_S + x_{SE} < 4] \quad (35)$$

holds at every cell  $x$ . (See, for example, Fig. 35, where the live cells are black.)

Conway called Life a "no-player game," because it involves no strategy: Once an initial state  $X_0$  has been set up, all subsequent states  $X_1, X_2, \dots$  are completely determined. Yet, in spite of the simple rules, he also proved that Life is inherently complicated and unpredictable, indeed beyond human comprehension, in the sense that it is universal: *Every finite, discrete, deterministic system, however complex, can be simulated faithfully by some finite initial state  $X_0$  of Life.* [See Berlekamp, Conway, and Guy, *Winning Ways* (2004), Chapter 25.]

In exercises 7.1.4–160 through 162, we've already seen some of the amazing Life histories that are possible, using BDD methods. And many further aspects of Life can be explored with SAT methods, because SAT solvers can often deal

auxiliary variables  
Tseytin encodings  
model  
Conway  
Life  
bitmaps  
cellular automaton  
no-player game  
universal  
Berlekamp  
Guy  
BDD+

with many more variables. For example, Fig. 35 was discovered by using  $7 \times 15 = 105$  variables for each state  $X_0, X_1, X_2, X_3$ . The values of  $X_3$  were obviously predetermined; but the other  $105 \times 3 = 315$  variables had to be computed, and BDDs can't handle that many. Moreover, additional variables were introduced to ensure that the initial state  $X_0$  would have as few live cells as possible.

encoded  
chessboard  
grid  
mobile

Here's the story behind Fig. 35, in more detail: Since Life is two-dimensional, we use variables  $x_{ij}$  instead of  $x_j$  to indicate the states of individual cells, and  $x_{tij}$  instead of  $x_{ij}$  to indicate the states of cells at time  $t$ . We generally assume that  $x_{tij} = 0$  for all cells outside of a given finite region, although the transition rule (35) can allow cells that are arbitrarily far away to become alive as Life goes on. In Fig. 35 the region was specified to be a  $7 \times 15$  rectangle at each unit of time. Furthermore, configurations with three consecutive live cells on a boundary edge were forbidden, so that cells "outside the box" wouldn't be activated.

The transitions  $T(X_t, X_{t+1})$  can be encoded without introducing additional variables, but only if we introduce 190 rather long clauses for each cell not on the boundary. There's a better way, based on the binary tree approach underlying (20) and (21) above, which requires only about 63 clauses of size  $\leq 3$ , together with about 14 auxiliary variables per cell. This approach (see exercise 65) takes advantage of the fact that many intermediate calculations can be shared. For example, cells  $x$  and  $x_w$  have four neighbors  $\{x_{nw}, x_n, x_{sw}, x_s\}$  in common; so we need to compute  $x_{nw} + x_n + x_{sw} + x_s$  only once, not twice.

The clauses that correspond to a four-step sequence  $X_0 \rightarrow X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4$  leading to  $X_4 = \mathbf{LIFE}$  turn out to be unsatisfiable without going outside of the  $7 \times 15$  frame. (Only 10 gigamems of calculation were needed to establish this fact, using Algorithm C below, even though roughly 34000 clauses in 9000 variables needed to be examined!) So the next step in the preparation of Fig. 35 was to try  $X_3 = \mathbf{LIFE}$ ; and this trial succeeded. Additional clauses, which permitted  $X_0$  to have at most 39 live cells, led to the solution shown, at a cost of about 17 gigamems; and that solution is optimum, because a further run (costing 12 gigamems) proved that there's no solution with at most 38.

Let's look for a moment at some of the patterns that can occur on a chessboard, an  $8 \times 8$  grid. Human beings will never be able to contemplate more than a tiny fraction of the  $2^{64}$  states that are possible; so we can be fairly sure that "Lifenthusiasts" haven't already explored every tantalizing configuration that exists, even on such a small playing field.

One nice way to look for a sequence of interesting Life transitions is to assert that no cell stays alive more than four steps in a row. Let us therefore say that a *mobile* Life path is a sequence of transitions  $X_0 \rightarrow X_1 \rightarrow \cdots \rightarrow X_r$  with the additional property that we have

$$(\bar{x}_{tij} \vee \bar{x}_{(t+1)ij} \vee \bar{x}_{(t+2)ij} \vee \bar{x}_{(t+3)ij} \vee \bar{x}_{(t+4)ij}), \quad \text{for } 0 \leq t \leq r - 4. \quad (36)$$

To avoid trivial solutions we also insist that  $X_r$  is not entirely dead. For example, if we impose rule (36) on a chessboard, with  $x_{tij}$  permitted to be alive only if  $1 \leq i, j \leq 8$ , and with the further condition that at most five cells are alive in each

generation, a SAT solver can quickly discover interesting mobile paths such as

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \end{array} \rightarrow \cdots, \quad (37)$$

which last quite awhile before leaving the board. And indeed, the five-celled object that moves so gracefully in this path is R. K. Guy's famous *glider* (1970), which is surely the most interesting small creature in Life's universe. The glider moves diagonally, recreating a shifted copy of itself after every four steps.

Interesting mobile paths appear also if we restrict the population at each time to  $\{6, 7, 8, 9, 10\}$  instead of  $\{1, 2, 3, 4, 5\}$ . For example, here are some of the first such paths that the author's solver came up with, having length  $r = 8$ :

$$\begin{array}{l} \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \end{array}; \\ \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \end{array}; \\ \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \end{array}; \\ \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \end{array}; \\ \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \end{array}. \end{array}$$

These paths illustrate the fact that symmetry can be gained, but never lost, as Life evolves deterministically. Marvelous designs are spawned in the process. In each of these sequences the next bitmap,  $X_9$ , would break our ground rules: The population immediately after  $X_8$  grows to 12 in the first and last examples, but shrinks to 5 in the second-from-last; and the path becomes immobile in the other two. Indeed, we have  $X_5 = X_7$  in the second example, hence  $X_6 = X_8$  and  $X_7 = X_9$ , etc. Such a repeating pattern is called an *oscillator* of period 2. The third example ends with an oscillator of period 1, known as a “still life.”

What are the ultimate destinations of these paths? The first one becomes still, with  $X_{69} = X_{70}$ ; and the fourth becomes *very* still, with  $X_{12} = 0$ ! The fifth is the most fascinating of the group, because it continues to produce ever more elaborate valentine shapes, then proceeds to dance and sparkle, until finally beginning to twinkle with period 2 starting at time 177. Thus its members  $X_2$  through  $X_7$  qualify as “Methuselahs,” defined by Martin Gardner as “Life patterns of population less than 10 that do not become stable within 50 generations.” (A predictable pattern, like the glider or an oscillator, is called *stable*.)

SAT solvers are basically useless for the study of Methuselahs, because the state space becomes too large. But they are quite helpful when we want to illuminate many other aspects of Life, and exercises 66–85 discuss some notable instances. We will consider one more instructive example before moving on,

Guy  
glider  
population  
Knuth  
symmetry  
oscillator  
cyclic patterns  
still life  
Methuselahs  
Gardner  
stable



namely an application to “eaters.” Consider a Life path of the form

$$X_0 = \begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix} \rightarrow \begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix} \rightarrow \begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix} \rightarrow \begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix} \rightarrow \begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix} \rightarrow \begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix} = X_5, \quad (38)$$

where the gray cells form a still life and the cells of  $X_1, X_2, X_3$  are unknown. Thus  $X_4 = X_5$  and  $X_0 = X_5 + \text{glider}$ . Furthermore we require that the still life  $X_5$  does not interact with the glider’s parent,  $\blacksquare$ ; see exercise 77. The idea is that a glider will be gobbled up if it happens to glide into this particular still life, and the still life will rapidly reconstitute itself as if nothing had happened.

Algorithm C almost instantaneously (well, after about 100 megamems) finds

$$\begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix} \rightarrow \begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix} \rightarrow \begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix} \rightarrow \begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix} \rightarrow \begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix}, \quad (39)$$

the four-step eater first observed in action by R. W. Gosper in 1971.

**Applications to mutual exclusion.** Let’s look now at how bounded model checking can help us to prove that algorithms are correct. (Or incorrect.) Some of the most challenging issues of verification arise when we consider parallel processes that need to synchronize their concurrent behavior. To simplify our discussion it will be convenient to tell a little story about Alice and Bob.

Alice and Bob are casual friends who share an apartment. One of their joint rooms is special: When they’re in that critical room, which has two doors, they don’t want the other person to be present. Furthermore, being busy people, they don’t want to interrupt each other needlessly. So they agree to control access to the room by using an indicator light, which can be switched on or off.

The first protocol they tried can be characterized by symmetrical algorithms:

- |                                      |                                      |      |
|--------------------------------------|--------------------------------------|------|
| A0. Maybe go to A1.                  | B0. Maybe go to B1.                  |      |
| A1. If $l$ go to A1, else to A2.     | B1. If $l$ go to B1, else to B2.     |      |
| A2. Set $l \leftarrow 1$ , go to A3. | B2. Set $l \leftarrow 1$ , go to B3. | (40) |
| A3. Critical, go to A4.              | B3. Critical, go to B4.              |      |
| A4. Set $l \leftarrow 0$ , go to A0. | B4. Set $l \leftarrow 0$ , go to B0. |      |

At any instant of time, Alice is in one of five states,  $\{A0, A1, A2, A3, A4\}$ , and the rules of her program show how that state might change. In state A0 she isn’t interested in the critical room; but she goes to A1 when she does wish to use it. She reaches that objective in state A3. Similar remarks apply to Bob. When the indicator light is on ( $l = 1$ ), they wait until the other person has exited the room and switched the light back off ( $l = 0$ ).

Alice and Bob don’t necessarily operate at the same speed. But they’re allowed to dawdle only when in the “maybe” state A0 or B0. More precisely, we model the situation by converting every relevant scenario into a discrete sequence of state transitions. At every time  $t = 0, 1, 2, \dots$ , either Alice or Bob (but not both) will perform the command associated with their current state, thereby perhaps changing to a different state at time  $t + 1$ . This choice is nondeterministic.

Only four kinds of primitive commands are permitted in the procedures we shall study, all of which are illustrated in (40): (1) “Maybe go to  $s$ ”; (2) “Critical,

eaters  
Gosper  
parallel processes  
Alice–  
Bob–  
“maybe” state  
nondeterministic

go to  $s$ ”; (3) “Set  $v \leftarrow b$ , go to  $s$ ”; and (4) “If  $v$  go to  $s_1$ , else to  $s_0$ ”. Here  $s$  denotes a state name,  $v$  denotes a shared Boolean variable, and  $b$  is 0 or 1.

Unfortunately, Alice and Bob soon learned that protocol (40) is unreliable: One day she went from A1 to A2 and he went from B1 to B2, before either of them had switched the indicator on. Embarrassment (A3 and B3) followed.

They could have discovered this problem in advance, if they’d converted the state transitions of (40) into clauses for bounded model checking, as in (33), then applied a SAT solver. In this case the vector  $X_t$  that corresponds to time  $t$  consists of Boolean variables that encode each of their current states, as well as the current value of  $l$ . We can, for example, have eleven variables  $A0_t, A1_t, A2_t, A3_t, A4_t, B0_t, B1_t, B2_t, B3_t, B4_t, l_t$ , together with ten binary exclusion clauses  $(\overline{A0_t} \vee \overline{A1_t})$ ,  $(\overline{A0_t} \vee \overline{A2_t})$ ,  $\dots$ ,  $(\overline{A3_t} \vee \overline{A4_t})$  to ensure that Alice is in at most one state, and with ten similar clauses for Bob. There’s also a variable  $@_t$ , which is true or false depending on whether Alice or Bob executes their program step at time  $t$ . (We say that Alice was “bumped” if  $@_t = 1$ , and Bob was bumped if  $@_t = 0$ .)

If we start with the initial state  $X_0$  defined by unit clauses

$$A0_0 \wedge \overline{A1_0} \wedge \overline{A2_0} \wedge \overline{A3_0} \wedge \overline{A4_0} \wedge B0_0 \wedge \overline{B1_0} \wedge \overline{B2_0} \wedge \overline{B3_0} \wedge \overline{B4_0} \wedge \bar{l}_0, \quad (41)$$

the following clauses for  $0 \leq t < r$  (discussed in exercise 87) will emulate the first  $r$  steps of every legitimate scenario defined by (40):

$$\begin{array}{lll} (@_t \vee \overline{A0_t} \vee A0_{t+1}) & (\overline{@_t} \vee \overline{A0_t} \vee A0_{t+1} \vee A1_{t+1}) & (@_t \vee \overline{B0_t} \vee B0_{t+1} \vee B1_{t+1}) \\ (@_t \vee \overline{A1_t} \vee A1_{t+1}) & (\overline{@_t} \vee \overline{A1_t} \vee \bar{l}_t \vee A1_{t+1}) & (@_t \vee \overline{B1_t} \vee \bar{l}_t \vee B1_{t+1}) \\ (@_t \vee \overline{A2_t} \vee A2_{t+1}) & (\overline{@_t} \vee \overline{A1_t} \vee l_t \vee A2_{t+1}) & (@_t \vee \overline{B1_t} \vee l_t \vee B2_{t+1}) \\ (@_t \vee \overline{A3_t} \vee A3_{t+1}) & (\overline{@_t} \vee \overline{A2_t} \vee A3_{t+1}) & (@_t \vee \overline{B2_t} \vee B3_{t+1}) \\ (@_t \vee \overline{A4_t} \vee A4_{t+1}) & (\overline{@_t} \vee \overline{A2_t} \vee l_{t+1}) & (@_t \vee \overline{B2_t} \vee l_{t+1}) \\ (@_t \vee \overline{B0_t} \vee B0_{t+1}) & (\overline{@_t} \vee \overline{A3_t} \vee A4_{t+1}) & (@_t \vee \overline{B3_t} \vee B4_{t+1}) \\ (@_t \vee \overline{B1_t} \vee B1_{t+1}) & (\overline{@_t} \vee \overline{A4_t} \vee A0_{t+1}) & (@_t \vee \overline{B4_t} \vee B0_{t+1}) \\ (@_t \vee \overline{B2_t} \vee B2_{t+1}) & (\overline{@_t} \vee \overline{A4_t} \vee \bar{l}_{t+1}) & (@_t \vee \overline{B4_t} \vee \bar{l}_{t+1}) \\ (@_t \vee \overline{B3_t} \vee B3_{t+1}) & (\overline{@_t} \vee l_t \vee A2_t \vee A4_t \vee \bar{l}_{t+1}) & (@_t \vee l_t \vee B2_t \vee B4_t \vee \bar{l}_{t+1}) \\ (@_t \vee \overline{B4_t} \vee B4_{t+1}) & (\overline{@_t} \vee \bar{l}_t \vee A2_t \vee A4_t \vee l_{t+1}) & (@_t \vee \bar{l}_t \vee B2_t \vee B4_t \vee l_{t+1}) \end{array} \quad (42)$$

If we now add the unit clauses  $(A3_r)$  and  $(B3_r)$ , the resulting set of  $13 + 50r$  clauses in  $11 + 12r$  variables is readily satisfiable when  $r = 6$ , thereby proving that the critical room might indeed be jointly occupied. (Incidentally, standard terminology for mutual exclusion protocols would say that “two threads concurrently execute a *critical section*”; but we shall continue with our roommate metaphor.)

Back at the drawing board, one idea is to modify (40) by letting Alice use the room only when  $l = 1$ , but letting Bob in when  $l = 0$ :

$$\begin{array}{ll} A0. \text{ Maybe go to A1.} & B0. \text{ Maybe go to B1.} \\ A1. \text{ If } l \text{ go to A2, else to A1.} & B1. \text{ If } l \text{ go to B1, else to B2.} \\ A2. \text{ Critical, go to A3.} & B2. \text{ Critical, go to B3.} \\ A3. \text{ Set } l \leftarrow 0, \text{ go to A0.} & B3. \text{ Set } l \leftarrow 1, \text{ go to B0.} \end{array} \quad (43)$$

Computer tests with  $r = 100$  show that the corresponding clauses are unsatisfiable; thus mutual exclusion is apparently guaranteed by (43).

But (43) is a nonstarter, because it imposes an intolerable cost: Alice can't use the room  $k$  times until Bob has already done so! Scrap that.

How about installing another light, so that each person controls one of them?

|                                      |                                      |      |
|--------------------------------------|--------------------------------------|------|
| A0. Maybe go to A1.                  | B0. Maybe go to B1.                  |      |
| A1. If $b$ go to A1, else to A2.     | B1. If $a$ go to B1, else to B2.     |      |
| A2. Set $a \leftarrow 1$ , go to A3. | B2. Set $b \leftarrow 1$ , go to B3. | (44) |
| A3. Critical, go to A4.              | B3. Critical, go to B4.              |      |
| A4. Set $a \leftarrow 0$ , go to A0. | B4. Set $b \leftarrow 0$ , go to B0. |      |

deadlock  
reboot  
Dijkstra  
starvation

No; this suffers from the same defect as (40). But maybe we can cleverly switch the order of steps 1 and 2:

|                                      |                                      |      |
|--------------------------------------|--------------------------------------|------|
| A0. Maybe go to A1.                  | B0. Maybe go to B1.                  |      |
| A1. Set $a \leftarrow 1$ , go to A2. | B1. Set $b \leftarrow 1$ , go to B2. |      |
| A2. If $b$ go to A2, else to A3.     | B2. If $a$ go to B2, else to B3.     | (45) |
| A3. Critical, go to A4.              | B3. Critical, go to B4.              |      |
| A4. Set $a \leftarrow 0$ , go to A0. | B4. Set $b \leftarrow 0$ , go to B0. |      |

Yes! Exercise 95 proves easily that this protocol does achieve mutual exclusion.

Alas, however, a new problem now arises, namely the problem known as “deadlock” or “livelock.” Alice and Bob can get into states A2 and B2, after which they're stuck — each waiting for the other to go critical.

In such cases they could agree to “reboot” somehow. But that would be a cop-out; they really seek a better solution. And they aren't alone: Many people have struggled with this surprisingly delicate problem over the years, and several solutions (both good and bad) appear in the exercises below. Edsger Dijkstra, in some pioneering lecture notes entitled *Cooperating Sequential Processes* [Technological University Eindhoven (September 1965), §2.1], thought of an instructive way to improve on (45):

|                                      |                                      |      |
|--------------------------------------|--------------------------------------|------|
| A0. Maybe go to A1.                  | B0. Maybe go to B1.                  |      |
| A1. Set $a \leftarrow 1$ , go to A2. | B1. Set $b \leftarrow 1$ , go to B2. |      |
| A2. If $b$ go to A3, else to A4.     | B2. If $a$ go to B3, else to B4.     | (46) |
| A3. Set $a \leftarrow 0$ , go to A1. | B3. Set $b \leftarrow 0$ , go to B1. |      |
| A4. Critical, go to A5.              | B4. Critical, go to B5.              |      |
| A5. Set $a \leftarrow 0$ , go to A0. | B5. Set $b \leftarrow 0$ , go to B0. |      |

But he realized that this too is unsatisfactory, because it permits scenarios in which Alice, say, might wait forever while Bob repeatedly uses the critical room. (Indeed, if Alice and Bob are in states A1 and B2, she might go to A2, A3, then A1, thereby letting him run to B4, B5, B0, B1, and B2; they're back where they started, yet she's made no progress.)

The existence of this problem, called *starvation*, can also be detected via bounded model checking. The basic idea (see exercise 91) is that starvation occurs if and only if there is a loop of transitions

$$X_0 \rightarrow X_1 \rightarrow \cdots \rightarrow X_p \rightarrow X_{p+1} \rightarrow \cdots \rightarrow X_r = X_p \quad (47)$$

such that (i) Alice and Bob each are bumped at least once during the loop; and (ii) at least one of them is never in a “maybe” or “critical” state during the loop.

And those conditions are easily encoded into clauses, because we can identify the variables for time  $r$  with the variables for time  $p$ , and we can append the clauses

$$(\overline{a_p} \vee \overline{a_{p+1}} \vee \cdots \vee \overline{a_{r-1}}) \wedge (a_p \vee a_{p+1} \vee \cdots \vee a_{r-1}) \quad (48)$$

to guarantee (i). Condition (ii) is simply a matter of appending unit clauses; for example, to test whether Alice can be starved by (46), the relevant clauses are  $\overline{A0_p} \wedge \overline{A0_{p+1}} \wedge \cdots \wedge \overline{A0_{r-1}} \wedge \overline{A4_p} \wedge \overline{A4_{p+1}} \wedge \cdots \wedge \overline{A4_{r-1}}$ .

The deficiencies of (43), (45), and (46) can all be viewed as instances of starvation, because (47) and (48) are satisfiable (see exercise 90). Thus we can use bounded model checking to find counterexamples to *any* unsatisfactory protocol for mutual exclusion, either by exhibiting a scenario in which Alice and Bob are both in the critical room or by exhibiting a feasible starvation cycle (47).

Of course we'd like to go the other way, too: If a protocol has no counterexamples for, say,  $r = 100$ , we still might not know that it is really reliable; a counterexample might exist only when  $r$  is extremely large. Fortunately there are ways to obtain decent upper bounds on  $r$ , so that bounded model checking can be used to prove correctness as well as to demonstrate incorrectness. For example, we can verify the simplest known correct solution to Alice and Bob's problem, a protocol by G. L. Peterson [*Information Proc. Letters* **12** (1981), 115–116], who noticed that a careful combination of (43) and (45) actually suffices:

- |                                      |                                      |      |
|--------------------------------------|--------------------------------------|------|
| A0. Maybe go to A1.                  | B0. Maybe go to B1.                  |      |
| A1. Set $a \leftarrow 1$ , go to A2. | B1. Set $b \leftarrow 1$ , go to B2. |      |
| A2. Set $l \leftarrow 0$ , go to A3. | B2. Set $l \leftarrow 1$ , go to B3. |      |
| A3. If $b$ go to A4, else to A5.     | B3. If $a$ go to B4, else to B5.     | (49) |
| A4. If $l$ go to A5, else to A3.     | B4. If $l$ go to B3, else to B5.     |      |
| A5. Critical, go to A6.              | B5. Critical, go to B6.              |      |
| A6. Set $a \leftarrow 0$ , go to A0. | B6. Set $b \leftarrow 0$ , go to B0. |      |

Now there are *three* signal lights,  $a$ ,  $b$ , and  $l$ —one controlled by Alice, one controlled by Bob, and one switchable by both.

To show that states A5 and B5 can't be concurrent, we can observe that the shortest counterexample will not repeat any state twice; in other words, it will be a *simple* path of transitions (33). Thus we can assume that  $r$  is at most the total number of states. However, (49) has  $7 \times 7 \times 2 \times 2 \times 2 = 392$  states; that's a finite bound, not really out of reach for a good SAT solver on this particular problem, but we can do much better. For example, it's not hard to devise clauses that are satisfiable if and only if there's a simple path of length  $\leq r$  (see exercise 92), and in this particular case the longest simple path turns out to have only 54 steps.

We can in fact do better yet by using the important notion of *invariants*, which we encountered in Section 1.2.1 and have seen repeatedly throughout this series of books. Invariant assertions are the key to most proofs of correctness, so it's not surprising that they also give a significant boost to bounded model checking. Formally speaking, if  $\Phi(X)$  is a Boolean function of the state vector  $X$ , we say that  $\Phi$  is invariant if  $\Phi(X)$  implies  $\Phi(X')$  whenever  $X \rightarrow X'$ . For example,

Peterson  
simple path  
longest simple path  
invariants

it's not hard to see that the following clauses are invariant with respect to (49):

$$\begin{aligned} \Phi(X) = & (A0 \vee A1 \vee A2 \vee A3 \vee A4 \vee A5 \vee A6) \wedge (B0 \vee B1 \vee B2 \vee B3 \vee B4 \vee B5 \vee B6) \\ & \wedge (\overline{A0} \vee \overline{a}) \wedge (\overline{A1} \vee \overline{a}) \wedge (\overline{A2} \vee a) \wedge (\overline{A3} \vee a) \wedge (\overline{A4} \vee a) \wedge (\overline{A5} \vee a) \wedge (\overline{A6} \vee a) \\ & \wedge (\overline{B0} \vee \overline{b}) \wedge (\overline{B1} \vee \overline{b}) \wedge (\overline{B2} \vee b) \wedge (\overline{B3} \vee b) \wedge (\overline{B4} \vee b) \wedge (\overline{B5} \vee b) \wedge (\overline{B6} \vee b). \end{aligned} \quad (50)$$

(The clause  $\overline{A0} \vee \overline{a}$  says that  $a = 0$  when Alice is in state A0, etc.) And we can use a SAT solver to *prove* that  $\Phi$  is invariant, by showing that the clauses

$$\Phi(X) \wedge (X \rightarrow X') \wedge \neg\Phi(X') \quad (51)$$

are *unsatisfiable*. Furthermore  $\Phi(X_0)$  holds for the initial state  $X_0$ , because  $\neg\Phi(X_0)$  is unsatisfiable. (See exercise 93.) Therefore  $\Phi(X_t)$  is true for all  $t \geq 0$ , by induction, and we may add these helpful clauses to all of our formulas.

The invariant (50) reduces the total number of states by a factor of 4. And the real clincher is the fact that the clauses

$$(X_0 \rightarrow X_1 \rightarrow \cdots \rightarrow X_r) \wedge \Phi(X_0) \wedge \Phi(X_1) \wedge \cdots \wedge \Phi(X_r) \wedge A5_r \wedge B5_r, \quad (52)$$

where  $X_0$  is *not* required to be the initial state, turn out to be unsatisfiable when  $r = 3$ . In other words, there's no way to go back more than two steps from a bad state, without violating the invariant. We can conclude that mutual exclusion needs to be verified for (49) only by considering paths of length 2(!). Furthermore, similar ideas (exercise 98) show that (49) is starvation-free.

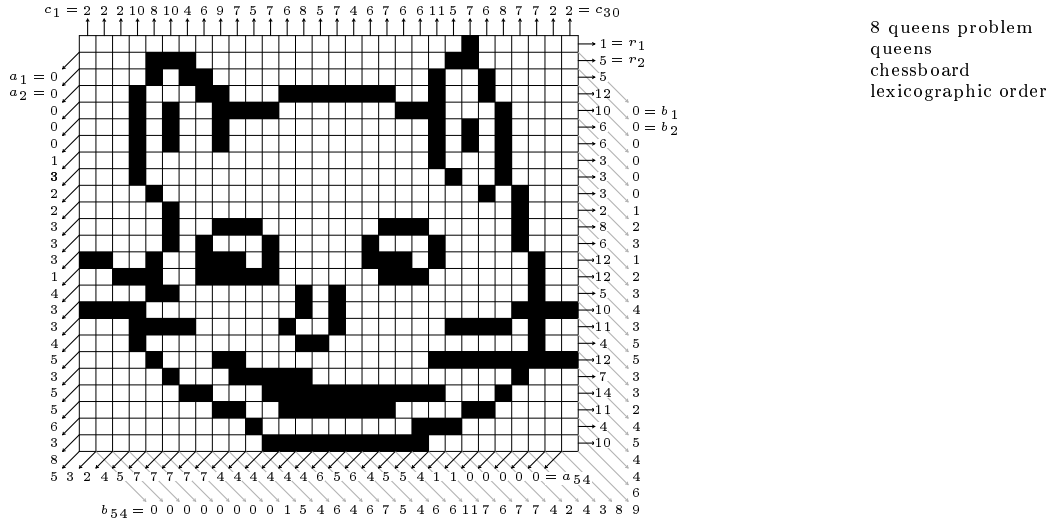
*Caveat:* Although (49) is a correct protocol for mutual exclusion according to Alice and Bob's ground rules, it *cannot* be used safely on most modern computers unless special care is taken to synchronize cache memories and write buffers. The reason is that hardware designers use all sorts of trickery to gain speed, and those tricks might allow one process to see  $a = 0$  at time  $t + 1$  even though another process has set  $a \leftarrow 1$  at time  $t$ . We have developed the algorithms above by assuming a model of parallel computation that Leslie Lamport has called *sequential consistency* [IEEE Trans. C-28 (1979), 690–691].

**Digital tomography.** Another set of appealing questions amenable to SAT solving comes from the study of binary images for which partial information is given. Consider, for example, Fig. 36, which shows the “Cheshire cat” of Section 7.1.3 in a new light. This image is an  $m \times n$  array of Boolean variables  $(x_{i,j})$ , with  $m = 25$  rows and  $n = 30$  columns: The upper left corner element,  $x_{1,1}$ , is 0, representing white; and  $x_{1,24} = 1$  corresponds to the lone black pixel in the top row. We are given the row sums  $r_i = \sum_{j=1}^n x_{i,j}$  for  $1 \leq i \leq m$  and the column sums  $c_j = \sum_{i=1}^m x_{i,j}$  for  $1 \leq j \leq n$ , as well as both sets of sums in the 45° diagonal directions, namely

$$a_d = \sum_{i+j=d+1} x_{i,j} \quad \text{and} \quad b_d = \sum_{i-j=d-n} x_{i,j} \quad \text{for } 0 < d < m+n. \quad (53)$$

To what extent can such an image be reconstructed from its sums  $r_i$ ,  $c_j$ ,  $a_d$ , and  $b_d$ ? Small examples are often uniquely determined by these Xray-like projections (see exercise 103). But the discrete nature of pixel images makes the reconstruction problem considerably more difficult than the corresponding

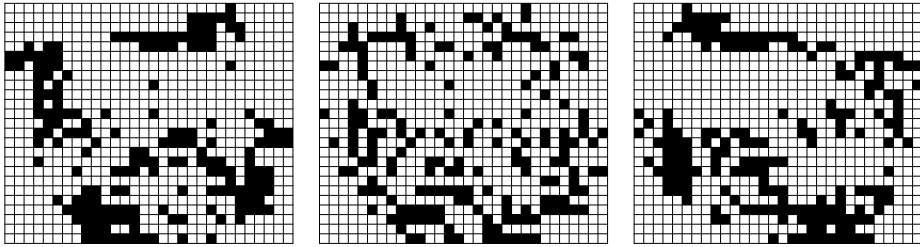
initial state  
induction  
cache memories  
write buffers  
parallel computation  
Lamport  
sequential consistency  
tomcat  
Cheshire cat  
diagonal  
Xray  
pixel images



**Fig. 36.** An array of black and white pixels together with its row sums  $r_i$ , column sums  $c_j$ , and diagonal sums  $a_d, b_d$ .

continuous problem, in which projections from many different angles are available. Notice, for example, that the classical “8 queens problem” —to place eight nonattacking queens on a chessboard —is equivalent to solving an  $8 \times 8$  digital tomography problem with the constraints  $r_i = 1, c_j = 1, a_d \leq 1,$  and  $b_d \leq 1$ .

The constraints of Fig. 36 appear to be quite strict, so we might expect that most of the pixels  $x_{i,j}$  are determined uniquely by the given sums. For instance, the fact that  $a_1 = \dots = a_5 = 0$  tells us that  $x_{i,j} = 0$  whenever  $i + j \leq 6$ ; and similar deductions are possible at all four corners of the image. A crude “ballpark estimate” suggests that we’re given a few more than 150 sums, most of which occupy 5 bits each; hence we have roughly  $150 \times 5 = 750$  bits of data, from which we wish to reconstruct  $25 \times 30 = 750$  pixels  $x_{i,j}$ . Actually, however, this problem turns out to have many billions of solutions (see Fig. 37), most of which aren’t catlike! Exercise 106 provides a less crude estimate, which shows that this abundance of solutions isn’t really surprising.



(a) lexicographically first; (b) maximally different; (c) lexicographically last.

**Fig. 37.** Extreme solutions to the constraints of Fig. 36.

A digital tomography problem such as Fig. 36 is readily represented as a sequence of clauses to be satisfied, because each of the individual requirements is just a special case of the cardinality constraints that we've already considered in the clauses of (18)–(21). This problem differs from the other instances of SAT that we've been discussing, primarily because it consists *entirely* of cardinality constraints: It is a question of solving  $25 + 30 + 54 + 54 = 163$  simultaneous linear equations in 750 variables  $x_{i,j}$ , where each variable must be either 0 or 1. So it's essentially an instance of *integer programming* (IP), not an instance of satisfiability (SAT). On the other hand, Bailleux and Boufkhad devised clauses (20) and (21) precisely because they wanted to apply SAT solvers, not IP solvers, to digital tomography. In the case of Fig. 36, their method yields approximately 40,000 clauses in 9,000 variables, containing about 100,000 literals altogether.

Figure 37(b) illustrates a solution that differs as much as possible from Fig. 36. Thus it minimizes the sum  $x_{1,24} + x_{2,5} + x_{2,6} + \cdots + x_{25,21}$  of the 182 variables that correspond to black pixels, over all 0-or-1-valued solutions to the linear equations. If we use linear programming to minimize that sum over  $0 \leq x_{i,j} \leq 1$ , *without* requiring the variables to be integers, we find almost instantly that the minimum value is  $\approx 31.38$  under these relaxed conditions; hence every black-and-white image must have at least 32 black pixels in common with Fig. 36. Furthermore, Fig. 37(b) — which can be computed in a few seconds by widely available IP solvers such as CPLEX — actually achieves this minimum. By contrast, state-of-the-art SAT solvers as of 2013 had great difficulty finding such an image, even when told that a 32-in-common solution is possible.

Parts (a) and (c) of Fig. 37 are, similarly, quite relevant to the current state of the SAT-solving art: They represent hundreds of individual SAT instances, where the first  $k$  variables are set to particular known values and we try to find a solution with the next variable either 0 or 1, respectively. Several of the subproblems that arose while computing rows 6 and 7 of Fig. 37(c) turned out to be quite challenging, although resolvable in a few hours; and similar problems, which correspond to different kinds of lexicographic order, apparently still lie beyond the reach of contemporary SAT-oriented methods. Yet IP solvers polish these problems off with ease. (See exercises 109 and 111.)

If we provide more information about an image, our chances of being able to reconstruct it uniquely are naturally enhanced. For example, suppose we also compute the numbers  $r'_i$ ,  $c'_j$ ,  $a'_d$ , and  $b'_d$ , which count the *runs of 1s* that occur in each row, column, and diagonal. (We have  $r'_1 = 1$ ,  $r'_2 = 2$ ,  $r'_3 = 4$ , and so on.) Given this additional data, we can show that Fig. 36 is the only solution, because a suitable set of clauses turns out to be unsatisfiable. Exercise 117 explains one way by which (20) and (21) can be modified so that they provide constraints based on the run counts. Furthermore, it isn't difficult to express even more detailed constraints, such as the assertion that “column 4 contains runs of respective lengths (6, 1, 3),” as a sequence of clauses; see exercise 438.

**SAT examples — summary.** We've now seen convincing evidence that simple Boolean clauses — ANDs of ORs of literals — are enormously versatile. Among

cardinality constraints  
linear equations  
integer programming  
Bailleux  
Boufkhad  
linear programming relaxation  
CPLEX  
IP: Integer programming  
lexicographic order  
runs of 1s

other things, we've used them to encode problems of graph coloring, integer factorization, hardware fault testing, machine learning, model checking, and tomography. And indeed, Section 7.9 will demonstrate that 3SAT is the “poster child” for NP-complete problems in general: *Any* problem in NP—which is a huge class, essentially comprising all yes-or-no questions of size  $N$  whose affirmative answers are verifiable in  $N^{O(1)}$  steps—can be formulated as an equivalent instance of 3SAT, without greatly increasing the problem size.

**Backtracking for SAT.** We've now seen a dizzying variety of intriguing and important examples of SAT that are begging to be solved. How shall we solve them?

Any instance of SAT that involves at least one variable can be solved systematically by choosing a variable and setting it to 0 or 1. Either of those choices gives us a smaller instance of SAT; so we can continue until reaching either an empty instance—which is trivially satisfiable, because no clauses need to be satisfied—or an instance that contains an empty clause. In the latter case we must back up and reconsider one of our earlier choices, proceeding in the same fashion until we either succeed or exhaust all the possibilities.

For example, consider again the formula  $F$  in (1). If we set  $x_1 = 0$ ,  $F$  reduces to  $\bar{x}_2 \wedge (x_2 \vee x_3)$ , because the first clause  $(x_1 \vee \bar{x}_2)$  loses its  $x_1$ , while the last two clauses contain  $\bar{x}_1$  and are satisfied. It will be convenient to have a notation for this reduced problem; so let's write

$$F|\bar{x}_1 = \bar{x}_2 \wedge (x_2 \vee x_3). \quad (54)$$

Similarly, if we set  $x_1 = 1$ , we obtain the reduced problem

$$F|x_1 = (x_2 \vee x_3) \wedge \bar{x}_3 \wedge (\bar{x}_2 \vee x_3). \quad (55)$$

$F$  is satisfiable if and only if we can satisfy either (54) or (55).

In general if  $F$  is any set of clauses and if  $l$  is any literal, then  $F|l$  (read “ $F$  given  $l$ ” or “ $F$  conditioned on  $l$ ”) is the set of clauses obtained from  $F$  by

- removing every clause that contains  $l$ ; and
- removing  $\bar{l}$  from every clause that contains  $\bar{l}$ .

This conditioning operation is commutative, in the sense that  $F|l|l' = F|l'|l$  when  $l' \neq \bar{l}$ . If  $L = \{l_1, \dots, l_k\}$  is any set of strictly distinct literals, we can also write  $F|L = F|l_1|\dots|l_k$ . In these terms,  $F$  is satisfiable if and only if  $F|L = \emptyset$  for some such  $L$ , because the literals of  $L$  satisfy every clause of  $F$  when  $F|L = \emptyset$ .

The systematic strategy for SAT that was sketched above can therefore be formulated as the following recursive procedure  $B(F)$ , which returns the special value  $\perp$  when  $F$  is unsatisfiable, otherwise it returns a set  $L$  that satisfies  $F$ :

$$B(F) = \begin{cases} \text{If } F = \emptyset, \text{ return } \emptyset. \text{ (} F \text{ is trivially satisfiable.)} \\ \text{Otherwise if } \epsilon \in F, \text{ return } \perp. \text{ (} F \text{ is unsatisfiable.)} \\ \text{Otherwise let } l \text{ be a literal in } F \text{ and set } L \leftarrow B(F|l). \\ \text{If } L \neq \perp, \text{ return } L \cup l. \text{ Otherwise set } L \leftarrow B(F|\bar{l}). \\ \text{If } L \neq \perp, \text{ return } L \cup \bar{l}. \text{ Otherwise return } \perp. \end{cases} \quad (56)$$

Let's try to flesh out this abstract algorithm by converting it to efficient code at a lower level. From our previous experience with backtracking, we know

backtracking-  
notation  $F|l$   
reduction of clauses  
given literals  
conditioning operation  
commutative  
notation  $F|L$   
recursive procedure



that it will be crucial to have data structures that allow us to go quickly from  $F$  to  $F | l$ , then back again to  $F$  if necessary, when  $F$  is a set of clauses and  $l$  is a literal. In particular, we'll want a good way to find all of the clauses that contain a given literal.

data structures—  
exact cover  
cells  
doubly linked  
head of the list

A combination of sequential and linked structures suggests itself for this purpose, based on our experience with exact cover problems: We can represent each clause as a set of *cells*, where each cell  $p$  contains a literal  $l = L(p)$  together with pointers  $F(p)$  and  $B(p)$  to other cells that contain  $l$ , in a doubly linked list. We'll also need  $C(p)$ , the number of the clause to which  $p$  belongs. The cells of clause  $C_i$  will be in consecutive locations  $START(i) + j$ , for  $0 \leq j < SIZE(i)$ .

We will find it convenient to represent the literals  $x_k$  and  $\bar{x}_k$ , which involve variable  $x_k$ , by using the integers  $2k$  and  $2k + 1$ . With this convention we have

$$\bar{l} = l \oplus 1 \quad \text{and} \quad |l| = x_{l \gg 1}. \quad (57)$$

Our implementation of (56) will assume that the variables are  $x_1, x_2, \dots, x_n$ ; thus the  $2n$  possible literals will be in the range  $2 \leq l \leq 2n + 1$ .

Cells 0 through  $2n + 1$  are reserved for special purposes: Cell  $l$  is the head of the list for the occurrences of  $l$  in other cells. Furthermore  $C(l)$  will be the length of that list, namely the number of currently active clauses in which  $l$  appears.

For example, the  $m = 7$  ternary clauses  $R'$  of (7) might be represented internally in  $2n + 2 + 3m = 31$  cells as follows, using these conventions:

```

p = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
L(p) = - - - - - - - - - 9 7 3 8 7 5 6 5 3 8 4 3 8 6 2 9 6 4 7 4 2
F(p) = - - 30 21 29 17 26 28 22 25 9 7 3 8 11 5 6 15 12 13 4 18 19 16 2 10 23 20 14 27 24
B(p) = - - 24 12 20 15 16 11 13 10 25 14 18 19 28 17 23 5 21 22 27 3 8 26 30 9 6 29 7 4 2
C(p) = - - 2 3 3 2 3 3 3 2 7 7 7 6 6 6 5 5 5 4 4 4 4 3 3 3 2 2 2 1 1 1

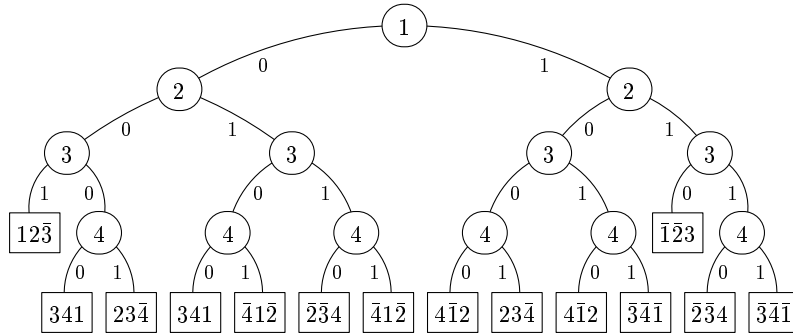
```

The literals of each clause appear in decreasing order here; for example, the literals  $L(p) = (8, 4, 3)$  in cells 19 through 21 represent the clause  $x_4 \vee x_2 \vee \bar{x}_1$ , which appears as the fourth clause, '4 $\bar{1}$ 2' in (7). This ordering turns out to be quite useful, because we'll always choose the smallest unset variable as the  $l$  or  $\bar{l}$  in (56); then  $l$  or  $\bar{l}$  will always appear at the right of its clauses, and we can remove it or put it back by simply changing the relevant  $SIZE$  fields.

The clauses in this example have  $START(i) = 31 - 3i$  for  $1 \leq i \leq 7$ , and  $SIZE(i) = 3$  when computation begins.

**Algorithm A** (*Satisfiability by backtracking*). Given nonempty clauses  $C_1 \wedge \dots \wedge C_m$  on  $n > 0$  Boolean variables  $x_1 \dots x_n$ , represented as above, this algorithm finds a solution if and only if the clauses are satisfiable. It records its current progress in an array  $m_1 \dots m_n$  of "moves," whose significance is explained below.

- A1.** [Initialize.] Set  $a \leftarrow m$  and  $d \leftarrow 1$ . (Here  $a$  represents the number of active clauses, and  $d$  represents the depth-plus-one in an implicit search tree.)
- A2.** [Choose.] Set  $l \leftarrow 2d$ . If  $C(l) \leq C(l + 1)$ , set  $l \leftarrow l + 1$ . Then set  $m_d \leftarrow (l \& 1) + 4[C(l \oplus 1) = 0]$ . (See below.) Terminate successfully if  $C(l) = a$ .
- A3.** [Remove  $\bar{l}$ .] Delete  $\bar{l}$  from all active clauses; but go to A5 if that would make a clause empty. (We want to ignore  $\bar{l}$ , because we're making  $l$  true.)



move codes  
pure literals

**Fig. 38.** The search tree that is implicitly traversed by Algorithm A, when that algorithm is applied to the eight unsatisfiable clauses  $R$  defined in (6). Branch nodes are labeled with the variable being tested; leaf nodes are labeled with a clause that is found to be contradicted.

- A4.** [Deactivate  $l$ 's clauses.] Suppress all clauses that contain  $l$ . (Those clauses are now satisfied.) Then set  $a \leftarrow a - C(l)$ ,  $d \leftarrow d + 1$ , and return to A2.
- A5.** [Try again.] If  $m_d < 2$ , set  $m_d \leftarrow 3 - m_d$ ,  $l \leftarrow 2d + (m_d \& 1)$ , and go to A3.
- A6.** [Backtrack.] Terminate unsuccessfully if  $d = 1$  (the clauses are unsatisfiable). Otherwise set  $d \leftarrow d - 1$  and  $l \leftarrow 2d + (m_d \& 1)$ .
- A7.** [Reactivate  $l$ 's clauses.] Set  $a \leftarrow a + C(l)$ , and unsuppress all clauses that contain  $l$ . (Those clauses are now unsatisfied, because  $l$  is no longer true.)
- A8.** [Unremove  $\bar{l}$ .] Reinstat  $\bar{l}$  in all the active clauses that contain it. Then go back to A5. ■

(See exercise 121 for details of the low-level list processing operations that are needed to update the data structures in steps A3 and A4, and to downdate them in A7 and A8.)

The move codes  $m_j$  of Algorithm A are integers between 0 and 5 that encode the state of the algorithm's progress as follows:

- $m_j = 0$  means we're trying  $x_j = 1$  and haven't yet tried  $x_j = 0$ .
- $m_j = 1$  means we're trying  $x_j = 0$  and haven't yet tried  $x_j = 1$ .
- $m_j = 2$  means we're trying  $x_j = 1$  after  $x_j = 0$  has failed.
- $m_j = 3$  means we're trying  $x_j = 0$  after  $x_j = 1$  has failed.
- $m_j = 4$  means we're trying  $x_j = 1$  when  $\bar{x}_j$  doesn't appear.
- $m_j = 5$  means we're trying  $x_j = 0$  when  $x_j$  doesn't appear.

Codes 4 and 5 refer to so-called "pure literals": If no clause contains the literal  $\bar{l}$ , we can't go wrong by assuming that  $l$  is true.

For example, when Algorithm A is presented with the clauses (7), it cruises directly to a solution by setting  $m_1 m_2 m_3 m_4 = 1014$ ; the solution is  $x_1 x_2 x_3 x_4 = 0101$ . But when the unsatisfiable clauses (6) are given, the successive code strings  $m_1 \dots m_d$  in step A2 are

$$1, 11, 110, 1131, 121, 1211, 1221, 21, 211, 2111, 2121, 221, 2221, \quad (58)$$

before the algorithm gives up. (See Fig. 38.)

It's helpful to display the current string  $m_1 \dots m_d$  now and then, as a convenient indication of progress; this string increases lexicographically. Indeed, fascinating patterns appear as the 2s and 3s gradually move to the left. (Try it!)

When the algorithm terminates successfully in step A2, a satisfying assignment can be read off from the move table by setting  $x_j \leftarrow 1 \oplus (m_j \& 1)$  for  $1 \leq j \leq d$ . Algorithm A stops after finding a single solution; see exercise 122 if you want them all.

lexicographically  
lazy data structures—  
Brown  
Purdom  
watched literals  
partial assignment  
consistent  
unit  
backtracking  
eager

**Lazy data structures.** Instead of using the elaborate doubly linked machinery that underlies Algorithm A, we can actually get by with a much simpler scheme discovered by Cynthia A. Brown and Paul W. Purdom, Jr. [*IEEE Trans. PAMI-4* (1982), 309–316], who introduced the notion of *watched literals*. They observed that we don't really need to know all of the clauses that contain a given literal, because only one literal per clause is actually relevant at any particular time.

Here's the idea: When we work on clauses  $F|L$ , the variables that occur in  $L$  have known values, but the other variables do not. For example, in Algorithm A, variable  $x_j$  is implicitly known to be either true or false when  $j \leq d$ , but its value is unknown when  $j > d$ . Such a situation is called a *partial assignment*. A partial assignment is *consistent* with a set of clauses if no clause consists entirely of false literals. Algorithms for SAT usually deal exclusively with consistent partial assignments; the goal is to convert them to consistent *total* assignments, by gradually eliminating the unknown values.

Thus every clause in a consistent partial assignment has at least one nonfalse literal; and we can assume that such a literal appears first, when the clause is represented in memory. Many nonfalse literals might be present, but only one of them is designated as the clause's "watchee." When a watched literal becomes false, we can find another nonfalse literal to swap into its place—unless the clause has been reduced to a unit, a clause of size 1.

With such a scheme we need only maintain a relatively short list for every literal  $l$ , namely a list  $W_l$  of all clauses that currently watch  $l$ . This list can be singly linked. Hence we need only one link per clause; and we have a total of only  $2n + m$  links altogether, instead of the two links for each *cell* that are required by Algorithm A.

Furthermore—and this is the best part!—*no updates need to be made to the watch lists when backtracking*. The backtrack operations never falsify a nonfalse literal, because they only change values from known to unknown. Perhaps for this reason, data structures based on watched literals are called *lazy*, in contrast with the "eager" data structures of Algorithm A.

Let us therefore redesign Algorithm A and make it more laid-back. Our new data structure for each cell  $p$  has only one field,  $L(p)$ ; the other fields  $F(p)$ ,  $B(p)$ ,  $C(p)$  are no longer necessary, nor do we need  $2n + 2$  special cells. As before we will represent clauses sequentially, with the literals of  $C_j$  beginning at  $\text{START}(j)$  for  $1 \leq j \leq m$ . The watched literal will be the one in  $\text{START}(j)$ ; and a new field,  $\text{LINK}(j)$ , will be the number of another clause with the same watched literal (or 0, if  $C_j$  is the last such clause). Moreover, our new algorithm won't

need  $\text{SIZE}(j)$ . Instead, we can assume that the final literal of  $C_j$  is in location  $\text{START}(j-1) - 1$ , provided that we define  $\text{START}(0)$  appropriately.

The resulting procedure is almost unbelievably short and sweet. It's surely the simplest SAT solver that can claim to be efficient on problems of modest size:

**Algorithm B** (*Satisfiability by watching*). Given nonempty clauses  $C_1 \wedge \dots \wedge C_m$  on  $n > 0$  Boolean variables  $x_1 \dots x_n$ , represented as above, this algorithm finds a solution if and only if the clauses are satisfiable. It records its current progress in an array  $m_1 \dots m_n$  of "moves," whose significance was explained above.

**B1.** [Initialize.] Set  $d \leftarrow 1$ .

**B2.** [Rejoice or choose.] If  $d > n$ , terminate successfully. Otherwise set  $m_d \leftarrow [W_{2d} = 0 \text{ or } W_{2d+1} \neq 0]$  and  $l \leftarrow 2d + m_d$ .

**B3.** [Remove  $\bar{l}$  if possible.] For all  $j$  such that  $\bar{l}$  is watched in  $C_j$ , watch another literal of  $C_j$ . But go to **B5** if that can't be done. (See exercise 124.)

**B4.** [Advance.] Set  $W_{\bar{l}} \leftarrow 0$ ,  $d \leftarrow d + 1$ , and return to **B2**.

**B5.** [Try again.] If  $m_d < 2$ , set  $m_d \leftarrow 3 - m_d$ ,  $l \leftarrow 2d + (m_d \& 1)$ , and go to **B3**.

**B6.** [Backtrack.] Terminate unsuccessfully if  $d = 1$  (the clauses are unsatisfiable). Otherwise set  $d \leftarrow d - 1$  and go back to **B5**. ■

Readers are strongly encouraged to work exercise 124, which spells out the low-level operations that are needed in step **B3**. Those operations accomplish essentially everything that Algorithm B needs to do.

This algorithm doesn't use move codes 4 or 5, because lazy data structures don't have enough information to identify pure literals. Fortunately pure literals are comparatively unimportant in practice; problems that are helped by the pure literal shortcut can usually also be solved quickly without it.

Notice that steps **A2** and **B2** use different criteria for deciding whether to try  $x_d = 1$  or  $x_d = 0$  first at each branch of the search tree. Algorithm A chooses the alternative that satisfies the most clauses; Algorithm B chooses to make  $l$  true instead of  $\bar{l}$  if the watch list for  $\bar{l}$  is empty but the watch list for  $l$  is not. (All clauses in which  $\bar{l}$  is watched will have to change, but those containing  $l$  are satisfied and in good shape.) In case of a tie, both algorithms set  $m_d \leftarrow 1$ , which corresponds to  $x_d = 0$ . The reason is that human-designed instances of SAT tend to have solutions made up of mostly false literals.

**Forced moves from unit clauses.** The simple logic of Algorithm B works well on many problems that aren't too large. But its insistence on setting  $x_1$  first, then  $x_2$ , etc., makes it quite inefficient on many other problems, because it fails to take advantage of unit clauses. A unit clause ( $l$ ) forces  $l$  to be true; therefore two-way branching is unnecessary whenever a unit clause is present. Furthermore, unit clauses aren't rare: Far from it. Experience shows that they're almost ubiquitous in practice, so that the actual search trees often involve only dozens of branch nodes instead of thousands or millions.

The importance of unit clauses was recognized already in the first computer implementation of a SAT solver, designed by Martin Davis, George Logemann,

move codes  
pure literals  
false literals preferred  
unit clauses  
Davis  
Logemann

and Donald Loveland [CACM 5 (1962), 394–397] and based on ideas that Davis had developed earlier with Hilary Putnam [JACM 7 (1960), 201–215]. They extended Algorithm A by introducing mechanisms that recognize when the size of a clause decreases to 1, or when the number of unsatisfied clauses containing a literal drops to 0. In such cases, they put variables onto a “ready list,” and assigned those variables to fixed values before doing any further two-way branching. The resulting program was fairly complex; indeed, computer memory was so limited in those days, they implemented branching by writing all the data for the current node of the search tree onto magnetic tape, then backtracking when necessary by restoring the data from the most recently written tape records! The names of these four authors are now enshrined in the term “DPLL algorithm,” which refers generally to SAT solving via partial assignment and backtracking.

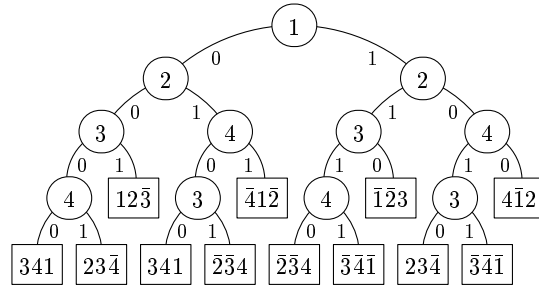
Loveland  
 Davis  
 Putnam  
 pure literals  
 ready list  
 search tree  
 magnetic tape  
 backtracking  
 tape records  
 DPLL algorithm  
 Brown  
 Purdom  
 circular list  
 active ring  
 units  
 waerden

Brown and Purdom, in the paper cited earlier, showed that unit clauses can be detected more simply by using watched literals as in Algorithm B. We can supplement the data structures of that algorithm by introducing indices  $h_1 \dots h_n$  so that the variable whose value is being set at depth  $d$  is  $x_{h_d}$  instead of  $x_d$ . Furthermore we can arrange the not-yet-set variables whose watch lists aren’t empty into a circular list called the “active ring”; the idea is to proceed through the active ring, checking to see whether any of its variables are currently in a unit clause. We resort to two-way branching only if we go all around the ring without finding any such units.

For example, let’s consider the 32 unsatisfiable clauses of *waerden*(3, 3; 9) in (g). The active ring is initially (1 2 3 4 5 6 7), because 8,  $\bar{8}$ , 9, and  $\bar{9}$  aren’t being watched anywhere. There are no unit clauses yet. The algorithm below will decide to try  $\bar{1}$  first; then it will change the clauses 123, 135, 147, and 159 to 213, 315, 417, and 519, respectively, so that nobody watches the false literal 1. The active ring becomes (2 3 4 5 6 7) and the next choice is  $\bar{2}$ ; so 213, 234, 246, and 258 morph respectively into 312, 324, 426, 528. Now, with active ring (3 4 5 6 7), the unit clause ‘3’ is detected (because 1 and 2 are false in ‘312’). This precipitates further changes, and the first steps of the computation can be summarized thus:

| Active ring     | $x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9$ | Units        | Choice    | Changed clauses   |      |
|-----------------|---------------------------------------|--------------|-----------|---|------|
| (1 2 3 4 5 6 7) | - - - - - - - -                       |              | $\bar{1}$ | 213, 315, 417, 519  |      |
| (2 3 4 5 6 7)   | 0 - - - - - - -                       |              | $\bar{2}$ | 312, 324, 426, 528  |      |
| (3 4 5 6 7)     | 0 0 - - - - - -                       | 3            | 3         | $\bar{4}\bar{3}\bar{5}, \bar{5}\bar{3}\bar{4}, \bar{6}\bar{3}\bar{9}$ |      |
| (4 5 6 7)       | 0 0 1 - - - - -                       |              | $\bar{4}$ | 624, 714, 546, 648  |      |
| (5 6 7)         | 0 0 1 0 - - - -                       | 6            | 6         | $\bar{9}\bar{3}\bar{6}, \bar{7}\bar{6}\bar{8}$                        |      |
| (9 7 5)         | 0 0 1 0 - 1 - -                       | $\bar{9}$    | $\bar{9}$ |   | (59) |
| (7 5)           | 0 0 1 0 - 1 - - 0                     | 7            | 7         | $\bar{8}\bar{6}\bar{7}, \bar{8}\bar{7}\bar{9}$                        |      |
| (8 5)           | 0 0 1 0 - 1 1 - 0                     | $\bar{8}$    | $\bar{8}$ |   |      |
| (5)             | 0 0 1 0 - 1 1 0 0                     | $5, \bar{5}$ | Backtrack |   |      |
| (6 9 7 8 5)     | 0 0 1 - - - - -                       |              | 4         | $\bar{5}\bar{3}\bar{4}, \bar{5}\bar{4}\bar{6}, \bar{6}\bar{4}\bar{8}$ |      |
| (6 9 7 8 5)     | 0 0 1 1 - - - -                       | $\bar{5}$    | $\bar{5}$ | 456, 825, 915, 657, 759   |      |

When 6 is found, 7 is also a unit clause; but the algorithm doesn’t see it yet, because variable  $x_6$  is tested first. The active ring changes first to (7 5) after 6



empty list  
cyclic DPLL  
false literals preferred

**Fig. 39.** The search tree that is implicitly traversed by Algorithm D, when that algorithm is applied to the eight unsatisfiable clauses  $R$  defined in (6). Branch nodes are labeled with the variable being tested; leaf nodes are labeled with a clause that is found to be contradicted. When the right child of a branch node is a leaf, the left branch was forced by a conditional unary clause.

is found, because 5 is cyclically after 6; we want to look at 7 before 5, instead of revisiting more or less the same clauses. After 6 has been chosen, 9 is inserted at the left, because the watch list for  $\bar{9}$  becomes nonempty. After backtracking, variables 8, 7, 9, 6 are successively inserted at the left as they lose their forced values.

The following algorithm represents the active ring by giving a NEXT field to each variable, with  $x_{\text{NEXT}(k)}$  the successor of  $x_k$ . The ring is accessed via “head” and “tail” pointers  $h$  and  $t$  at the left and right, with  $h = \text{NEXT}(t)$ . If the ring is empty, however,  $t = 0$ , and  $h$  is undefined.

**Algorithm D** (*Satisfiability by cyclic DPLL*). Given nonempty clauses  $C_1 \wedge \dots \wedge C_m$  on  $n > 0$  Boolean variables  $x_1 \dots x_n$ , represented with lazy data structures and an active ring as explained above, this algorithm finds a solution if and only if the clauses are satisfiable. It records its current progress in an array  $h_1 \dots h_n$  of indices and an array  $m_0 \dots m_n$  of “moves,” whose significance is explained below.

- D1.** [Initialize.] Set  $m_0 \leftarrow d \leftarrow h \leftarrow t \leftarrow 0$ , and do the following for  $k = n, n-1, \dots, 1$ : Set  $x_k \leftarrow -1$  (denoting an unset value); if  $W_{2k} \neq 0$  or  $W_{2k+1} \neq 0$ , set  $\text{NEXT}(k) \leftarrow h$ ,  $h \leftarrow k$ , and if  $t = 0$  also set  $t \leftarrow k$ . Finally, if  $t \neq 0$ , complete the active ring by setting  $\text{NEXT}(t) \leftarrow h$ .
- D2.** [Success?]. Terminate if  $t = 0$  (all clauses are satisfied). Otherwise set  $k \leftarrow t$ .
- D3.** [Look for unit clauses.] Set  $h \leftarrow \text{NEXT}(k)$  and use the subroutine in exercise 129 to compute  $f \leftarrow [2h \text{ is a unit}] + 2[2h+1 \text{ is a unit}]$ . If  $f = 3$ , go to D7. If  $f = 1$  or 2, set  $m_{d+1} \leftarrow f + 3$ ,  $t \leftarrow k$ , and go to D5. Otherwise, if  $h \neq t$ , set  $k \leftarrow h$  and repeat this step.
- D4.** [Two-way branch.] Set  $h \leftarrow \text{NEXT}(t)$  and  $m_{d+1} \leftarrow [W_{2h} = 0 \text{ or } W_{2h+1} \neq 0]$ .
- D5.** [Move on.] Set  $d \leftarrow d + 1$ ,  $h_d \leftarrow k \leftarrow h$ . If  $t = k$ , set  $t \leftarrow 0$ ; otherwise delete variable  $k$  from the ring by setting  $\text{NEXT}(t) \leftarrow h \leftarrow \text{NEXT}(k)$ .
- D6.** [Update watches.] Set  $b \leftarrow (m_d + 1) \bmod 2$ ,  $x_k \leftarrow b$ , and clear the watch list for  $\bar{x}_k$  (see exercise 130). Return to D2.

- D7.** [Backtrack.] Set  $t \leftarrow k$ . While  $m_d \geq 2$ , set  $k \leftarrow h_d$ ,  $x_k \leftarrow -1$ ; if  $W_{2k} \neq 0$  or  $W_{2k+1} \neq 0$ , set  $\text{NEXT}(k) \leftarrow h$ ,  $h \leftarrow k$ ,  $\text{NEXT}(t) \leftarrow h$ ; and set  $d \leftarrow d - 1$ .
- D8.** [Failure?] If  $d > 0$ , set  $m_d \leftarrow 3 - m_d$ ,  $k \leftarrow h_d$ , and return to D6. Otherwise terminate the algorithm (because the clauses aren't satisfiable). ■

move codes  
nodes  
search tree  
*langford*( $n$ )  
Langford pairs  
mems  
pure literals

The move codes of this algorithm are slightly different from the earlier ones:

- $m_j = 0$  means we're trying  $x_{h_j} = 1$  and haven't yet tried  $x_{h_j} = 0$ .
- $m_j = 1$  means we're trying  $x_{h_j} = 0$  and haven't yet tried  $x_{h_j} = 1$ .
- $m_j = 2$  means we're trying  $x_{h_j} = 1$  after  $x_{h_j} = 0$  has failed.
- $m_j = 3$  means we're trying  $x_{h_j} = 0$  after  $x_{h_j} = 1$  has failed.
- $m_j = 4$  means we're trying  $x_{h_j} = 1$  because it's forced by a unit clause.
- $m_j = 5$  means we're trying  $x_{h_j} = 0$  because it's forced by a unit clause.

As before, the number of two-way branch nodes in the implicit search tree is the number of times that  $m_j$  is set to 0 or 1.

**Comparison of the algorithms.** OK, we've just seen three rudimentary SAT solvers. How well do they actually do? Detailed performance statistics will be given later in this section, after we've studied several more algorithms. But a brief quantitative study of Algorithms A, B, and D now will give us some concrete facts with which we can calibrate our expectations before moving on.

Consider, for example, *langford*( $n$ ), the problem of Langford pairs. This problem is typical of SAT instances where many unit clauses arise during the computation. For example, when Algorithm D is applied to *langford*(5), it reaches a stage where the move codes are

$$m_1 m_2 \dots m_d = 125555555555555555114545545, \quad (60)$$

indicating only four two-way branches (the 1s and the 2) amongst a sea of forced moves. We therefore expect Algorithm D to outperform Algorithms A and B, which don't capitalize on unit clauses.

Sure enough, Algorithm D wins (slightly), even on a small example such as *langford*(5), which has 213 clauses, 480 cells, 28 variables. The detailed stats are

Algorithm A: 5379 + 108952 mems, 10552 bytes, 705 nodes.  
 Algorithm B: 1206 + 30789 mems, 4320 bytes, 771 nodes.  
 Algorithm D: 1417 + 28372 mems, 4589 bytes, 11 nodes.

(Here "5379 + 108952 mems" means that 5379 memory accesses were made while initializing the data structures before the algorithm began; then the algorithm itself accessed octabytes of memory 108,952 times.) Notice that Algorithm B is more than thrice as fast as Algorithm A in this example, although it makes 771 two-way branches instead of 705. Algorithm A needs fewer nodes, because it recognizes pure literals; but Algorithm B does much less work per node. Algorithm D, on the other hand, works very hard at each node, yet comes out ahead because its decision-making choices reduce the search to only a few nodes.

These differences become more dramatic when we consider larger problems. For instance, *langford*(9) has 1722 clauses, 3702 cells, 104 variables, and we find

Algorithm A: 332.0 megamems, 77216 bytes, 1,405,230 nodes.  
 Algorithm B: 53.4 megamems, 31104 bytes, 1,654,352 nodes.  
 Algorithm D: 23.4 megamems, 32057 bytes, 6093 nodes.

And with *langford*(13)'s 5875 clauses, 12356 cells, 228 variables, the results are

Algorithm A: 2699.1 gigamems, 253.9 kilobytes, 8.7 giganodes.  
 Algorithm B: 305.2 gigamems, 101.9 kilobytes, 10.6 giganodes.  
 Algorithm D: 71.7 gigamems, 104.0 kilobytes, 14.0 meganodes.

Mathematicians will recall that, at the beginning of Chapter 7, we used elementary reasoning to prove the unsatisfiability of *langford*( $4k + 1$ ) for all  $k$ . Evidently SAT solvers have great difficulty discovering this fact, even when  $k$  is fairly small. We are using that problem here as a benchmark test, not because we recommend replacing mathematics by brute force! Its unsatisfiability actually enhances its utility as a benchmark, because algorithms for satisfiability are more easily compared with respect to unsatisfiable instances: Extreme variations in performance occur when clauses are satisfiable, because solutions can be found purely by luck. Still, we might as well see what happens when our three algorithms are set loose on the satisfiable problem *langford*(16), which turns out to be “no sweat.” Its 11494 clauses, 23948 cells, and 352 variables lead to the statistics

Algorithm A: 11262.6 megamems, 489.2 kilobytes, 28.8 meganodes.  
 Algorithm B: 932.1 megamems, 196.2 kilobytes, 40.9 meganodes.  
 Algorithm D: 4.9 megamems, 199.4 kilobytes, 167 nodes.

Algorithm D is certainly our favorite so far, based on the *langford* data. But it is far from a panacea, because it loses badly to the lightweight Algorithm B on other problems. For example, the 2779 unsatisfiable clauses, 11662 cells, and 97 variables of *waarden*(3, 10; 97) yield

Algorithm A: 150.9 gigamems, 212.8 kilobytes, 106.7114 meganodes.  
 Algorithm B: 6.2 gigamems, 71.2 kilobytes, 106.7116 meganodes.  
 Algorithm D: 1430.4 gigamems, 72.1 kilobytes, 102.7 meganodes.

And *waarden*(3, 10; 96)'s 2721 satisfiable clauses, 11418 cells, 96 variables give us

Algorithm A: 96.9 megamems, 208.3 kilobytes, 72.9 kilonodes.  
 Algorithm B: 12.4 megamems, 69.8 kilobytes, 207.7 kilonodes.  
 Algorithm D: 57962.8 megamems, 70.6 kilobytes, 4447.7 kilonodes.

In such cases unit clauses don't reduce the search tree size by very much, so we aren't justified in spending so much time per node.

**\*Speeding up by working harder.** Algorithms A, B, and D are OK on smallish problems, but they cannot really cope with the larger instances of SAT that have arisen in our examples. Significant enhancements are possible if we are willing to do more work and to develop more elaborate algorithms.

Mathematicians generally strive for nice, short, elegant proofs of theorems; and computer scientists generally aim for nice, short, elegant sequences of steps

benchmark test  
 variations in performance  
 waerden



with which a problem can quickly be solved. But some theorems have no short proofs, and some problems cannot be solved efficiently with short programs.

Let us therefore adopt a new attitude, at least temporarily, by fearlessly deciding to throw lots of code at SAT: Let's look at the bottlenecks that hinder Algorithm D on large problems, and let's try to devise new methods that will streamline the calculations even though the resulting program might be ten times larger. In this subsection we shall examine an advanced SAT solver, Algorithm L, which is able to outperform Algorithm D by many orders of magnitude on many important problems. This algorithm cannot be described in just a few lines; but it does consist of cooperating procedures that are individually nice, short, elegant, and understandable by themselves.

The first important ingredient of Algorithm L is an improved mechanism for unit propagation. Algorithm D needs only a few lines of code in step D3 to discover whether or not the value of an unknown variable has been forced by previous assignments; but that mechanism isn't particularly fast, because it is based on indirect inferences from a lazy data structure. We can do better by using "eager" data structures that are specifically designed to recognize forced values quickly, because high-speed propagation of the consequences of a newly asserted value turns out to be extremely important in practice.

A literal  $l$  is forced true when it appears in a clause  $C$  whose other literals have become false, namely when the set of currently assigned literals  $L$  has reduced  $C$  to the unit clause  $C|L = (l)$ . Such unit clauses arise from the reduction of binary clauses. Algorithm L therefore keeps track of the binary clauses  $(u \vee v)$  that are relevant to the current subproblem  $F|L$ . This information is kept in a so-called "bimp table"  $\text{BIMP}(l)$  for every literal  $l$ , which is a list of other literals  $l'$  whose truth is implied by the truth of  $l$ . Indeed, instead of simply including binary clauses within the whole list of given clauses, as Algorithms A, B, and D do, Algorithm L stores the relevant facts about  $(u \vee v)$  directly, in a ready-to-use way, by listing  $u$  in  $\text{BIMP}(\bar{v})$  and  $v$  in  $\text{BIMP}(\bar{u})$ . Each of the  $2n$  tables  $\text{BIMP}(l)$  is represented internally as a sequential list of length  $\text{BSIZE}(l)$ , with memory allocated dynamically via the buddy system (see exercise 134).

Binary clauses, in turn, are spawned by ternary clauses. For simplicity, Algorithm L assumes that all clauses have length 3 or less, because every instance of general SAT can readily be converted to 3SAT form (see exercise 28). And for speed, Algorithm L represents the ternary clauses by means of "timp tables," which are analogous to the bimp tables: Every literal  $l$  has a sequential list  $\text{TIMP}(l)$  of length  $\text{TSIZE}(l)$ , consisting of pairs  $p_1 = (u_1, v_1)$ ,  $p_2 = (u_2, v_2)$ ,  $\dots$ , such that the truth of  $l$  implies that each  $(u_i \vee v_i)$  becomes a relevant binary clause. If  $(u \vee v \vee w)$  is a ternary clause, there will be three pairs  $p = (v, w)$ ,  $p' = (w, u)$ , and  $p'' = (u, v)$ , appearing in the respective lists  $\text{TIMP}(\bar{u})$ ,  $\text{TIMP}(\bar{v})$ , and  $\text{TIMP}(\bar{w})$ . Moreover, these three pairs are linked together cyclically, with

$$\text{LINK}(p) = p', \quad \text{LINK}(p') = p'', \quad \text{LINK}(p'') = p. \quad (61)$$

Memory is allocated for the timp tables once and for all, as the clauses are input, because Algorithm L does not generate new ternaries during its computations.

unit propagation  
lazy data structure  
eager  
unit clause  
binary clauses  
bimp table  
sequential list  
buddy system  
ternary clauses  
timp tables

Individual pairs  $p$  are, however, swapped around within these sequential tables, so that the currently active ternary clauses containing  $u$  always appear in the first  $\text{Tsize}(\bar{u})$  positions that have been allocated to  $\text{TIMP}(\bar{u})$ .

For example, let's consider again the ternary clauses (9) of *waarden*(3, 3; 9). Initially there are no binary clauses, so all BIMP tables are empty. Each of the ternary clauses appears in three of the TIMP tables. At level 0 of the search tree we might decide that  $x_5 = 0$ ; then  $\text{TIMP}(\bar{5})$  tells us that we gain eight binary clauses, namely  $\{13, 19, 28, 34, 37, 46, 67, 79\}$ . These new binary clauses are represented by sixteen entries in BIMP tables;  $\text{BIMP}(\bar{3})$ , for instance, will now be  $\{1, 4, 7\}$ . Furthermore, we'll want all of the TIMP pairs that involve either 5 or  $\bar{5}$  to become inactive, because the ternary clauses that contain 5 are weaker than the new binary clauses, and the ternary clauses that contain  $\bar{5}$  are now satisfied. (See exercise 136.)

As in (57) above, we shall assume that the variables of a given formula are numbered from 1 to  $n$ , and we represent the literals  $k$  and  $\bar{k}$  internally by the numbers  $2k$  and  $2k+1$ . Algorithm L introduces a new twist, however, by allowing variables to have many different *degrees of truth* [see M. Heule, M. Dufour, J. van Zwieten, and H. van Maaren, *LNCS 3542* (2005), 345–359]: We say that  $x_k$  is true with degree  $D$  if  $\text{VAL}[k] = D$ , and false with degree  $D$  if  $\text{VAL}[k] = D + 1$ , where  $D$  is any even number.

The highest possible degree, typically  $2^{32} - 2$  inside a computer, is called RT for “real truth.” The next highest degree, typically  $2^{32} - 4$ , is called NT for “near truth”; and then comes PT =  $2^{32} - 6$ , “proto truth.” Lower degrees PT – 2, PT – 4, . . . , 2 also turn out to be useful. A literal  $l$  is said to be *fixed in context*  $T$  if and only if  $\text{VAL}[|l|] \geq T$ ; it is *fixed true* if we also have  $\text{VAL}[|l|] \& 1 = l \& 1$ , and it is *fixed false* if its complement  $\bar{l}$  is fixed true.

Suppose, for example, that  $\text{VAL}[2] = \text{RT} + 1$  and  $\text{VAL}[7] = \text{PT}$ ; hence  $x_2$  is “really false” while  $x_7$  is “proto true.” Then the literal ‘7’, represented internally by  $l = 14$ , is fixed true in context PT, but  $l$  is not fixed in contexts NT or RT. The literal ‘ $\bar{2}$ ’, represented internally by  $l = 5$ , is fixed true in *every* context.

Algorithm L uses a sequential stack  $R_0, R_1, \dots$ , to record the names of literals that have received values. The current stack size,  $E$ , satisfies  $0 \leq E \leq n$ . With those data structures we can use a simple breadth-first search procedure to propagate the binary consequences of a literal  $l$  in context  $T$  at high speed:

$$\begin{aligned} &\text{Set } H \leftarrow E; \text{ take account of } l; \\ &\text{while } H < E, \text{ set } l \leftarrow R_H, H \leftarrow H + 1, \text{ and} \\ &\quad \text{take account of } l' \text{ for all } l' \text{ in } \text{BIMP}(l). \end{aligned} \tag{62}$$

Here “take account of  $l$ ” means “if  $l$  is fixed true in context  $T$ , do nothing; if  $l$  is fixed false in context  $T$ , go to step CONFLICT; otherwise set  $\text{VAL}[|l|] \leftarrow T + (l \& 1)$ ,  $R_E \leftarrow l$ , and  $E \leftarrow E + 1$ .” The step called CONFLICT is changeable.

A literal's BIMP table might grow repeatedly as computation proceeds. But we can undo the consequences of bad decisions by simply resetting  $\text{BSIZE}(l)$  to the value that it had before those decisions were made. A special variable *ISTAMP* is increased whenever we begin a new round of decision-making, and each

waarden  
degrees of truth  
Heule  
Dufour  
van Zwieten  
van Maaren  
RT  
real truth  
NT  
near truth  
proto truth  
PT  
fixed literals  
sequential  
stack  
breadth-first search  
undo  
ISTAMP

literal  $l$  has its private stamp  $\text{IST}(l)$ . Whenever  $\text{BSIZE}(l)$  is about to increase, we check if  $\text{IST}(l) = \text{ISTAMP}$ . If not, we set

$$\text{IST}(l) \leftarrow \text{ISTAMP}, \quad \text{ISTACK}[I] \leftarrow (l, \text{BSIZE}(l)), \quad I \leftarrow I + 1. \quad (63)$$

Then the entries on  $\text{ISTACK}$  make it easy to downdate the  $\text{BIMP}$  tables when we backtrack. (See step L13 in the algorithm below.)

We're ready now to look at the detailed steps of Algorithm L, except that one more member of its arsenal of data structures needs to be introduced: There's an array  $\text{VAR}$ , which contains a permutation of  $\{1, \dots, n\}$ , with  $\text{VAR}[k] = x$  if and only if  $\text{INX}[x] = k$ . Furthermore  $\text{VAR}[k]$  is a "free variable" — not fixed in context  $\text{RT}$ —if and only if  $0 \leq k < N$ . This setup makes it convenient to keep track of the variables that are currently free: A variable becomes fixed by swapping it to the end of the free list and decreasing  $N$  (see exercise 137); then we can free it later by simply increasing  $N$ , without swapping.

**Algorithm L** (*Satisfiability by DPLL with lookahead*). Given nonempty clauses  $C_1 \wedge \dots \wedge C_m$  of size  $\leq 3$ , on  $n > 0$  Boolean variables  $x_1 \dots x_n$ , this algorithm finds a solution if and only if the clauses are satisfiable. Its family of cooperating data structures is discussed in the text.

- L1.** [Initialize.] Record all binary clauses in the  $\text{BIMP}$  array and all ternary clauses in the  $\text{TIMP}$  array. Let  $U$  be the number of distinct variables in unit clauses; terminate unsuccessfully if two unit clauses contradict each other, otherwise record all distinct unit literals in  $\text{FORCE}[k]$  for  $0 \leq k < U$ . Set  $\text{VAR}[k] \leftarrow k + 1$  and  $\text{INX}[k + 1] \leftarrow k$  for  $0 \leq k < n$ ; and  $d \leftarrow F \leftarrow I \leftarrow \text{ISTAMP} \leftarrow 0$ . (Think  $d = \text{depth}$ ,  $F = \text{fixed variables}$ ,  $I = \text{ISTACK size}$ .)
- L2.** [New node.] Set  $\text{BRANCH}[d] \leftarrow -1$ . If  $U = 0$ , invoke Algorithm X below (which looks ahead for simplifications and also gathers data about how to make the next branch). Terminate happily if Algorithm X finds all clauses satisfied; go to L15 if Algorithm X discovers a conflict; go to L5 if  $U > 0$ .
- L3.** [Choose  $l$ .] Select a literal  $l$  that's desirable for branching (see exercise 168). If  $l = 0$ , set  $d \leftarrow d + 1$  and return to L2. Otherwise set  $\text{DEC}[d] \leftarrow l$ ,  $\text{BACKF}[d] \leftarrow F$ ,  $\text{BACKI}[d] \leftarrow I$ , and  $\text{BRANCH}[d] \leftarrow 0$ .
- L4.** [Try  $l$ .] Set  $U \leftarrow 1$ ,  $\text{FORCE}[0] \leftarrow l$ .
- L5.** [Accept near truths.] Set  $T \leftarrow \text{NT}$ ,  $G \leftarrow E \leftarrow F$ ,  $\text{ISTAMP} \leftarrow \text{ISTAMP} + 1$ , and  $\text{CONFLICT} \leftarrow \text{L11}$ . Perform the binary propagation routine (62) for  $l \leftarrow \text{FORCE}[0], \dots, l \leftarrow \text{FORCE}[U - 1]$ ; then set  $U \leftarrow 0$ .
- L6.** [Choose a nearly true  $L$ .] (At this point the stacked literals  $R_k$  are "really true" for  $0 \leq k < G$ , and "nearly true" for  $G \leq k < E$ . We want them all to be really true.) If  $G = E$ , go to L10. Otherwise set  $L \leftarrow R_G$ ,  $G \leftarrow G + 1$ .
- L7.** [Promote  $L$  to real truth.] Set  $X \leftarrow |L|$  and  $\text{VAL}[X] \leftarrow \text{RT} + L \& 1$ . Remove variable  $X$  from the free list and from all  $\text{TIMP}$  pairs (see exercise 137). Do step L8 for all pairs  $(u, v)$  in  $\text{TIMP}(L)$ , then return to L6.
- L8.** [Consider  $u \vee v$ .] (We have deduced that  $u$  or  $v$  must be true; five cases arise.) If either  $u$  or  $v$  is fixed true (in context  $T$ , which equals  $\text{NT}$ ), do

stamp  
stamping  
ISTAMP  
ISTACK  
IST  
VAR  
INX  
free variable  
DPLL with lookahead  
really true  
nearly true

nothing. If both  $u$  and  $v$  are fixed false, go to CONFLICT. If  $u$  is fixed false but  $v$  isn't fixed, perform (62) with  $l \leftarrow v$ . If  $v$  is fixed false but  $u$  isn't fixed, perform (62) with  $l \leftarrow u$ . If neither  $u$  nor  $v$  is fixed, do step L9.

- L9.** [Exploit  $u \vee v$ .] If  $\bar{v} \in \text{BIMP}(\bar{u})$ , perform (62) with  $l \leftarrow u$  (because  $\bar{u}$  implies both  $v$  and  $\bar{v}$ ). Otherwise if  $v \in \text{BIMP}(\bar{u})$ , do nothing (because we already have the clause  $u \vee v$ ). Otherwise if  $\bar{u} \in \text{BIMP}(\bar{v})$ , perform (62) with  $l \leftarrow v$ . Otherwise append  $v$  to  $\text{BIMP}(\bar{u})$  and  $u$  to  $\text{BIMP}(\bar{v})$ . (Each change to  $\text{BIMP}$  means that (63) might be invoked. Exercise 139 explains how to improve this step by deducing further implications called “compensation resolvents.”)
- L10.** [Accept real truths.] Set  $F \leftarrow E$ . If  $\text{BRANCH}[d] \geq 0$ , set  $d \leftarrow d + 1$  and go to L2. Otherwise go to L3 if  $d > 0$ , to L2 if  $d = 0$ .
- L11.** [Unfix near truths.] While  $E > G$ , set  $E \leftarrow E - 1$  and  $\text{VAL}[|R_E|] \leftarrow 0$ .
- L12.** [Unfix real truths.] While  $E > F$ , do the following: Set  $E \leftarrow E - 1$  and  $X \leftarrow |R_E|$ ; reactivate the  $\text{TIMP}$  pairs that involve  $X$  and restore  $X$  to the free list (see exercise 137); set  $\text{VAL}[X] \leftarrow 0$ .
- L13.** [Downdate  $\text{BIMPs}$ .] If  $\text{BRANCH}[d] \geq 0$ , do the following while  $I > \text{BACKI}[d]$ : Set  $I \leftarrow I - 1$  and  $\text{BSIZE}(I) \leftarrow s$ , where  $\text{ISTACK}[I] = (l, s)$ .
- L14.** [Try again?] (We've discovered that  $\text{DEC}[d]$  doesn't work.) If  $\text{BRANCH}[d] = 0$ , set  $l \leftarrow \text{DEC}[d]$ ,  $\text{DEC}[d] \leftarrow l \leftarrow \bar{l}$ ,  $\text{BRANCH}[d] \leftarrow 1$ , and go back to L4.
- L15.** [Backtrack.] Terminate unsuccessfully if  $d = 0$ . Otherwise set  $d \leftarrow d - 1$ ,  $E \leftarrow F$ ,  $F \leftarrow \text{BACKF}[d]$ , and return to L12. ■

Exercise 143 extends this algorithm so that it will handle clauses of arbitrary size.

**\*Speeding up by looking ahead.** Algorithm L as it stands is incomplete, because step L2 relies on an as-yet-unspecified “Algorithm X” before choosing a literal for branching. If we use the simplest possible Algorithm X, by branching on whatever literal happens to be first in the current list of free variables, the streamlined methods for propagating forced moves in (62) and (63) will tend to make Algorithm L run roughly three times as fast as Algorithm D, and that isn't a negligible improvement. But with a sophisticated Algorithm X we can often gain another factor of 10 or more in speed, on significant problems.

For example, here are some typical empirical statistics:

| Problem                      | Algorithm D                      | Algorithm L <sup>0</sup>         | Algorithm L <sup>+</sup>        |
|------------------------------|----------------------------------|----------------------------------|---------------------------------|
| <i>waerden</i> (3, 10; 97)   | 1430 gigamems,<br>103 meganodes  | 391 gigamems,<br>31 meganodes    | 772 megamems,<br>4672 nodes     |
| <i>langford</i> (13)         | 71.7 gigamems,<br>14.0 meganodes | 21.5 gigamems,<br>10.9 meganodes | 45.7 gigamems,<br>944 kilonodes |
| <i>rand</i> (3, 420, 100, 0) | 184 megamems,<br>34 kilonodes    | 34 megamems,<br>7489 nodes       | 626 kilomems,<br>19 nodes       |

Here Algorithm L<sup>0</sup> stands for Algorithm L with the simplest Algorithm X, while Algorithm L<sup>+</sup> uses all of the lookahead heuristics that we are about to discuss. The first two problems involve rather large clauses, so they use the extended

compensation resolvents  
langford  
waerden  
rand+  
Algorithm L<sup>0</sup>

Algorithm L of exercise 143. The third problem consists of 420 random ternary clauses on 100 variables. (Algorithm B, incidentally, needs 80.1 teramems, and a search tree of 4.50 teranodes, to show that those clauses are unsatisfiable.)

random ternary clauses  
Heule  
**march**  
heuristic score

The moral of this story is that it's wise to do 100 times as much computation at every node of a large search tree, if we can thereby decrease the size of the tree by a factor of 1000.

How then can we distinguish a variable that's good for branching from a variable that isn't? We shall consider a three-step approach:

- Preselecting, to identify free variables that appear to be good candidates;
- Nesting, to allow candidate literals to share implied computations;
- Exploring, to examine the immediate consequences of hypothetical decisions.

While carrying out these steps, Algorithm X might discover a contradiction (in which case Algorithm L will take charge again at step L15); or the lookahead process might discover that several of the free literals are forced to be true (in which case it places them in the first  $U$  positions of the FORCE array). The explorations might even discover a way to satisfy all of the clauses (in which case Algorithm L will terminate and everybody will be happy). Thus, Algorithm X might do much more than simply choose a good variable on which to branch.

The following recommendations for Algorithm X are based on Marijn Heule's lookahead solver called **march**, one of the world's best, as it existed in 2013.

The first stage, preselection, is conceptually simplest, although it also involves some "handwaving" because it depends on necessarily shaky assumptions. Suppose there are  $N$  free variables. Experience has shown that we tend to get a good heuristic score  $h(l)$  for each literal  $l$ , representing the relative amount by which asserting  $l$  will reduce the current problem, if these scores approximately satisfy the simultaneous nonlinear equations

$$h(l) = 0.1 + \alpha \sum_{\substack{u \in \text{BIMP}(l) \\ u \text{ not fixed}}} \hat{h}(u) + \sum_{(u,v) \in \text{TIMP}(l)} \hat{h}(u) \hat{h}(v). \quad (64)$$

Here  $\alpha$  is a magic constant, typically 3.5; and  $\hat{h}(l)$  is a multiple of  $h(l)$  chosen so that  $\sum_l \hat{h}(l) = 2N$  is the total number of free literals. (In other words, the  $h$  scores on the right are "normalized" so that their average is 1.)

Any given set of scores  $h(l)$  can be used to derive a refined set  $h'(l)$  by letting

$$h'(l) = 0.1 + \alpha \sum_{\substack{u \in \text{BIMP}(l) \\ u \text{ not fixed}}} \frac{h(u)}{h_{\text{ave}}} + \sum_{(u,v) \in \text{TIMP}(l)} \frac{h(u)}{h_{\text{ave}}} \frac{h(v)}{h_{\text{ave}}}, \quad h_{\text{ave}} = \frac{1}{2N} \sum_l h(l). \quad (65)$$

Near the root of the search tree, when  $d \leq 1$ , we start with  $h(l) = 1$  for all  $l$  and then refine it five times (say). At deeper levels we start with the  $h(l)$  values from the parent node and refine them once. Exercise 145 contains an example.

We've computed  $h(l)$  for all of the free literals  $l$ , but we won't have time to explore them all. The next step is to select free variables  $\text{CAND}[j]$  for  $0 \leq j < C$ , where  $C$  isn't too large; we will insist that the number of candidates does not exceed

$$C_{\text{max}} = \max(C_0, C_1/d), \quad (66)$$

using cutoff parameters that are typically  $C_0 = 30, C_1 = 600$ . (See exercise 148.)

We start by dividing the free variables into “participants” and “newbies”: A *participant* is a variable such that either  $x$  or  $\bar{x}$  has played the role of  $u$  or  $v$  in step L8, at some node above us in the search tree; a *newbie* is a nonparticipant. When  $d = 0$  every variable is a newbie, because we’re at the root of the tree. But usually there is at least one participant, and we want to branch only on participants whenever possible, in order to maintain focus while backtracking.

If we’ve got too many potential candidates, even after restricting consideration to participants, we can winnow the list down by preferring the variables  $x$  that have the largest combined score  $h(x)h(\bar{x})$ . Step X3 below describes a fairly fast way to come up with the desired selection of  $C \leq C_{\max}$  candidates.

A simple lookahead algorithm can now proceed to compute a more accurate heuristic score  $H(l)$ , for each of the  $2C$  literals  $l = \text{CAND}[j]$  or  $l = \neg\text{CAND}[j]$  that we’ve selected for further scrutiny. The idea is to simulate what would happen if  $l$  were used for branching, by mimicking steps L4–L9 (at least to a first approximation): Unit literals are propagated as in the exact algorithm, but whenever we get to the part of step L9 that changes the BIMP tables, we don’t actually make such a change; we simply note that a branch on  $l$  would imply  $u \vee v$ , and we consider the value of that potential new clause to be  $h(u)h(v)$ . The heuristic score  $H(l)$  is then defined to be the sum of all such clause weights:

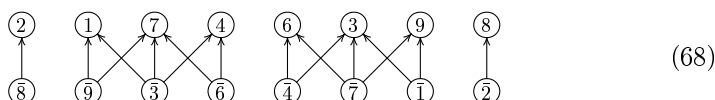
$$H(l) = \sum \{h(u)h(v) \mid \text{asserting } l \text{ in L4 leads to asserting } u \vee v \text{ in L9}\}. \quad (67)$$

For example, the problem *waerden* (3, 3; 9) of (9) has nine candidate variables  $\{1, 2, \dots, 9\}$  at the root of the search tree, and exercise 145 finds their rough heuristic scores  $h(l)$ . The more discriminating scores  $H(l)$  turn out to be

$$\begin{aligned} H(1) &= h(2)h(3) + h(3)h(5) + h(4)h(7) + h(5)h(9) = 168.6; \\ H(2) &= h(1)h(3) + h(3)h(4) + h(4)h(6) + h(5)h(8) = 157.3; \\ H(3) &= h(1)h(2) + h(2)h(4) + h(4)h(5) + \dots + h(6)h(9) = 233.4; \\ H(4) &= h(2)h(3) + h(3)h(5) + h(5)h(6) + \dots + h(1)h(7) = 231.8; \\ H(5) &= h(3)h(4) + h(3)h(6) + h(6)h(7) + \dots + h(1)h(9) = 284.0. \end{aligned}$$

This problem is symmetrical, so we also have  $H(6) = H(\bar{6}) = H(4) = H(\bar{4})$ , etc. The best literal for branching, according to this estimate, is 5 or  $\bar{5}$ .

Suppose we set  $x_5$  false and proceed to look ahead at the reduced problem, with  $d = 1$ . At this point there are eight candidates,  $\{1, 2, 3, 4, 6, 7, 8, 9\}$ ; and they’re now related also by binary implications, because the original clause ‘357’ has, for instance, been reduced to ‘37’. In fact, the BIMP tables now define the dependency digraph

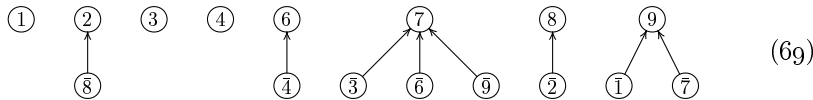


because  $\bar{3} \rightarrow 7$ , etc.; and in general the  $2C$  candidate literals will define a dependency digraph whose structure yields important clues about the current subproblem. We can, for example, use Tarjan’s algorithm to find the strong

cutoff parameters  
 participants  
 newbies  
 focus  
 heuristic score  
 dependency digraph  
 binary implication graph, see dependency d  
 Tarjan  
 strong components

components of that digraph, as mentioned after Theorem 7.1.1K. If some strong component includes both  $l$  and  $\bar{l}$ , the current subproblem is unsatisfiable. Otherwise two literals of the same component are constrained to have the same value; so we shall choose one literal from each of the  $S \leq 2C$  strong components, and use those choices as the actual candidates for lookahead.

Continuing our example, at this point we can use a nice trick to save redundant computation, by extracting a *subforest* of the dependency digraph:



subforest  
 Tarjan  
 preorder  
 postorder  
 lookahead forest  
 tree-based lookahead, see lookahead forest  
 proto true  
 PT

The relation  $\bar{8} \rightarrow 2$  means that whatever happens after asserting the literal ‘2’ will also happen after asserting ‘ $\bar{8}$ ’; hence we need not repeat the steps for ‘2’ while studying ‘ $\bar{8}$ ’. And similarly, each of the other subordinate literals ‘ $\bar{1}$ ’,  $\dots$ , ‘ $\bar{9}$ ’ inherits the assertions of its parent in this hierarchy. Tarjan’s algorithm actually produces such a subforest with comparatively little extra work.

The nested structure of a forest also fits beautifully with “degrees of truth” in our data structure, if we visit the  $S$  candidate literals in *preorder* of the subforest, and if we successively assert each literal  $l$  at the truth degree that corresponds to twice its position in *postorder*. For instance, (69) becomes the following arrangement, which we shall call the “lookahead forest”:

$$\begin{array}{cccccccccccccccc} \text{preorder} & 1 & 2 & \bar{8} & 3 & 4 & 6 & \bar{4} & 7 & \bar{3} & \bar{6} & \bar{9} & 8 & \bar{2} & 9 & \bar{1} & \bar{7} \\ 2\cdot\text{postorder} & 2 & 6 & 4 & 8 & 10 & 14 & 12 & 22 & 16 & 18 & 20 & 26 & 24 & 32 & 28 & 30 \end{array} \quad (70)$$

A simulation of steps L4–L9 with  $l \leftarrow 1$  and  $T \leftarrow 2$  makes  $x_1$  true at degree 2 (we say that it’s “2fixed” or “2true”); it also computes the score  $H(1) \leftarrow h(\bar{2})h(\bar{3}) + h(\bar{4})h(\bar{7})$ , but it spawns no other activity if Algorithm Y below isn’t active. Simulation with  $l \leftarrow 2$  and  $T \leftarrow 6$  then 6fixes 2 and computes  $H(2) \leftarrow h(\bar{1})h(\bar{3}) + h(\bar{4})h(\bar{6})$ ; during this process the value of  $x_1$  isn’t seen, because it is less than  $T$ . But things get more interesting when  $l \leftarrow \bar{8}$  and  $T \leftarrow 4$ : Now we 4fix  $\bar{8}$ , and we’re still able to see that  $x_2$  is true because  $6 > T$ . So we save a little computation by inheriting  $H(2)$  and setting  $H(\bar{8}) \leftarrow H(2) + h(4)h(6) + h(6)h(7) + h(7)h(9)$ .

The real action begins to break through a few steps later, when we set  $l \leftarrow \bar{4}$  and  $T \leftarrow 12$ . Then (62) will 12fix not only  $\bar{4}$  but also 3, since  $\bar{4} \rightarrow 3$ ; and the 12truth of 3 will soon take us to the simulated step L8 with  $u = \bar{6}$  and  $v = \bar{9}$ . Aha: We 12fix  $\bar{9}$ , because 6 is 14true. Then we also 12fix the literals 7, 1,  $\dots$ , and reach a contradiction. *This contradiction shows that branching on  $\bar{4}$  will lead to a conflict*; hence the literal 4 must be true, if the current clauses are satisfiable.

Whenever the lookahead simulation of Algorithm X learns that some literal  $l$  must be true, as in this example, it places  $l$  on the FORCE list and makes  $l$  *proto true* (that is, true in context PT). A proto true literal will remain fixed true throughout this round of lookahead, because all relevant values of  $T$  will be less than PT. Later, Algorithm L will promote proto truth to near truth, and ultimately to real truth—unless a contradiction arises. (And in the case of *waerden*(3, 3; 9), such a contradiction does in fact arise; see exercise 150.)

Why does the combination of preorder and postorder work so magically in (70)? It's because of a basic property of forests in general, which we noted for example in exercise 2.3.2–20: *If  $u$  and  $v$  are nodes of a forest,  $u$  is a proper ancestor of  $v$  if and only if  $u$  precedes  $v$  in preorder and  $u$  follows  $v$  in postorder.* Moreover, when we look ahead at candidate literals in this way, an important invariant relation is maintained on the  $R$  stack, namely that truth degrees never increase as we move from the bottom to the top:

$$\text{VAL}[|R_{j-1}|] \mid 1 \geq \text{VAL}[|R_j|], \quad \text{for } 1 < j < E. \quad (71)$$

Real truths appear at the bottom, then near truths, then proto truth, etc. For example, the stack at one point in the problem above contains seven literals,

$$\begin{array}{rcccccc} j = & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ R_j = & \bar{5} & 6 & \bar{4} & 3 & \bar{9} & 7 & 1 \quad . \\ \text{VAL}[|R_j|] = & \text{RT}+1 & 14 & 13 & 12 & 13 & 12 & 12 \end{array}$$

One consequence is that the current visibility of truth values matches the recursive structure by which false literals are purged from ternary clauses.

The second phase of Algorithm X, after preselection of candidates, is called “nesting,” because it constructs a lookahead forest analogous to (70). More precisely, it constructs a sequence of literals  $\text{LL}[j]$  and corresponding truth offsets  $\text{LO}[j]$ , for  $0 \leq j < S$ . It also sets up  $\text{PARENT}$  pointers to indicate the forest structure more directly; for example, with (69) we would have  $\text{PARENT}(\bar{8}) = 2$  and  $\text{PARENT}(2) = \Lambda$ .

The third phase, “exploration,” now does the real work. It uses the lookahead forest to evaluate heuristics  $H(l)$  for the candidate literals — and also (if it's lucky) to discover literals whose values are forced.

The heart of the exploration phase is a breadth-first search based on steps L5, L6, and L8. This routine propagates truth values of degree  $T$  and also computes  $w$ , the weight of new binary clauses that would be spawned by branching on  $l$ :

$$\begin{array}{l} \text{Set } l_0 \leftarrow l, i \leftarrow w \leftarrow 0, \text{ and } G \leftarrow E \leftarrow F; \text{ perform (62);} \\ \text{while } G < E, \text{ set } L \leftarrow R_G, G \leftarrow G + 1, \text{ and} \\ \quad \text{take account of } (u, v) \text{ for all } (u, v) \text{ in } \text{TIMP}(L); \\ \text{generate new binary clauses } (\bar{l}_0 \vee W_k) \text{ for } 0 \leq k < i. \end{array} \quad (72)$$

Here “take account of  $(u, v)$ ” means “if either  $u$  or  $v$  is fixed true (in context  $T$ ), do nothing; if both  $u$  and  $v$  are fixed false, go to  $\text{CONFLICT}$ ; if  $u$  is fixed false but  $v$  isn't fixed, set  $W_i \leftarrow v$ ,  $i \leftarrow i + 1$ , and perform (62) with  $l \leftarrow v$ ; if  $v$  is fixed false but  $u$  isn't fixed, set  $W_i \leftarrow u$ ,  $i \leftarrow i + 1$ , and perform (62) with  $l \leftarrow u$ ; if neither  $u$  nor  $v$  is fixed, set  $w \leftarrow w + h(u)h(v)$ .”

*Explanation:* A ternary clause of the form  $\bar{L} \vee u \vee v$ , where  $L$  is fixed true and  $u$  is fixed false as a consequence of  $l_0$  being fixed true, is called a “windfall.” Such clauses are good news, because they imply that the *binary* clause  $\bar{l}_0 \vee v$  must be satisfied in the current subproblem. Windfalls are recorded on a stack called  $W$ , and appended to the  $\text{BIMP}$  database at the end of (72).

forests  
ancestor  
invariant relation  
nesting  
lookahead forest  
exploration  
breadth-first search  
windfall



The exploration phase also exploits an important fact called the *autarky principle*, which generalizes the notion of “pure literal” that we discussed above in connection with Algorithm A. An “autarky” for a SAT problem  $F$  is a set of strictly distinct literals  $A = \{a_1, \dots, a_t\}$  with the property that every clause of  $F$  either contains at least one literal of  $A$  or contains none of the literals of  $\overline{A} = \{\overline{a}_1, \dots, \overline{a}_t\}$ . In other words,  $A$  satisfies every clause that  $A$  or  $\overline{A}$  “touches.”

An autarky is a self-sufficient system. Whenever  $A$  is an autarky, we can assume without loss of generality that all of its literals are actually true; for if  $F$  is satisfiable, the untouched clauses are satisfiable, and  $A$  tells us how to satisfy the touched ones. Step X9 of the following algorithm shows that we can detect certain autarkies easily while we’re looking ahead.

autarky principle  
pure literal  
touches  
participants

**Algorithm X** (*Lookahead for Algorithm L*). This algorithm, which is invoked in step L2 of Algorithm L, uses the data structures of that algorithm together with additional arrays of its own to explore properties of the current subproblem. It discovers  $U \geq 0$  literals whose values are forced, and puts them in the FORCE array. It terminates either by (i) satisfying all clauses; (ii) finding a contradiction; or (iii) computing heuristic scores  $H(l)$  that will allow step L3 to choose a good literal for branching. In case (iii) it might also discover new binary clauses.

- X1.** [Satisfied?] If  $F = n$ , terminate happily (no variables are free).
- X2.** [Compile rough heuristics.] Set  $N = n - F$  and use (65) to compute a rough score  $h(l)$  for each free literal  $l$ .
- X3.** [Preselect candidates.] Let  $C$  be the current number of free variables that are “participants,” and put them into the CAND array. If  $C = 0$ , set  $C \leftarrow N$  and put *all* free variables into CAND; terminate happily, however, if all clauses are satisfied (see exercise 152). Give each variable  $x$  in CAND the rating  $r(x) = h(x)h(\overline{x})$ . Then while  $C > 2C_{\max}$  (see (66)), delete all elements of CAND whose rating exceeds the mean rating; but terminate this loop if no elements are actually deleted. Finally, if  $C > C_{\max}$ , reduce  $C$  to  $C_{\max}$  by retaining only top-ranked candidates. (See exercise 153.)
- X4.** [Nest the candidates.] Construct a lookahead forest, represented in LL[ $j$ ] and LO[ $j$ ] for  $0 \leq j < S$  and by PARENT pointers (see exercise 155).
- X5.** [Prepare to explore.] Set  $U' \leftarrow j' \leftarrow \text{BASE} \leftarrow j \leftarrow 0$  and CONFLICT  $\leftarrow$  X13.
- X6.** [Choose  $l$  for lookahead.] Set  $l \leftarrow \text{LL}[j]$  and  $T \leftarrow \text{BASE} + \text{LO}[j]$ . Set  $H(l) \leftarrow H(\text{PARENT}(l))$ , where  $H(\Lambda) = 0$ . If  $l$  is not fixed in context  $T$ , go to X8. Otherwise, if  $l$  is fixed false but not proto false, do step X12 with  $l \leftarrow \overline{l}$ .
- X7.** [Move to next.] If  $U > U'$ , set  $U' \leftarrow U$  and  $j' \leftarrow j$ . Then set  $j \leftarrow j + 1$ . If  $j = S$ , set  $j \leftarrow 0$  and  $\text{BASE} \leftarrow \text{BASE} + 2S$ . Terminate normally if  $j = j'$ , or if  $j = 0$  and  $\text{BASE} + 2S \geq \text{PT}$ . Otherwise return to X6.
- X8.** [Compute sharper heuristic.] Perform (72). Then if  $w > 0$ , set  $H(l_0) \leftarrow H(l_0) + w$  and go to X10.
- X9.** [Exploit an autarky.] If  $H(l_0) = 0$ , do step X12 with  $l \leftarrow l_0$ . Otherwise generate the new binary clause  $l_0 \vee \neg \text{PARENT}(l_0)$ . (Exercise 166 explains why.)

**X10.** [Optionally look deeper.] Perform Algorithm Y below.

**X11.** [Exploit necessary assignments.] Do step X12 for all literals  $l \in \text{BIMP}(\bar{l}_0)$  that are fixed true but not proto true. Then go to X7. (See exercise 167.)

**X12.** [Force  $l$ .] Set  $\text{FORCE}[U] \leftarrow l$ ,  $U \leftarrow U + 1$ ,  $T' \leftarrow T$ , and perform (72) with  $T \leftarrow \text{PT}$ . Then set  $T \leftarrow T'$ . (This step is a subroutine, used by other steps.)

**X13.** [Recover from conflict.] If  $T < \text{PT}$ , do step X12 with  $l \leftarrow \bar{l}_0$  and go to X7. Otherwise terminate with a contradiction. ■

Notice that, in steps X5–X7, this algorithm proceeds cyclically through the forest, continuing to look ahead until completing a pass in which no new forced literals are found. The BASE address of truth values continues to grow, if necessary, but it isn't allowed to become too close to PT.

**\*Looking even further ahead.** If it's a good idea to look one step ahead, maybe it's a better idea to look *two* steps ahead. Of course that's a somewhat scary proposition, because our data structures are already pretty stretched; and besides, double lookahead might take way too much time. Nevertheless, there's a way to pull it off, and to make Algorithm L run even faster on many problems.

Algorithm X looks at the immediate consequences of assuming that some literal  $l_0$  is true. Algorithm Y, which is launched in step X10, goes further out on that limb, and investigates what would happen if *another* literal,  $\hat{l}_0$ , were also true. The goal is to detect branches that die off early, allowing us to discover new implications of  $l_0$  or even to conclude that  $l_0$  must be false.

For this purpose Algorithm Y stakes out an area of truth space between the current context  $T$  and a degree of truth called “double truth” or DT, which is defined in step Y2. The size of this area is determined by a parameter  $Y$ , which is typically less than 10. The same lookahead forest is used to give relative truth degrees below DT. Double truth is less trustworthy than proto truth, PT; but literals that are fixed at level DT are known to be conditionally true (“Dtrue”) or conditionally false (“Dfalse”) under the hypothesis that  $l_0$  is true.

Going back to our example of *waarden*(3, 3; 9), the scenario described above was based on the assumption that double lookahead was not done. Actually, however, further activity by Algorithm Y will usually take place after  $H(1)$  has been set to  $h(\bar{2})h(\bar{3}) + h(\bar{4})h(\bar{7})$ . The value of DT will be set to 130, assuming that  $Y = 8$ , because  $S = 8$ . Literal 1 will become Dtrue. Looking then at 2 will 6fix 2; and that will 6fix  $\bar{3}$  because of the clause  $\bar{1}\bar{2}\bar{3}$ . Then  $\bar{3}$  will 6fix 4 and 7, contradicting  $\bar{1}\bar{4}\bar{7}$  and causing 2 to become Dfalse. Other literals also will soon become Dtrue or Dfalse, leading to a contradiction; and that contradiction will allow Algorithm Y to make literal 1 proto false before Algorithm X has even begun to look ahead at literal 2.

The main loop of double lookahead is analogous to (72), but it's simpler, because we're further removed from reality:

$$\begin{aligned} &\text{Set } \hat{l}_0 \leftarrow l \text{ and } G \leftarrow E \leftarrow F; \text{ perform (62);} \\ &\text{while } G < E, \text{ set } L \leftarrow R_G, G \leftarrow G + 1, \text{ and} \\ &\quad \text{take account of } (u, v) \text{ for all } (u, v) \text{ in } \text{TIMP}(L). \end{aligned} \tag{73}$$

necessary assignments  
forced literals  
double truth  
DT  
Dtrue  
Dfalse

Now “take account of  $(u, v)$ ” means “if either  $u$  or  $v$  is fixed true (in context  $T$ ), or if neither  $u$  nor  $v$  is fixed, do nothing; if both  $u$  and  $v$  are fixed false, go to CONFLICT; if  $u$  is fixed false but  $v$  isn’t fixed, perform (62) with  $l \leftarrow v$ ; if  $v$  is fixed false but  $u$  isn’t fixed, perform (62) with  $l \leftarrow u$ .”

Since double-looking is costly, we want to try it only when there’s a fairly good chance that it will be helpful, namely when  $H(l_0)$  is large. But how large is large enough? The proper threshold depends on the problem being solved: Some sets of clauses are handled more quickly by double-looking, while others are immune to such insights. Marijn Heule and Hans van Maaren [*LNCS 4501* (2007), 258–271] have developed an elegant feedback mechanism that automatically tunes itself to the characteristics of the problem at hand: Let  $\tau$  be a “trigger,” initially 0. Step Y1 allows double-look only if  $H(l_0) > \tau$ ; otherwise  $\tau$  is decreased to  $\beta\tau$ , where  $\beta$  is a damping factor (typically 0.999), so that double-looking will become more attractive. On the other hand if double-look doesn’t find a contradiction that makes  $l_0$  proto false, the trigger is raised to  $H(l_0)$  in step Y6.

**Algorithm Y** (*Double lookahead for Algorithm X*). This algorithm, invoked in step X10, uses the same data structures (and a few more) to look ahead more deeply. Parameters  $\beta$  and  $Y$  are explained above. Initially  $\text{DFAIL}(l) = 0$  for all  $l$ .

- Y1.** [Filter.] Terminate if  $\text{DFAIL}(l_0) = \text{ISTAMP}$ , or if  $T + 2S(Y + 1) > \text{PT}$ . Otherwise, if  $H(l_0) \leq \tau$ , set  $\tau \leftarrow \beta\tau$  and terminate.
- Y2.** [Initialize.] Set  $\text{BASE} \leftarrow T - 2$ ,  $\text{LBASE} \leftarrow \text{BASE} + 2S \cdot Y$ ,  $\text{DT} \leftarrow \text{LBASE} + \text{LO}[j]$ ,  $i \leftarrow \hat{j}' \leftarrow \hat{j} \leftarrow 0$ ,  $E \leftarrow F$ , and  $\text{CONFLICT} \leftarrow \text{Y8}$ . Perform (62) with  $l \leftarrow l_0$  and  $T \leftarrow \text{DT}$ .
- Y3.** [Choose  $l$  for double look.] Set  $l \leftarrow \text{LL}[j]$  and  $T \leftarrow \text{BASE} + \text{LO}[j]$ . If  $l$  is not fixed in context  $T$ , go to Y5. Otherwise, if  $l$  is fixed false but not Dfalse, do step Y7 with  $l \leftarrow \bar{l}$ .
- Y4.** [Move to next.] Set  $\hat{j}' \leftarrow \hat{j} + 1$ . If  $\hat{j}' = S$ , set  $\hat{j}' \leftarrow 0$  and  $\text{BASE} \leftarrow \text{BASE} + 2S$ . Go to Y6 if  $\hat{j}' = \hat{j}$ , or if  $\hat{j}' = 0$  and  $\text{BASE} = \text{LBASE}$ . Otherwise return to Y3.
- Y5.** [Look ahead.] Perform (73), and return to Y4 (if no conflict arises).
- Y6.** [Finish.] Generate new binary clauses  $(\bar{l}_0 \vee W_k)$  for  $0 \leq k < i$ . Then set  $\text{BASE} \leftarrow \text{LBASE}$ ,  $T \leftarrow \text{DT}$ ,  $\tau \leftarrow H(l_0)$ ,  $\text{DFAIL}(l_0) \leftarrow \text{ISTAMP}$ ,  $\text{CONFLICT} \leftarrow \text{X13}$ , and terminate.
- Y7.** [Make  $\hat{l}_0$  false.] Set  $\hat{j}' \leftarrow \hat{j}$ ,  $T' \leftarrow T$ , and perform (73) with  $l \leftarrow \hat{l}_0$  and  $T \leftarrow \text{DT}$ . Then set  $T \leftarrow T'$ ,  $W_i \leftarrow \hat{l}_0$ ,  $i \leftarrow i + 1$ . (This step is a subroutine.)
- Y8.** [Recover from conflict.] If  $T < \text{DT}$ , do step Y7 with  $l \leftarrow \neg \text{LL}[\hat{j}']$  and go to Y4. Otherwise set  $\text{CONFLICT} \leftarrow \text{X13}$  and exit to X13. ■

Some quantitative statistics will help to ground these algorithms in reality: When Algorithm L was let loose on *rand*(3, 2062, 500, 314), a problem with 500 variables and 2062 random ternary clauses, it proved unsatisfiability after making 684,433,234,661 memory accesses and constructing a search tree of 9,530,489 nodes. Exercise 173 explains what would have happened if various parts of the algorithm had been disabled. None of the other SAT solvers we shall discuss are able to handle such random problems in a reasonable amount of time.

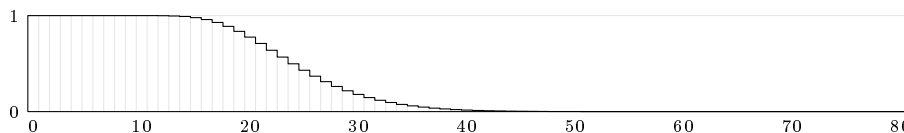
Heule  
van Maaren  
feedback mechanism  
adaptive control  
trigger  
damping factor  
DFAIL  
 $\pi$   
random ternary clauses

**Random satisfiability.** There seems to be no easy way to analyze the satisfiability problem under random conditions. In fact, the basic question “How many random clauses of 3SAT on  $n$  variables do we need to consider, on the average, before they can’t all be satisfied?” is a famous unsolved research problem.

From a practical standpoint this question isn’t as relevant as the analogous questions were when we studied algorithms for sorting or searching, because real-world instances of 3SAT tend to have highly *nonrandom* clauses. Deviations from randomness in combinatorial algorithms often have a dramatic effect on running time, while methods of sorting and searching generally stay reasonably close to their expected behavior. Thus a focus on randomness can be misleading. On the other hand, random SAT clauses do serve as a nice, clean model, so they give us insights into what goes on in Boolean territory. Furthermore the mathematical issues are of great interest in their own right. And fortunately, much of the basic theory is in fact elementary and easy to understand. So let’s take a look at it.

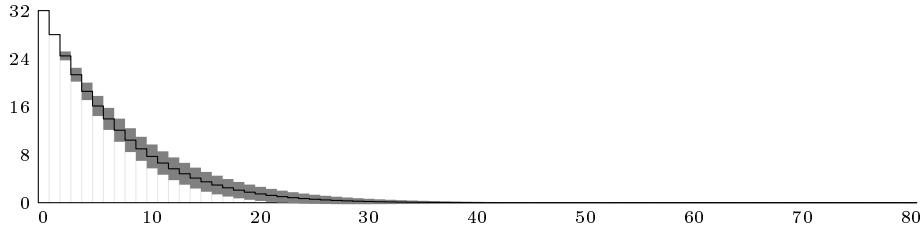
Exercise 180 shows that random satisfiability can be analyzed *exactly*, when there are at most five variables. We might as well start there, because the “tiny” 5-variable case is still large enough to shed some light on the bigger picture. When there are  $n$  variables and  $k$  literals per clause, the number  $N$  of possible clauses that involve  $k$  different variables is clearly  $2^k \binom{n}{k}$ : There are  $\binom{n}{k}$  ways to choose the variables, and  $2^k$  ways to either complement or not. So we have, for example,  $N = 2^3 \binom{5}{3} = 80$  possible clauses in a 3SAT problem on 5 variables.

Let  $q_m$  be the probability that  $m$  of those clauses, distinct but otherwise selected at random, are satisfiable. Thus  $q_m = Q_m / \binom{N}{m}$ , where  $Q_m$  is the number of ways to choose  $m$  of the  $N$  clauses so that at least one Boolean vector  $x = x_1 \dots x_n$  satisfies them all. Figure 40 illustrates these probabilities when  $k = 3$  and  $n = 5$ . Suppose we’re being given distinct random clauses one by one. According to Fig. 40, the chances are better than 77% that we’ll still be able to satisfy them after 20 different clauses have been received, because  $q_{20} \approx 0.776$ . But by the time we’ve accumulated 30 of the 80 clauses, the chance of satisfiability has dropped to  $q_{30} \approx 0.179$ ; and after ten more we reach  $q_{40} \approx 0.016$ .



**Fig. 40.** The probability  $q_m$  that  $m$  distinct clauses of 3SAT on 5 variables are simultaneously satisfiable, for  $0 \leq m \leq 80$ .

The illustration makes it appear as if  $q_m = 1$  for  $m < 15$ , say, and as if  $q_m = 0$  for  $m > 55$ . But  $q_8$  is actually *less* than 1, because of (6); exercise 179 gives the exact value. And  $q_{70}$  is *greater* than 0, because  $Q_{70} = 32$ ; indeed, every Boolean vector  $x$  satisfies exactly  $(2^k - 1) \binom{n}{k} = (1 - 2^{-k})N$  of the  $N$  possible  $k$ -clauses, so it’s no surprise that 70 noncontradictory 3-clauses on 5 variables can be found. Of course those clauses will hardly ever be the first 70 received, in a random situation. The actual value of  $q_{70}$  is  $32/1646492110120 \approx 2 \times 10^{-11}$ .

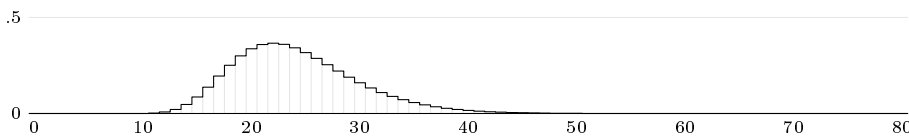


solutions, number of+  
standard deviation  
uniquely satisfiable  
stopping time  
summation by parts

**Fig. 41.** The total number  $T_m$  of different Boolean vectors  $x = x_1 \dots x_5$  that simultaneously satisfy  $m$  distinct clauses of 3SAT on 5 variables, for  $0 \leq m \leq 80$ .

Figure 41 portrays the same process from another standpoint: It shows *in how many ways* a random set of  $m$  clauses can be satisfied. This value,  $T_m$ , is a random variable whose mean is indicated in black, surrounded by a gray region that shows the mean plus-or-minus the standard deviation. For example,  $T_0$  is always 32, and  $T_1$  is always 28; but  $T_2$  is either 24, 25, or 26, and it takes these values with the respective probabilities  $(2200, 480, 480)/3160$ . Thus the mean for  $m = 2$  is  $\approx 24.5$ , and the standard deviation is  $\approx 0.743$ .

When  $m = 20$ , we know from Fig. 40 that  $T_{20}$  is nonzero more than 77% of the time; yet Fig. 41 shows that  $T_{20} \approx 1.47 \pm 1.17$ . (Here the notation  $\mu \pm \sigma$  stands for the mean value  $\mu$  with standard deviation  $\sigma$ .) It turns out, in fact, that 20 random clauses are *uniquely satisfiable*, with  $T_{20} = 1$ , more than 33% of the time; and the probability that  $T_{20} > 4$  is only 0.013. With 30 clauses, satisfiability gets dicier and dicier:  $T_{30} \approx 0.20 \pm 0.45$ ; indeed,  $T_{30}$  is less than 2, more than 98% of the time — although it can be as high as 11 if the clause-provider is being nice to us. By the time 40 clauses are reached, the odds that  $T_{40}$  exceeds 1 are less than 1 in 4700. Figure 42 shows the probability that  $T_m = 1$  as  $m$  varies.

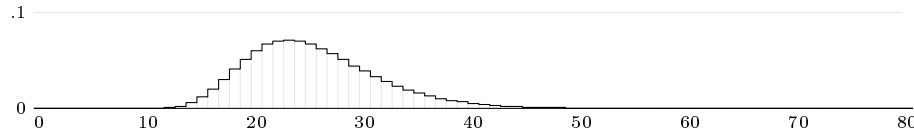


**Fig. 42.**  $\Pr(T_m = 1)$ , the probability that  $m$  distinct clauses of 3SAT on 5 variables are uniquely satisfiable, for  $0 \leq m \leq 80$ .

Let  $P$  be the number of clauses that have been received when we're first unable to satisfy them all. Thus we have  $P = m$  with probability  $p_m$ , where  $p_m = q_{m-1} - q_m$  is the probability that  $m - 1$  random clauses are satisfiable but  $m$  are not. These probabilities are illustrated in Fig. 43. Is it surprising that Figs. 42 and 43 look roughly the same? (See exercise 183.)

The expected “stopping time,”  $E P$ , is by definition equal to  $\sum_m m p_m$ ; and it's not difficult to see, for example by using the technique of summation by parts (exercise 1.2.7–10), that we can compute it by summing the probabilities in Fig. 40:

$$E P = \sum_m q_m. \tag{74}$$



**Fig. 43.** The stopping time probabilities,  $p_m$ , that  $m$  distinct clauses of 3SAT on 5 variables have just become unsatisfiable, for  $0 \leq m \leq 80$ .

The variance of  $P$ , namely  $E(P - EP)^2 = (EP^2) - (EP)^2$ , also has a simple expression in terms of the  $q$ 's, because

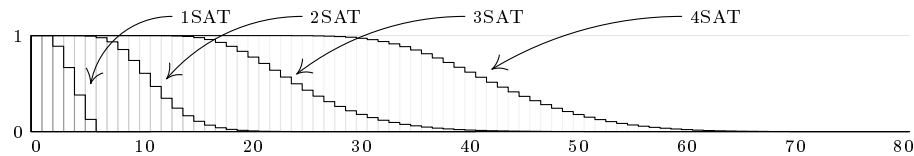
$$EP^2 = \sum_m (2m + 1)q_m. \quad (75)$$

In Figs. 40 and 43 we have  $EP \approx 25.22$ , with variance  $\approx 35.73$ .

So far we've been focusing our attention on 3SAT problems, but the same ideas apply also to  $k$ SAT for other clause sizes  $k$ . Figure 44 shows exact results for the probabilities when  $n = 5$  and  $1 \leq k \leq 4$ . Larger values of  $k$  give clauses that are easier to satisfy, so they increase the stopping time. With five variables the typical stopping times for random 1SAT, 2SAT, 3SAT, and 4SAT turn out to be respectively  $4.06 \pm 1.19$ ,  $11.60 \pm 3.04$ ,  $25.22 \pm 5.98$ , and  $43.39 \pm 7.62$ . In general if  $P_{k,n}$  is the stopping time for  $k$ SAT on  $n$  variables, we let

$$S_{k,n} = EP_{k,n} \quad (76)$$

be its expected value.

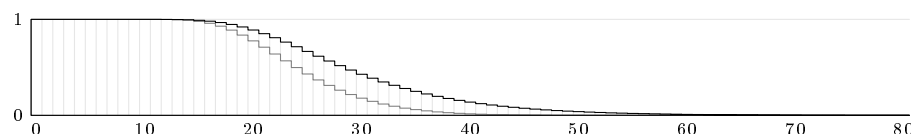


**Fig. 44.** Extension of Fig. 40 to clauses of other sizes.

Our discussions so far have been limited in another way too: We've been assuming that  $m$  *distinct* clauses are being presented to a SAT solver for solution. In practice, however, it's much easier to generate clauses by allowing repetitions, so that every clause is chosen without any dependence on the past history. In other words, there's a more natural way to approach random satisfiability, by assuming that  $N^m$  possible *ordered sequences* of clauses are equally likely after  $m$  steps, not that we have  $\binom{N}{m}$  equally likely *sets* of clauses.

Let  $\hat{q}_m$  be the probability that  $m$  random clauses  $C_1 \wedge \dots \wedge C_m$  are satisfiable, where each  $C_j$  is randomly chosen from among the  $N = 2^k \binom{n}{k}$  possibilities in a  $k$ SAT problem on  $n$  variables. Figure 45 illustrates these probabilities in the case  $k = 3$ ,  $n = 5$ ; notice that we always have  $\hat{q}_m \geq q_m$ . If  $N$  is large while  $m$  is small, it's clear that  $\hat{q}_m$  will be very close to  $q_m$ , because repeated clauses are unlikely in such a case. Still, we must keep in mind that  $q_N$  is always zero, while  $\hat{q}_m$  is *never* zero. Furthermore, the "birthday paradox" discussed in Section 6.4 warns

variance  
kSAT  
4SAT  
2SAT  
1SAT  
sampling with and without replacement  
repeated clauses  
birthday paradox



**Fig. 45.** Random 3SAT on 5 variables when the clauses are sampled with replacement. The probabilities  $\hat{q}_m$  are shown with a black line; the smaller probabilities  $q_m$  of Fig. 40 are shown in gray.

us that repetitions aren't as rare as we might expect. The deviations of  $\hat{q}_m$  from  $q_m$  are particularly noticeable in small cases such as the scenario of Fig. 45.

In any event, there's a direct way to compute  $\hat{q}_m$  from the probabilities  $q_t$  and the value of  $N$  (see exercise 184):

$$\hat{q}_m = \sum_{t=0}^N \binom{m}{t} q_t \binom{N}{t} / N^m. \quad (77)$$

And there are surprisingly simple formulas analogous to (74) and (75) for the stopping time  $\hat{P}$ , where  $\hat{p}_m = \hat{q}_{m-1} - \hat{q}_m$ , as shown in exercise 186:

$$\mathbb{E} \hat{P} = \sum_{m=0}^{N-1} \frac{N}{N-m} q_m; \quad (78)$$

$$\mathbb{E} \hat{P}^2 = \sum_{m=0}^{N-1} \frac{N}{N-m} q_m \left( 1 + 2 \left( \frac{N}{N-1} + \cdots + \frac{N}{N-m} \right) \right). \quad (79)$$

These formulas prove that the expected behavior of  $\hat{P}$  is very much like that of  $P$ , if  $q_m$  is small whenever  $m/N$  isn't small. In the case  $k=3$  and  $n=5$ , the typical stopping times  $\hat{P} = 30.58 \pm 9.56$  are significantly larger than those of  $P$ ; but we are mostly interested in cases where  $n$  is large and where  $\hat{q}_m$  is essentially indistinguishable from  $q_m$ . In order to indicate plainly that the probability  $\hat{q}_m$  depends on  $k$  and  $n$  as well as on  $m$ , we shall denote it henceforth by  $S_k(m, n)$ :

$$S_k(m, n) = \Pr(m \text{ random clauses of } k\text{SAT are satisfiable}), \quad (80)$$

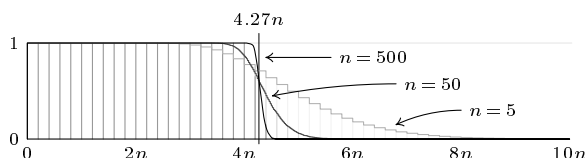
where the  $m$  clauses are "sampled with replacement" (they needn't be distinct). Suitable pseudorandom clauses  $\text{rand}(k, m, n, \text{seed})$  can easily be generated.

Exact formulas appear to be out of reach when  $n > 5$ , but we can make empirical tests. For example, extensive experiments on random 3SAT problems by B. Selman, D. G. Mitchell, and H. J. Levesque [*Artificial Intelligence* 81 (1996), 17–29] showed a dramatic drop in the chances of satisfiability when the number of clauses exceeds about  $4.27n$ . This "phase transition" becomes much sharper as  $n$  grows (see Fig. 46).

Similar behavior occurs for random  $k$ SAT, and this phenomenon has spawned an enormous amount of research aimed at evaluating the so-called *satisfiability thresholds*

$$\alpha_k = \lim_{n \rightarrow \infty} S_{k,n}/n. \quad (81)$$

kSAT  
sampled with replacement  
*rand*  
Selman  
Mitchell  
Levesque  
phase transition  
density of clauses: the number of clauses per  
crossover point, see threshold  
satisfiability thresholds



**Fig. 46.** Empirical probability data shows that random 3SAT problems rapidly become unsatisfiable when there are more than  $\alpha_3 n$  clauses, if  $n$  is large enough.

Indeed, we can obtain quite difficult  $k$ SAT problems by generating approximately  $\alpha_k n$  random  $k$ -clauses, using empirically observed estimates of  $\alpha_k$ . If  $n$  is large, the running time for random 3SAT with  $4.3n$  clauses is typically orders of magnitude larger than it is when the number of clauses is  $4n$  or  $4.6n$ . (And still tougher problems arise in rare instances when we have, say,  $3.9n$  clauses that happen to be *unsatisfiable*.)

Strictly speaking, however, nobody has been able to prove that the so-called constants  $\alpha_k$  actually exist, for all  $k$ ! The empirical evidence is overwhelming; but rigorous proofs for  $k = 3$  have so far only established the bounds

$$\liminf_{n \rightarrow \infty} S_{3,n}/n \geq 3.52; \quad \limsup_{n \rightarrow \infty} S_{3,n}/n \leq 4.49. \quad (82)$$

[See A. C. Kaporis, L. M. Kirousis, and E. G. Lalas, *Random Structures & Algorithms* **28** (2006), 444–480; J. Díaz, L. Kirousis, D. Mitsche, and X. Pérez-Giménez, *Theoretical Comp. Sci.* **410** (2009), 2920–2934.] A “sharp threshold” result has been established by E. Friedgut [*J. Amer. Math. Soc.* **12** (1999), 1017–1045, 1053–1054], who proved the existence for  $k \geq 2$  of functions  $\alpha_k(n)$  with

$$\lim_{n \rightarrow \infty} S_k(\lfloor (\alpha_k(n) - \epsilon)n \rfloor, n) = 1, \quad \lim_{n \rightarrow \infty} S_k(\lfloor (\alpha_k(n) + \epsilon)n \rfloor, n) = 0, \quad (83)$$

when  $\epsilon$  is any positive number. But those functions might not approach a limit. They might, for example, fluctuate periodically, like the “wobble function” that we encountered in Eq. 5.2.2–(47).

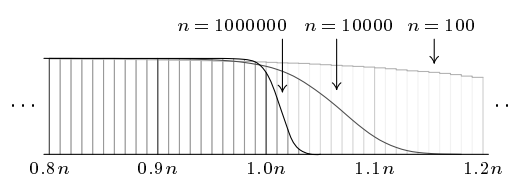
The current best guess for  $\alpha_3$ , based on heuristics of the “survey propagation” technique to be discussed below, is that  $\alpha_3 = 4.26675 \pm 0.00015$  [S. Mertens, M. Mézard, and R. Zecchina, *Random Structures & Algorithms* **28** (2006), 340–373]. Similarly, it appears reasonable to believe that  $\alpha_4 \approx 9.931$ ,  $\alpha_5 \approx 21.12$ ,  $\alpha_6 \approx 43.37$ ,  $\alpha_7 \approx 87.79$ . The  $\alpha$ ’s grow as  $\Theta(2^k)$  (see exercise 195); and they *are* known to be constant when  $k$  is sufficiently large [see J. Ding, A. Sly, and N. Sun, *STOC* **47** (2015), to appear].

**Analysis of random 2SAT.** Although nobody knows how to prove that random 3SAT problems almost always become unsatisfiable when the number of clauses reaches  $\approx 4.27n$ , the corresponding question for 2SAT does have a nice answer: *The satisfiability threshold  $\alpha_2$  equals 1.* For example, when the author first tried 1000 random 2SAT problems with a million variables, 999 of them turned out to be satisfiable when there were 960,000 clauses, while all were unsatisfiable when the number of clauses rose to 1,040,000. Figure 47 shows how this transition becomes sharper as  $n$  increases.

Kaporis  
Kirousis  
Lalas  
Díaz  
Mitsche  
Pérez-Giménez  
sharp threshold  
Friedgut  
wobble function  
survey propagation  
Mertens  
Mézard  
Zecchina  
4SAT  
5SAT  
6SAT  
7SAT  
kSAT  
Ding  
Sly  
Sun  
2SAT-



**Fig. 47.** Empirical satisfaction probabilities for 2SAT with approximately  $n$  random clauses. (When  $n = 100$ , the probability doesn't become negligible until more than roughly 180 clauses have been generated.)



Chvátal  
Reed  
Goerdts  
Fernandez de la Vega  
implication digraph  
Karp  
giant strong component  
strong components  
Chvátal  
Reed  
 $s$ -chain  
strictly distinct literals

The fact that  $S_{2,n} \approx n$  was discovered in 1991 by V. Chvátal and B. Reed [FOCS **33** (1992), 620–627], and the same result was obtained independently at about the same time by A. Goerdts and by W. Fernandez de la Vega [see *J. Comp. Syst. Sci.* **53** (1996), 469–486; *Theor. Comp. Sci.* **265** (2001), 131–146].

The study of this phenomenon is instructive, because it relies on properties of the digraph that characterizes all instances of 2SAT. Furthermore, the proof below provides an excellent illustration of the “first and second moment principles,” equations MPR-(21) and MPR-(22). Armed with those principles, we’re ready to derive the 2SAT threshold:

**Theorem C.** *Let  $c$  be a fixed constant. Then*

$$\lim_{n \rightarrow \infty} S_2(\lfloor cn \rfloor, n) = \begin{cases} 1, & \text{if } c < 1; \\ 0, & \text{if } c > 1. \end{cases} \quad (84)$$

*Proof.* Every 2SAT problem corresponds to an *implication digraph* on the literals, with arcs  $\bar{l} \rightarrow l'$  and  $\bar{l}' \rightarrow l$  for each clause  $l \vee l'$ . We know from Theorem 7.1.1K that a set of 2SAT clauses is satisfiable if and only if no strong component of its implication digraph contains both  $x$  and  $\bar{x}$  for some variable  $x$ . That digraph has  $2m = 2\lfloor cn \rfloor$  arcs and  $2n$  vertices. If it were a random digraph, well-known theorems of Karp (which we shall study in Section 7.4.1) would imply that only  $O(\log n)$  vertices are reachable from any given vertex when  $c < 1$ , but that there is a unique “giant strong component” of size  $\Omega(n)$  when  $c > 1$ .

The digraph that arises from random 2SAT isn’t truly random, because its arcs come in pairs,  $u \rightarrow v$  and  $\bar{v} \rightarrow \bar{u}$ . But intuitively we can expect that similar behavior will apply to digraphs that are just halfway random. For example, when the author generated a random 2SAT problem with  $n = 1000000$  and  $m = .99n$ , the resulting digraph had only two complementary pairs of strong components with more than one vertex, and their sizes were only 2, 2 and 7, 7; so the clauses were easily satisfiable. Adding another  $.01n$  clauses didn’t increase the number of nontrivial strong components, and the problem remained satisfiable. But another experiment with  $m = n = 1000000$  yielded a strong component of size 420, containing 210 variables and their complements; that problem was unsatisfiable.

Based on a similar intuition into the underlying structure, Chvátal and Reed introduced the following “snares and snakes” approach to the proof of Theorem C: Let’s say that an *s-chain* is any sequence of  $s$  strictly distinct literals; thus there are  $2^s n^s$  possible  $s$ -chains. Every  $s$ -chain  $C$  corresponds to clauses

$$(\bar{l}_1 \vee l_2), (\bar{l}_2 \vee l_3), \dots, (\bar{l}_{s-1} \vee l_s), \quad (85)$$

which in turn correspond to two paths

$$l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow \dots \rightarrow l_s \quad \text{and} \quad \bar{l}_s \rightarrow \dots \rightarrow \bar{l}_3 \rightarrow \bar{l}_2 \rightarrow \bar{l}_1 \quad (86)$$

in the digraph. An  $s$ -snare  $(C; t, u)$  consists of an  $s$ -chain  $C$  and two indices  $t$  and  $u$ , where  $1 < t \leq s$  and  $1 \leq |u| < s$ ; it specifies the clauses (85) together with

$$(l_t \vee l_1) \quad \text{and} \quad (\bar{l}_s \vee l_u) \text{ if } u > 0, \quad (\bar{l}_s \vee \bar{l}_{-u}) \text{ if } u < 0, \quad (87)$$

representing  $\bar{l}_t \rightarrow l_1$  and either  $l_s \rightarrow l_{|u|}$  or  $l_s \rightarrow \bar{l}_{|u|}$ . The number of possible  $s$ -snares is  $2^{s+1}(s-1)^2n^s$ . Their clauses are rarely all present when  $m$  is small.

Exercise 200 explains how to use these definitions to prove Theorem C in the case  $c < 1$ . First we show that every unsatisfiable 2SAT formula contains all the clauses of at least one snare. Then, if we define the binary random variable

$$X(C; t, u) = [\text{all clauses of } (C; t, u) \text{ are present}], \quad (88)$$

it isn't difficult to prove that the snares of every  $s$ -chain  $C$  are unlikely:

$$EX(C; t, u) \leq m^{s+1}/(2n(n-1))^{s+1}. \quad (89)$$

Finally, letting  $X$  be the sum of  $X(C; t, u)$  over all snares, we obtain

$$EX = \sum_{s \geq 0} EX(C; t, u) \leq \sum_{s \geq 0} 2^{s+1}s(s-1)n^s \left(\frac{m}{2n(n-1)}\right)^{s+1} = \frac{2}{n} \left(\frac{m}{n-1-m}\right)^3$$

by Eq. 1.2.9-(20). This formula actually establishes a stronger form of (84), because it shows that  $EX$  is only  $O(n^{-1/4})$  when  $m = n - n^{3/4} > cn$ . Thus

$$S_2(\lfloor n - n^{3/4} \rfloor, n) \geq \Pr(X = 0) = 1 - \Pr(X > 0) \geq 1 - O(n^{-1/4}) \quad (90)$$

by the first moment principle.

The other half of Theorem C can be proved by using the concept of a  $t$ -snake, which is the special case  $(C; t, -t)$  of a  $(2t-1)$ -snare. In other words, given any chain  $(l_1, \dots, l_t, \dots, l_{2t-1})$ , with  $s = 2t-1$  and  $l_t$  in the middle, a  $t$ -snake generates the clauses (85) together with  $(l_t \vee l_1)$  and  $(\bar{l}_s \vee \bar{l}_t)$ . When  $t = 5$ , for example, and  $(l_1, \dots, l_{2t-1}) = (x_1, \dots, x_9)$ , the  $2t = 10$  clauses are

$$51, \bar{1}2, \bar{2}3, \bar{3}4, \bar{4}5, \bar{5}6, \bar{6}7, \bar{7}8, \bar{8}9, \bar{9}\bar{5},$$

and they correspond to 20 arcs that loop around to form a strong component as shown here. We will prove that, when  $c > 1$  in (84), the digraph almost always contains such impediments to satisfiability.

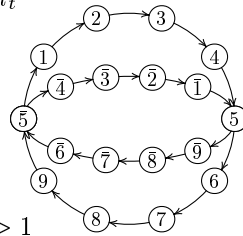
Given a  $(2t-1)$ -chain  $C$ , where the parameter  $t$  will be chosen later, let

$$X_C = [\text{each clause of } (C; t, -t) \text{ occurs exactly once}]. \quad (91)$$

The expected value  $EX_C$  is clearly  $f(2t)$ , where

$$f(r) = m^r(2n(n-1) - r)^{m-r}/(2n(n-1))^m \quad (92)$$

$s$ -snare  
first moment principle  
 $t$ -snake



is the probability that  $r$  specific clauses occur once each. Notice that

$$f(r) = \left(\frac{m}{2n(n-1)}\right)^r \left(1 + O\left(\frac{r^2}{m}\right) + O\left(\frac{rm}{n^2}\right)\right); \tag{93}$$

thus the relative error will be  $O(t^2/n)$  if  $m = \Theta(n)$  as  $n \rightarrow \infty$ .

Now let  $X = \sum X_C$ , summed over all  $R = 2^{2t-1}n^{2t-1}$  possible  $t$ -snakes  $C$ ; thus  $\mathbb{E} X = Rf(2t)$ . We want to show that  $\Pr(X > 0)$  is very nearly 1, using the second moment principle; so we want to show that the expectation  $\mathbb{E} X^2 = \mathbb{E}(\sum_C X_C)(\sum_D X_D) = \sum_C \sum_D \mathbb{E} X_C X_D$  is small. The key observation is that

$$\mathbb{E} X_C X_D = f(4t - r) \quad \text{if } C \text{ and } D \text{ have exactly } r \text{ clauses in common.} \tag{94}$$

Let  $p_r$  be the probability that a randomly chosen  $t$ -snake has exactly  $r$  clauses in common with the fixed snake  $(x_1, \dots, x_{2t-1})$ . Then

$$\begin{aligned} \frac{\mathbb{E} X^2}{(\mathbb{E} X)^2} &= \frac{R^2 \sum_{r=0}^{2t} p_r f(4t - r)}{R^2 f(2t)^2} \\ &= \sum_{r=0}^{2t} p_r \frac{f(4t - r)}{f(2t)^2} = \sum_{r=0}^{2t} p_r \left(\frac{2n(n-1)}{m}\right)^r \left(1 + O\left(\frac{t^2}{n}\right)\right). \end{aligned} \tag{95}$$

By studying the interaction of snakes (see exercise 201) one can prove that

$$(2n)^r p_r = O(t^4/n) + O(t)[r \geq t] + O(n)[r = 2t], \quad \text{for } 1 \leq r \leq 2t. \tag{96}$$

Finally then, as explained in exercise 202, we can choose  $t = \lfloor n^{1/5} \rfloor$  and  $m = \lfloor n + n^{5/6} \rfloor$ , to deduce a sharper form of (84) when  $c > 1$ :

$$S_2(\lfloor n + n^{5/6} \rfloor, n) = O(n^{-1/30}). \tag{97}$$

(Deep breath.) Theorem C is proved. ■

Much more precise results have been derived by B. Bollobás, C. Borgs, J. T. Chayes, J. H. Kim, and D. B. Wilson, in *Random Structures & Algorithms* **18** (2001), 201–256. For example, they showed that

$$S_2(\lfloor n - n^{3/4} \rfloor, n) = \exp(-\Theta(n^{-1/4})); \quad S_2(\lfloor n + n^{3/4} \rfloor, n) = \exp(-\Theta(n^{1/4})). \tag{98}$$

**Resolution.** The backtracking process of Algorithms A, B, D, and L is closely connected to a logical proof procedure called *resolution*. Starting with a family of clauses called “axioms,” there’s a simple rule by which new clauses can be derived from this given set: Whenever both  $x \vee A'$  and  $\bar{x} \vee A''$  are in our repertoire of clauses, we’re allowed to derive the “resolvent” clause  $A = A' \vee A''$ , denoted by  $(x \vee A') \diamond (\bar{x} \vee A'')$ . (See exercises 218 and 219.)

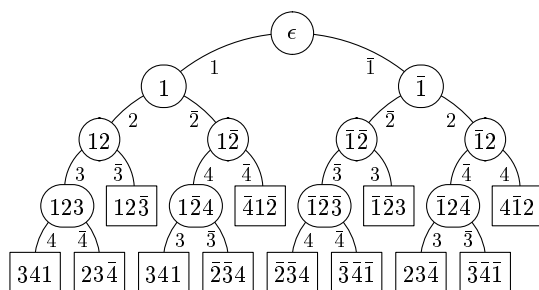
A *proof by resolution* consists of a directed acyclic graph (dag) whose vertices are labeled with clauses in the following way: (i) Every source vertex is labeled with an axiom. (ii) Every other vertex has in-degree 2. (iii) If the predecessors of vertex  $v$  are  $v'$  and  $v''$ , the label of  $v$  is  $C(v) = C(v') \diamond C(v'')$ .

When such a dag has a sink vertex labeled  $A$ , we call it a “resolution proof of  $A$ ”; and if  $A$  is the empty clause, the dag is also called a “resolution refutation.”

second moment principle  
 Bollobás  
 Borgs  
 Chayes  
 Kim  
 Wilson  
 resolution-  
 axioms  
 Notation  $C' \diamond C''$   
 directed acyclic graph  
 dag  
 refutation

The dag of a proof by resolution can be expanded to a binary tree, by replicating any vertex that has out-degree greater than 1. Such a tree is said to be *regular* if no path from the root to a leaf uses the same variable twice to form a resolvent. For example, Fig. 48 is a regular resolution tree that refutes Rivest's unsatisfiable axioms (6). All arcs in this tree are directed upwards.

regular resolution  
treelike resolution  
Rivest  
unnecessary branch  
lookahead  
Impagliazzo  
Pudlák  
Prover–Delayer game



**Fig. 48.** One way to derive  $\epsilon$  by resolving the inconsistent clauses (6).

Notice that Fig. 48 is essentially identical to Fig. 39 on page 33, the backtrack tree by which Algorithm D discovers that the clauses of (6) are unsatisfiable. In fact this similarity is no coincidence: *Every backtrack tree that records the behavior of Algorithm D on a set of unsatisfiable clauses corresponds to a regular resolution tree that refutes those axioms, unless Algorithm D makes an unnecessary branch.* (An unnecessary branch occurs if the algorithm tries  $x \leftarrow 0$  and  $x \leftarrow 1$  without using their consequences to discover an unsatisfiable subset of axioms.) Conversely, every regular refutation tree corresponds to a sequence of choices by which a backtrack-based SAT solver could prove unsatisfiability.

The reason behind this correspondence isn't hard to see. Suppose both values of  $x$  need to be tried in order to prove unsatisfiability. When we set  $x \leftarrow 0$  in one branch of the backtrack tree, we replace the original clauses  $F$  by  $F \mid \bar{x}$ , as in (54). The key point is that *we can prove the empty clause by resolution from  $F \mid \bar{x}$  if and only if we can prove  $x$  by resolution from  $F$  without resolving on  $x$ .* (See exercise 224.) Similarly, setting  $x \leftarrow 1$  corresponds to changing the clauses from  $F$  to  $F \mid x$ .

Consequently, if  $F$  is an inconsistent set of clauses that has no short refutation tree, Algorithm D cannot conclude that those clauses are unsatisfiable unless it runs for a long time. Neither can Algorithm L, in spite of enhanced lookahead.

R. Impagliazzo and P. Pudlák [SODA 11 (2000), 128–136] have introduced an appealing *Prover–Delayer game*, with which it's relatively easy to demonstrate that certain sets of unsatisfiable clauses require large refutation trees. The Prover names a variable  $x$ , and the Delayer responds by saying either  $x \leftarrow 0$  or  $x \leftarrow 1$  or  $x \leftarrow *$ . In the latter case the Prover gets to decide the value of  $x$ ; but the Delayer scores one point. The game ends when the current assignments have falsified at least one clause. If the Delayer has a strategy that guarantees a score of at least  $m$  points, exercise 226 shows that every refutation tree has at least  $2^m$

leaves; hence at least  $2^m - 1$  resolutions must be done, and every backtrack-based solver needs  $\Omega(2^m)$  operations to declare the clauses unsatisfiable.

We can apply their game, for example, to the following interesting clauses:

$$(\bar{x}_{jj}), \quad \text{for } 1 \leq j \leq m; \quad (99)$$

$$(\bar{x}_{ij} \vee \bar{x}_{jk} \vee x_{ik}), \quad \text{for } 1 \leq i, j, k \leq m; \quad (100)$$

$$(x_{j1} \vee x_{j2} \vee \cdots \vee x_{jm}), \quad \text{for } 1 \leq j \leq m. \quad (101)$$

There are  $m^2$  variables  $x_{jk}$ , for  $1 \leq j, k \leq m$ , which we can regard as the incidence matrix for a binary relation ' $j \prec k$ '. With this formulation, (99) says that the relation is irreflexive, and (100) says that it's transitive; thus, (99) and (100) amount to saying that  $j \prec k$  is a *partial ordering*. Finally, (101) says that, for every  $j$ , there's a  $k$  with  $j \prec k$ . So these clauses state that there's a partial ordering on  $\{1, \dots, m\}$  in which no element is maximal; and they can't all be satisfied.

We can, however, always score  $m - 1$  points if we're playing Delayer in that game, by using the following strategy suggested by Massimo Lauria: At every step we know an ordered set  $S$  of elements, regarded as "small"; initially  $S = \emptyset$ , and we'll have  $S = \{j_1, \dots, j_s\}$  when our score is  $s$ . Suppose the Prover queries  $x_{jk}$ , and  $s < m - 2$ . If  $j = k$ , we naturally reply that  $x_{jk} \leftarrow 0$ . Otherwise, if  $j \notin S$  and  $k \notin S$ , we respond  $x_{jk} \leftarrow *$ ; then  $s \leftarrow s + 1$ , and  $j_s \leftarrow j$  or  $k$  according as the Prover specifies  $x_{jk} \leftarrow 1$  or  $x_{jk} \leftarrow 0$ . Otherwise, if  $j \in S$  and  $k \notin S$ , we respond  $x_{jk} \leftarrow 1$ ; if  $j \notin S$  and  $k \in S$ , we respond  $x_{jk} \leftarrow 0$ . Finally, if  $j = j_a \in S$  and  $k = j_b \in S$ , we respond  $x_{jk} \leftarrow [a < b]$ . These responses always satisfy (99) and (100). And no clause of (101) becomes false before the Delayer is asked a question with  $s = m - 2$ . Then the response  $x_{jk} \leftarrow *$  gains another point. We've proved

**Theorem R.** *Every refutation tree for the clauses (99), (100), (101) represents at least  $2^{m-1} - 1$  resolution steps. ■*

On the other hand, those clauses do have a refutation *dag* of size  $O(m^3)$ . Let  $I_j$  and  $T_{ijk}$  stand for the irreflexivity and transitivity axioms (99) and (100); and let  $M_{jk} = x_{j1} \vee \cdots \vee x_{jk}$ , so that (101) is  $M_{jm}$ . Then we have

$$M_{im} \diamond T_{imk} = M_{i(m-1)} \vee \bar{x}_{mk}, \quad \text{for } 1 \leq i, k < m. \quad (102)$$

Calling this new clause  $M'_{imk}$ , we can now derive

$$M_{j(m-1)} = ((\cdots ((M_{mm} \diamond M'_{jm1}) \diamond M'_{jm2}) \diamond \cdots) \diamond M'_{jm(m-1)}) \diamond I_m,$$

for  $1 \leq j < m$ . Hence  $(m - 1)^2 + (m - 1)m$  resolutions have essentially reduced  $m$  to  $m - 1$ . Eventually we can therefore derive  $M_{11}$ ; then  $M_{11} \diamond I_1 = \epsilon$ . [This elegant refutation is due to G. Stålmarck, *Acta Informatica* **33** (1996), 277–280.]

The method we've just used to obtain  $M_{j(m-1)}$  from  $M_{mm}$  is, incidentally, a special case of a useful general formula called *hyperresolution* that is easily proved by induction on  $r$ :

$$\begin{aligned} & (\cdots ((C_0 \vee x_1 \vee \cdots \vee x_r) \diamond (C_1 \vee \bar{x}_1)) \diamond \cdots) \diamond (C_r \vee \bar{x}_r) \\ & = C_0 \vee C_1 \vee \cdots \vee C_r. \end{aligned} \quad (103)$$

anti-maximal-element clauses  
binary relation  
irreflexive  
transitive  
partial ordering  
maximal  
Lauria  
Stålmarck  
hyperresolution

**\*Lower bounds for general resolution.** Let's change our perspective slightly: Instead of visualizing a proof by resolution as a directed graph, we can think of it as a "straight line" *resolution chain*, analogous to the addition chains of Section 4.6.3 and the Boolean chains of Section 7.1.2. A resolution chain based on  $m$  axioms  $C_1, \dots, C_m$  appends additional clauses  $C_{m+1}, \dots, C_{m+r}$ , each of which is obtained by resolving two previous clauses of the chain. Formally, we have

$$C_i = C_{j(i)} \diamond C_{k(i)}, \quad \text{for } m+1 \leq i \leq m+r, \quad (104)$$

where  $1 \leq j(i) < i$  and  $1 \leq k(i) < i$ . It's a *refutation chain* for  $C_1, \dots, C_m$  if  $C_{m+r} = \epsilon$ . The tree in Fig. 48, for example, yields the refutation chain

$$12\bar{3}, 23\bar{4}, 34\bar{1}, 4\bar{1}2, \bar{1}\bar{2}\bar{3}, \bar{2}\bar{3}\bar{4}, \bar{3}\bar{4}\bar{1}, \bar{4}\bar{1}\bar{2}, 123, 1\bar{2}4, \bar{1}\bar{2}\bar{3}, \bar{1}\bar{2}\bar{4}, 12, \bar{1}\bar{2}, \bar{1}\bar{2}, \bar{1}\bar{2}, 1, \bar{1}, \epsilon$$

for the axioms (6); and there are many other ways to refute those axioms, such as

$$12\bar{3}, 23\bar{4}, 34\bar{1}, 4\bar{1}2, \bar{1}\bar{2}\bar{3}, \bar{2}\bar{3}\bar{4}, \bar{3}\bar{4}\bar{1}, \bar{4}\bar{1}\bar{2}, 1\bar{2}\bar{3}, 1\bar{3}, 14, \bar{3}\bar{4}, 24, 2\bar{4}, 2, \bar{1}\bar{3}, \bar{3}\bar{4}, 1\bar{4}, \bar{3}, 1, \bar{1}, \epsilon. \quad (105)$$

This chain is quite different from Fig. 48, and perhaps nicer: It has three more steps, but after forming ' $\bar{1}\bar{2}\bar{3}$ ' it constructs only very short clauses.

We'll see in a moment that short clauses are crucial if we want short chains. That fact turns out to be important when we try to prove that certain easily understood families of axioms are inherently more difficult than (99), (100), and (101), in the sense that they can't be refuted with a chain of polynomial size.

Consider, for example, the well known "pigeonhole principle," which states that  $m+1$  pigeons don't fit in  $m$  pigeon-sized holes. If  $x_{jk}$  means that pigeon  $j$  occupies hole  $k$ , for  $0 \leq j \leq m$  and  $1 \leq k \leq m$ , the relevant unsatisfiable clauses are

$$(x_{j1} \vee x_{j2} \vee \dots \vee x_{jm}), \quad \text{for } 0 \leq j \leq m; \quad (106)$$

$$(\bar{x}_{ik} \vee \bar{x}_{jk}), \quad \text{for } 0 \leq i < j \leq m \text{ and } 1 \leq k \leq m. \quad (107)$$

("Every pigeon has a hole, but no hole hosts more than one pigeon.") These clauses increased the pigeonhole principle's fame during the 1980s, when Armin Haken [*Theoretical Computer Science* **39** (1985), 297–308] proved that they have no short refutation chain. His result marked the first time that *any* set of clauses had been shown to be intractable for resolution in general.

*It is absolutely necessary that two people have equally many hairs.*

— JEAN APPIER HANZELET, *Recreation Mathematicque* (1624)

Haken's original proof was rather complicated. But simpler approaches were eventually found, culminating in a method by E. Ben-Sasson and A. Wigderson [*JACM* **48** (2001), 149–169], which is based on clause length and applies to many other sets of axioms. If  $\alpha$  is any sequence of clauses, let us say that its *width*, written  $w(\alpha)$ , is the length of its longest clause or clauses. Furthermore, if  $\alpha_0 = (C_1, \dots, C_m)$ , we write  $w(\alpha_0 \vdash \epsilon)$  for the minimum of  $w(\alpha)$  over all refutation chains  $\alpha = (C_1, \dots, C_{m+r})$  for  $\alpha_0$ , and  $\|\alpha_0 \vdash \epsilon\|$  for the minimum length  $r$  of all such chains. The following lemma is the key to proving lower bounds with Ben-Sasson and Wigderson's strategy:

resolution chain  
refutation chain  
pigeonhole principle  
Haken  
APPIER HANZELET  
Ben-Sasson  
Wigderson  
width  
notation  $w(\alpha)$   
notation  $w(\alpha \vdash \epsilon)$   
notation  $\|\alpha \vdash \epsilon\|$

**Lemma B.**  $\|\alpha_0 \vdash \epsilon\| \geq e^{(w(\alpha_0 \vdash \epsilon)-1)^2/(8n)} - 2$ , for clauses in  $n \geq w(\alpha_0)^2$  variables. Thus there's exponential growth if we have  $w(\alpha_0) = O(1)$  and  $w(\alpha_0 \vdash \epsilon) = \Omega(n)$ .

*Proof.* Let  $\alpha = (C_1, \dots, C_{m+r})$  be a refutation of  $\alpha_0$  with  $r = \|\alpha_0 \vdash \epsilon\|$ . We will say that a clause is “fat” if its length is  $W$  or more, where  $W \geq w(\alpha_0)$  is a parameter to be set later. If  $\alpha \setminus \alpha_0$  contains  $f$  fat clauses, those clauses contain at least  $Wf$  literals; hence some literal  $l$  appears in at least  $Wf/(2n)$  of them.

Now  $\alpha \mid l$ , the chain obtained by replacing each clause  $C_j$  by  $C_j \mid l$ , is a refutation of  $\alpha_0 \mid l$  that contains at most  $\lfloor \rho f \rfloor$  fat clauses, where  $\rho = 1 - W/(2n)$ . (The clause  $C_j \mid l$  will be  $\varnothing$  if  $l \in C_j$ , thus tautological and effectively absent.)

Suppose  $f < \rho^{-b}$  for some integer  $b$ . We will prove, by induction on  $b$  and secondarily on the total length of all clauses, that there's a refutation  $\beta$  of  $\alpha_0$  such that  $w(\beta) \leq W + b$ . This assertion holds when  $b = 0$ , since  $W \geq w(\alpha_0)$ . If  $b > 0$ , there's a refutation  $\beta_0$  of  $\alpha_0 \mid l$  with  $w(\beta_0) \leq W + b - 1$ , when we choose  $l$  as above, because  $\rho f < \rho^{1-b}$  and  $\alpha \mid l$  refutes  $\alpha_0 \mid l$ . Then we can form a resolution chain  $\beta_1$  that derives  $\bar{l}$  from  $\alpha_0$ , by inserting  $\bar{l}$  appropriately into clauses of  $\beta_0$ . And there's a simple chain  $\beta_2$  that derives the clauses of  $\alpha_0 \mid \bar{l}$  from  $\alpha_0$  and  $\bar{l}$ . There's also a refutation  $\beta_3$  of  $\alpha_0 \mid \bar{l}$  with  $w(\beta_3) \leq W + b$ , by induction, because  $\alpha \mid \bar{l}$  refutes  $\alpha_0 \mid \bar{l}$ . Thus the combination  $\beta = \{\beta_1, \beta_2, \beta_3\}$  refutes  $\alpha_0$ , with

$$w(\beta) = \max(w(\beta_0) + 1, w(\beta_2), w(\beta_3)) \leq \max(W + b, w(\alpha_0), W + b) = W + b.$$

Finally, exercise 238 chooses  $W$  so that we get the claimed bound. ■

The pigeon axioms are too wide to be inserted directly into Lemma B. But Ben-Sasson and Wigderson observed that a simplified version of those axioms, involving only clauses of 5SAT, is already intractable.

Notice that we can regard the variable  $x_{jk}$  as indicating the presence of an edge between  $a_j$  and  $b_k$  in a bipartite graph on the vertices  $A = \{a_0, \dots, a_m\}$  and  $B = \{b_1, \dots, b_m\}$ . Condition (106) says that each  $a_j$  has degree  $\geq 1$ , while condition (107) says that each  $b_k$  has degree  $\leq 1$ . There is, however, a bipartite graph  $G_0$  on those vertices for which each  $a_j$  has degree  $\leq 5$  and such that the following strong “expansion” condition is satisfied:

$$\text{Every subset } A' \subseteq A \text{ with } |A'| \leq m/3000 \text{ has } |\partial A'| \geq |A'| \text{ in } G_0. \quad (108)$$

Here  $\partial A'$  denotes the bipartite boundary of  $A'$ , namely the set of all  $b_k$  that have exactly one neighbor in  $A'$ .

Given such a graph  $G_0$ , whose existence is proved (nonconstructively) in exercise 240, we can formulate a *restricted pigeonhole principle*, by which the pigeonhole clauses are unsatisfiable if we also require  $\bar{x}_{jk}$  whenever  $a_j \dashv b_k$  in  $G_0$ .

Let  $\alpha(G_0)$  denote the resulting clauses, which are obtained when axioms (106) and (107) are conditioned on all such literals  $\bar{x}_{jk}$ . Then  $w(\alpha(G_0)) \leq 5$ , and at most  $5m + 5$  unspecified variables  $x_{jk}$  remain. Lemma B tells us that all refutation chains for  $\alpha(G_0)$  have length  $\exp \Omega(m)$  if we can prove that they all have width  $\Omega(m)$ . Haken's theorem, which asserts that all refutation chains for (106) and (107) also have length  $\exp \Omega(m)$ , will follow, because any short refutation would yield a short refutation of  $\alpha(G_0)$  after conditioning on the  $\bar{x}_{jk}$ .

fat  
tautological  
Ben-Sasson  
Wigderson  
5SAT  
bipartite graph  
expansion  
boundary  
restricted pigeonhole principle  
Haken

Thus the following result gives our story a happy ending:

**Theorem B.** *The restricted pigeonhole axioms  $\alpha(G_0)$  have refutation width*

$$w(\alpha(G_0) \vdash \epsilon) \geq m/6000. \quad (109)$$

*Proof.* We can assign a complexity measure to every clause  $C$  by defining

$$\mu(C) = \min\{|A'| \mid A' \subseteq A \text{ and } \alpha(A') \vdash C\}. \quad (110)$$

Here  $\alpha(A')$  is the set of “pigeon axioms” (106) for  $a_j \in A'$ , together with all of the “hole axioms” (107); and  $\alpha(A') \vdash C$  means that clause  $C$  can be proved by resolution when starting with only those axioms. If  $C$  is one of the pigeon axioms, this definition makes  $\mu(C) = 1$ , because we can let  $A' = \{a_j\}$ . And if  $C$  is a hole axiom, clearly  $\mu(C) = 0$ . The subadditive law

$$\mu(C' \diamond C'') \leq \mu(C') + \mu(C'') \quad (111)$$

also holds, because a proof of  $C' \diamond C''$  needs at most the axioms of  $\alpha(A') \cup \alpha(A'')$  if  $C'$  follows from  $\alpha(A')$  and  $C''$  follows from  $\alpha(A'')$ .

We can assume that  $m \geq 6000$ . And we must have  $\mu(\epsilon) > m/3000$ , because of the strong expansion condition (108). (See exercise 241.) Therefore every refutation of  $\alpha(G_0)$  must contain a clause  $C$  with  $m/6000 \leq \mu(C) < m/3000$ ; indeed, the first clause  $C_j$  with  $\mu(C_j) \geq m/6000$  will satisfy this condition, by (111).

Let  $A'$  be a set of vertices with  $|A'| = \mu(C)$  and  $\alpha(A') \vdash C$ . Also let  $b_k$  be any element of  $\partial A'$ , with  $a_j$  its unique neighbor in  $A'$ . Since  $|A' \setminus a_j| < \mu(C)$ , there must be an assignment of variables that satisfies all axioms of  $\alpha(A' \setminus a_j)$ , but falsifies  $C$  and the pigeon axiom for  $j$ . That assignment puts no two pigeons into the same hole, and it places every pigeon of  $A' \setminus a_j$ .

Now suppose  $C$  contains no literal of the form  $x_{j'k}$  or  $\bar{x}_{j'k}$ , for any  $a_{j'} \in A$ . Then we could set  $x_{j'k} \leftarrow 0$  for all  $j'$ , without falsifying any axiom of  $\alpha(A' \setminus a_j)$ ; and we could then make the axioms of  $\alpha(\{a_j\})$  true by setting  $x_{jk} \leftarrow 1$ . But that change to the assignment would leave  $C$  false, contradicting our assumption that  $\alpha(A') \vdash C$ . Thus  $C$  contains some  $\pm x_{j'k}$  for each  $b_k \in \partial A'$ ; and we must have  $w(C) \geq |\partial A'| \geq m/6000$ . ■

A similar proof establishes a linear lower bound on the refutation width, hence an exponential lower bound on the refutation length, of almost all random 3SAT instances with  $n$  variables and  $\lfloor \alpha n \rfloor$  clauses, for fixed  $\alpha$  as  $n \rightarrow \infty$  (see exercise 243), a theorem of V. Chvátal and E. Szemerédi [JACM **35** (1988), 759–768].

*Historical notes:* Proofs by resolution, in the more general setting of first order logic, were introduced by J. A. Robinson in JACM **12** (1965), 23–41. [They’re also equivalent to G. Gentzen’s “cut rule for sequents,” *Mathematische Zeitschrift* **39** (1935), 176–210, III.1.2.1.] Inspired by Robinson’s paper, Gregory Tseytin developed the first nontrivial techniques to prove lower bounds on the length of resolution proofs, based on unsatisfiable graph axioms that are considered in exercise 245. His lectures of 1966 were published in Volume 8 of the Steklov Mathematical Institute Seminars in Mathematics (1968); see A. O. Slisenko’s English translation, *Studies in Constructive Mathematics and Mathematical Logic*, part 2 (1970), 115–125.

notation  $F \vdash C$   
notation  $\mu(C)$   
subadditive law  
3SAT  
random 3SAT  
Chvátal  
Szemerédi  
first order logic  
Robinson  
Gentzen  
cut rule  
sequents  
Robinson  
Tseytin  
graph-based axioms  
Slisenko



Tseytin pointed out that there's a simple way to get around the lower bounds he had proved for his graph-oriented problems, by allowing new kinds of proof steps: Given any set of axioms  $F$ , we can introduce a new variable  $z$  that doesn't appear anywhere in  $F$ , and add three new clauses  $G = \{xz, yz, \bar{x}\bar{y}\bar{z}\}$ ; here  $x$  and  $y$  are arbitrary literals of  $F$ . It's clear that  $F$  is satisfiable if and only if  $F \cup G$  is satisfiable, because  $G$  essentially says that  $z = \text{NAND}(x, y)$ . Adding new variables in this way is somewhat analogous to using lemmas when proving a theorem, or to introducing a memo cache in a computer program.

His method, which is called *extended resolution*, can be much faster than pure resolution. For example, it allows the pigeonhole clauses (106) and (107) to be refuted in only  $O(m^4)$  steps (see exercise 237). It doesn't appear to help much with certain other classes of problems such as random 3SAT; but who knows?

**SAT solving via resolution.** The concept of resolution also suggests alternative ways to solve satisfiability problems. In the first place we can use it to eliminate variables: If  $F$  is any set of clauses on  $n$  variables, and if  $x$  is one of those variables, we can construct a set  $F'$  of clauses on the other  $n - 1$  variables in such a way that  $F$  is satisfiable if and only if  $F'$  is satisfiable. The idea is simply to resolve every clause of the form  $x \vee A'$  with every clause of the form  $\bar{x} \vee A''$ , and then to discard those clauses.

For example, consider the following six clauses in four variables:

$$1234, \bar{1}\bar{2}\bar{3}, \bar{1}3, 2\bar{3}, 3\bar{4}. \quad (112)$$

We can eliminate the variable  $x_4$  by forming  $1234 \diamond 3\bar{4} = 123$ . Then we can eliminate  $x_3$  by resolving  $123$  and  $\bar{1}3$  with  $\bar{1}\bar{2}\bar{3}$  and  $2\bar{3}$ :

$$123 \diamond \bar{1}\bar{2}\bar{3} = \emptyset, \quad 123 \diamond 2\bar{3} = 12, \quad \bar{1}3 \diamond \bar{1}\bar{2}\bar{3} = \bar{1}\bar{2}, \quad \bar{1}3 \diamond 2\bar{3} = \bar{1}2.$$

Now we're left with  $\{12, \bar{1}\bar{2}, \bar{1}2, \bar{1}\bar{2}\}$ , because the tautology  $\emptyset$  goes away. Eliminating  $x_2$  gives  $\{1, \bar{1}\}$ , and eliminating  $x_1$  gives  $\{\epsilon\}$ ; hence (112) is unsatisfiable.

This method, which was originally proposed for hand calculation by E. W. Samson and R. K. Mueller in 1955, works beautifully on small problems. But why is it valid? There are (at least) two good ways to understand the reason. First, it's easy to see that  $F'$  is satisfiable whenever  $F$  is satisfiable, because  $C' \diamond C''$  is true whenever  $C'$  and  $C''$  are both true. Conversely, if  $F'$  is satisfied by some setting of the other  $n - 1$  variables, that setting must either satisfy  $A'$  for all clauses of the form  $x \vee A'$ , or else it must satisfy  $A''$  for all clauses of the form  $\bar{x} \vee A''$ . (Otherwise neither  $A'$  nor  $A''$  would be satisfied, for some  $A'$  and some  $A''$ , and the clause  $A' \vee A''$  in  $F'$  would be false.) Thus at least one of the settings  $x \leftarrow 0$  or  $x \leftarrow 1$  will satisfy  $F$ .

Another good way to understand variable elimination is to notice that it corresponds to the elimination of an existential quantifier (see exercise 248).

Suppose  $p$  clauses of  $F$  contain  $x$  and  $q$  clauses contain  $\bar{x}$ . Then the elimination of  $x$  will give us at most  $pq$  new clauses, in the worst case; so  $F'$  will have no more clauses than  $F$  did, whenever  $pq \leq p + q$ , namely when  $(p - 1)(q - 1) \leq 1$ . This condition clearly holds whenever  $p = 0$  or  $q = 0$ ; indeed, we called  $x$  a "pure literal" when such cases arose in Algorithm A. The condition also holds whenever  $p = 1$  or  $q = 1$ , and even when  $p = q = 2$ .

Tseytin  
NAND  
variables, introducing new  
auxiliary variables  
memo cache  
extended resolution  
random 3SAT  
3SAT  
tautology  
Samson  
Mueller  
existential quantifier  
quantifiers  
pure literal

Furthermore we don't always get  $pq$  new clauses. Some of the resolvents might turn out to be tautologous, as above; others might be subsumed by existing clauses. (The clause  $C$  is said to *subsume* another clause  $C'$  if  $C \subseteq C'$ , in the sense that every literal of  $C$  appears also in  $C'$ . In such cases we can safely discard  $C'$ .) And some of the resolvents might also subsume existing clauses.

subsumed  
Cook  
Method I  
subsumed  
heuristics  
Method IA  
Reckhow

Therefore repeated elimination of variables doesn't always cause the set of clauses to explode. In the worst case, however, it can be quite inefficient.

In January of 1972, Stephen Cook showed his students at the University of Toronto a rather different way to employ resolution in SAT-solving. His elegant procedure, which he called "Method I," essentially learns new clauses by doing resolution on demand:

**Algorithm I** (*Satisfiability by clause learning*). Given  $m$  nonempty clauses  $C_1 \wedge \dots \wedge C_m$  on  $n$  Boolean variables  $x_1 \dots x_n$ , this algorithm either proves them unsatisfiable or finds strictly distinct literals  $l_1 \dots l_n$  that satisfy them all. In the process, new clauses may be generated by resolution (and  $m$  will then increase).

**I1.** [Initialize.] Set  $d \leftarrow 0$ .

**I2.** [Advance.] If  $d = n$ , terminate successfully (the literals  $\{l_1, \dots, l_d\}$  satisfy  $\{C_1, \dots, C_m\}$ ). Otherwise set  $d \leftarrow d+1$ , and let  $l_d$  be a literal strictly distinct from  $l_1, \dots, l_{d-1}$ .

**I3.** [Find falsified  $C_i$ .] If none of  $C_1, \dots, C_m$  are falsified by  $\{l_1, \dots, l_d\}$ , go back to I2. Otherwise let  $C_i$  be a falsified clause.

**I4.** [Find falsified  $C_j$ .] (At this point we have  $\bar{l}_d \in C_i \subseteq \{\bar{l}_1, \dots, \bar{l}_d\}$ , but no clause is contained in  $\{\bar{l}_1, \dots, \bar{l}_{d-1}\}$ .) Set  $l_d \leftarrow \bar{l}_d$ . If none of  $C_1, \dots, C_m$  are falsified by  $\{l_1, \dots, l_d\}$ , go back to I2. Otherwise let  $\bar{l}_d \in C_j \subseteq \{\bar{l}_1, \dots, \bar{l}_d\}$ .

**I5.** [Resolve.] Set  $m \leftarrow m+1$ ,  $C_m \leftarrow C_i \diamond C_j$ . Terminate unsuccessfully if  $C_m$  is empty. Otherwise set  $d \leftarrow \max\{t \mid \bar{l}_t \in C_m\}$ ,  $i \leftarrow m$ , and return to I4. ■

In step I5 the new clause  $C_m$  cannot be subsumed by any previous clause  $C_k$  for  $k < m$ , because  $C_i \diamond C_j \subseteq \{\bar{l}_1, \dots, \bar{l}_{d-1}\}$ . Therefore, in particular, no clause is generated twice, and the algorithm must terminate.

This description is intentionally vague when it uses the word "let" in steps I2, I3, and I4: *Any* available literal  $l_d$  can be selected in step I2, and *any* falsified clauses  $C_i$  and  $C_j$  can be selected in steps I3 and I4, without making the method fail. Thus Algorithm I really represents a family of algorithms, depending on what heuristics are used to make those selections.

For example, Cook proposed the following way ("Method IA") to select  $l_d$  in step I2: Choose a literal that occurs most frequently in the set of currently unsatisfied clauses that have the fewest unspecified literals. When applied to the six clauses (112), this rule would set  $l_1 \leftarrow 3$  and  $l_2 \leftarrow 2$  and  $l_3 \leftarrow 1$ ; then step I3 would find  $C_i = \bar{1}\bar{2}\bar{3}$  false. So step I4 would set  $l_3 \leftarrow \bar{1}$  and find  $C_j = 1\bar{2}$  false, and step I5 would learn  $C_7 = \bar{2}\bar{3}$ . (See exercise 249 for the sequel.)

Cook's main interest when introducing Algorithm I was to minimize the number of resolution steps; he wasn't particularly concerned with minimizing the running time. Subsequent experiments by R. A. Reckhow [Ph.D. thesis

(Univ. Toronto, 1976), 81–84] showed that, indeed, relatively short resolution refutations are found with this approach. Furthermore, exercise 251 demonstrates that Algorithm I can handle the anti-maximal-element clauses (99)–(101) in polynomial time; thus it trounces the exponential behavior exhibited by all backtrack-based algorithms for this problem (see Theorem R).

On the other hand, Algorithm I does tend to fill memory with a great many new clauses when it is applied to large problems, and there’s no obvious way to deal with those clauses efficiently. Therefore Cook’s method did not appear to be of practical importance, and it remained unpublished for more than forty years.

**Conflict driven clause learning.** Algorithm I demonstrates the fact that unsuccessful choices of literals can lead us to discover valuable new clauses, thereby increasing our knowledge about the characteristics of a problem. When that idea was rediscovered from another point of view in the 1990s, it proved to be revolutionary: Significant industrial instances of SAT with many thousands or even millions of variables suddenly became feasible for the first time.

The name CDCL solver is often given to these new methods, because they are based on “conflict driven clause learning” rather than on classical backtracking. A CDCL solver shares many concepts with the DPLL algorithms that we’ve already seen; yet it is sufficiently different that we can understand it best by developing the ideas from scratch. Instead of implicitly exploring a search tree such as Fig. 39, a CDCL solver is built on the notion of a *trail*, which is a sequence  $L_0L_1 \dots L_{F-1}$  of strictly distinct literals that do not falsify any clause. We can start with  $F = 0$  (the empty trail). As computation proceeds, our task is to extend the current trail until  $F = n$ , thus solving the problem, or to prove that no solution exists, by essentially learning that the empty clause is true.

Suppose there’s a clause  $c$  of the form  $l \vee \bar{a}_1 \vee \dots \vee \bar{a}_k$ , where  $a_1$  through  $a_k$  are in the trail but  $l$  isn’t. Literals in the trail are tentatively assumed to be true, and  $c$  must be satisfied; so we’re forced to make  $l$  true. In such cases we therefore append  $l$  to the current trail and say that  $c$  is its “reason.” (This operation is equivalent to what we called “unit propagation” in previous algorithms; those algorithms effectively removed the literals  $\bar{a}_1, \dots, \bar{a}_k$  when they became false, thereby leaving  $l$  as a “unit” all by itself. But our new viewpoint keeps each clause  $c$  intact, and knows all of its literals.) A *conflict* occurs if the complementary literal  $\bar{l}$  is already in the trail, because  $l$  can’t be both true and false; but let’s assume for now that no conflicts arise, so that  $l$  can legally be appended by setting  $L_F \leftarrow l$  and  $F \leftarrow F + 1$ .

If no such forcing clause exists, and if  $F < n$ , we choose a new distinct literal in some heuristic way, and we append it to the current trail with a “reason” of  $\Lambda$ . Such literals are called *decisions*. They partition the trail into a sequence of decision *levels*, whose boundaries can be indicated by a sequence of indices with  $0 = i_0 \leq i_1 < i_2 < i_3 < \dots$ ; literal  $L_t$  belongs to level  $d$  if and only if  $i_d \leq t < i_{d+1}$ . Level 0, at the beginning of the trail, is special: It contains literals that are forced by clauses of length 1, if such clauses exist. Any such literals are unconditionally true. Every other level begins with exactly one decision.

anti-maximal-element clauses  
Cook  
Conflict driven clause learning  
CDCL solver  
DPLL  
trail  
reason  
unit propagation  
conflict  
forcing clause  
asserting clause, see forcing clause  
decisions  
Level 0

Consider, for example, the problem *waerden*(3, 3; 9) of (9). The first items placed on the trail might be

| $t$ | $L_t$     | level | reason                  |  |
|-----|-----------|-------|-------------------------|--|
| 0   | $\bar{6}$ | 1     | $\Lambda$               | (a decision)   |
| 1   | 9         | 2     | $\Lambda$               | (a decision)   |
| 2   | 3         | 2     | 396                     | (rearrangement of the clause 369)                      |
| 3   | 4         | 3     | $\Lambda$               | (a decision)   |
| 4   | 5         | 3     | 546                     | (rearrangement of the clause 456)                      |
| 5   | 8         | 3     | 846                     | (rearrangement of the clause 468)                      |
| 6   | 2         | 3     | 246                     |  |
| 7   | $\bar{7}$ | 3     | $\bar{7}\bar{5}\bar{3}$ | (rearrangement of the clause $\bar{3}\bar{5}\bar{7}$ ) |
| 8   | $\bar{2}$ | 3     | $\bar{2}\bar{5}\bar{8}$ | (a conflict!)  |

dependence of literals  
learning  
conflict clause  
Resolving

(113)

Three decisions were made, and they started levels at  $i_1 = 0$ ,  $i_2 = 1$ ,  $i_3 = 3$ . Several clauses have been rearranged; we'll soon see why. And propagations have led to a conflict, because both 2 and  $\bar{2}$  have been forced. (We don't actually consider the final entry  $L_8$  to be part of the trail, because it contradicts  $L_6$ .)

If the reason for  $l$  includes the literal  $\bar{l}'$ , we say “ $l$  depends directly on  $l'$ .” And if there's a chain of one or more direct dependencies, from  $l$  to  $l_1$  to  $\dots$  to  $l_k = l'$ , we say simply that “ $l$  depends on  $l'$ .” For example, 5 depends directly on  $\bar{4}$  and  $\bar{6}$  in (113), and  $\bar{2}$  depends directly on 5 and 8; hence  $\bar{2}$  depends on  $\bar{6}$ .

Notice that a literal can depend only on literals that precede it in the trail. Furthermore, every literal  $l$  that's forced at level  $d > 0$  depends directly on some *other* literal on that same level  $d$ ; otherwise  $l$  would already have been forced at a previous level. Consequently  $l$  must necessarily depend on the  $d$ th decision.

The reason for reasons is that we need to deal with conflicts. We will see that every conflict allows us to construct a new clause  $c$  that must be true whenever the existing clauses are satisfiable, although  $c$  itself does not contain any existing clause. Therefore we can “learn”  $c$  by adding it to the existing clauses, and we can try again. This learning process can't go on forever, because only finitely many clauses are possible. Sooner or later we will therefore either find a solution or learn the empty clause. That will be nice, especially if it happens sooner.

A conflict clause  $c$  on decision level  $d$  has the form  $\bar{l} \vee \bar{a}_1 \vee \dots \vee \bar{a}_k$ , where  $l$  and all the  $a$ 's belong to the trail; furthermore  $l$  and at least one  $a_i$  belong to level  $d$ . We can assume that  $l$  is rightmost in the trail, of all the literals in  $c$ . Hence  $l$  cannot be the  $d$ th decision; and it has a reason, say  $l \vee \bar{a}'_1 \vee \dots \vee \bar{a}'_k$ . Resolving  $c$  with this reason gives the clause  $c' = \bar{a}_1 \vee \dots \vee \bar{a}_k \vee \bar{a}'_1 \vee \dots \vee \bar{a}'_k$ , which includes at least one literal belonging to level  $d$ . If more than one such literal is present, then  $c'$  is itself a conflict clause; we can set  $c \leftarrow c'$  and repeat the process. Eventually we are bound to obtain a new clause  $c'$  of the form  $\bar{l}' \vee \bar{b}_1 \vee \dots \vee \bar{b}_r$ , where  $l'$  is on level  $d$  and where  $b_1$  through  $b_r$  are on lower levels.

Such a  $c'$  is learnable, as desired, because it can't contain any existing clauses. (Every subclass of  $c'$ , including  $c'$  itself, would otherwise have given us something to force at a lower level.) We can now *discard* levels  $> d'$  of the trail, where  $d'$  is the maximum level of  $b_1$  through  $b_r$ ; and —this is the punch line—

we can append  $\bar{l}'$  to the end of level  $d'$ , with  $c'$  as its reason. The forcing process now resumes at level  $d'$ , as if the learned clause had been present all along.

For example, after the conflict in (113), the initial conflict clause is  $c = \bar{2}\bar{5}\bar{8}$ , our shorthand notation for  $\bar{x}_2 \vee \bar{x}_5 \vee \bar{x}_8$ ; and its rightmost complemented literal in the trail is 2, because 5 and 8 came earlier. So we resolve  $c$  with 246, the reason for 2, and get  $c' = 4\bar{5}\bar{6}\bar{8}$ . This new clause contains complements of three literals from level 3, namely  $\bar{4}$ ,  $\bar{5}$ , and  $\bar{8}$ ; so it's still a conflict clause. We resolve it with the reason for 8 and get  $c' = 4\bar{5}6$ . Again  $c'$  is a conflict clause. But the result of resolving this conflict with the reason for 5 is  $c' = 46$ , a clause that is falsified by the literals currently on the trail but has only  $\bar{4}$  at level 3. Good—we have learned ‘46’: In every solution to *waerden*(3, 3; 9), either  $x_4$  or  $x_6$  must be true.

Thus the sequel to (113) is

| $t$ | $L_t$     | level | reason    |   |
|-----|-----------|-------|-----------|---|
| 0   | $\bar{6}$ | 1     | $\Lambda$ | (a decision) <span style="float: right;">(114)</span> |
| 1   | 4         | 1     | 46        | (the newly learned clause)                            |

and the next step will be to begin a new level 2, because nothing more is forced.

Notice that the former level 2 has gone away. We've learned that there was no need to branch on the decision variable  $x_9$ , because  $\bar{6}$  already forces 4. This improvement to the usual backtrack regimen is sometimes called “backjumping,” because we've jumped back to a level that can be regarded as the root cause of the conflict that was just discovered.

Exercise 253 explores a possible continuation of (114); dear reader, please jump to it now. Incidentally, the clause ‘46’ that we learned in this example involves the complements of former decisions  $\bar{4}$  and  $\bar{6}$ ; but exercise 255 shows that newly learned clauses might not contain any decision variables whatsoever.

The process of constructing the learned clause from a conflict is not as difficult as it may seem, because there's an efficient way to perform all of the necessary resolution steps. Suppose, as above, that the initial conflict clause is  $\bar{l} \vee \bar{a}_1 \vee \dots \vee \bar{a}_k$ . Then we “stamp” each of the literals  $a_i$  with a unique number  $s$ ; and we also insert  $\bar{a}_i$  into an auxiliary array, which will eventually hold the literals  $\bar{b}_1, \dots, \bar{b}_r$ , whenever  $a_i$  is a literal that received its value on a level  $d'$  with  $0 < d' < d$ . We stamp  $l$  too; and we count how many literals of level  $d$  have thereby been stamped. Then we repeatedly go back through the trail until coming to a literal  $L_t$  whose stamp equals  $s$ . If the counter is bigger than 1 at this point, and if the reason of  $L_t$  is  $L_t \vee \bar{a}'_1 \vee \dots \vee \bar{a}'_{k'}$ , we look at each  $a'_i$ , stamping it and possibly putting it into the  $b$  array if it had not already been stamped with  $s$ . Eventually the count of unresolved literals will decrease to 1; the learned clause is then  $\bar{L}_t \vee \bar{b}_1 \vee \dots \vee \bar{b}_r$ .

These new clauses might turn out to be quite large, even when we're solving a problem whose clauses were rather small to start with. For example, Table 3 gives a glimpse of typical behavior in a medium-size problem. It shows the beginning of the trail generated when a CDCL solver was applied to the 2779 clauses of *waerden*(3, 10; 97), after about 10,000 clauses had been learned. (Recall that this problem tries to find a binary vector  $x_1 x_2 \dots x_{97}$  that has no three equally

backtrack  
backjumping  
look-back, see backjumping  
stamp  
waerden

**Table 3**  
THE FIRST LEVELS OF A MODERATE-SIZE TRAIL

| $t$ | $L_t$           | level | reason    | $t$ | $L_t$           | level | reason    | $t$ | $L_t$           | level | reason    |
|-----|-----------------|-------|-----------|-----|-----------------|-------|-----------|-----|-----------------|-------|-----------|
| 0   | $\overline{53}$ | 1     | $\Lambda$ | 15  | 70              | 11    | 70 36 53  | 30  | 08              | 15    | 08 46 27  |
| 1   | 55              | 2     | $\Lambda$ | 16  | 35              | 12    | $\Lambda$ | 31  | 65              | 15    | 65 46 27  |
| 2   | 44              | 3     | $\Lambda$ | 17  | 39              | 13    | $\Lambda$ | 32  | 60              | 15    | 60 46 53  |
| 3   | 54              | 4     | $\Lambda$ | 18  | $\overline{37}$ | 14    | $\Lambda$ | 33  | $\overline{50}$ | 15    | **        |
| 4   | 43              | 5     | $\Lambda$ | 19  | 38              | 14    | 38 37 36  | 34  | 64              | 15    | 64 50 36  |
| 5   | 30              | 6     | $\Lambda$ | 20  | 47              | 14    | 47 37 27  | 35  | 22              | 15    | 22 50 36  |
| 6   | 34              | 7     | $\Lambda$ | 21  | 17              | 14    | 17 37 27  | 36  | 24              | 15    | 24 50 37  |
| 7   | 45              | 8     | $\Lambda$ | 22  | 32              | 14    | 32 37 27  | 37  | 42              | 15    | 42 50 46  |
| 8   | 40              | 9     | $\Lambda$ | 23  | 69              | 14    | 69 37 53  | 38  | 48              | 15    | 48 50 46  |
| 9   | $\overline{27}$ | 10    | $\Lambda$ | 24  | 21              | 14    | 21 37 53  | 39  | 73              | 15    | 73 50 27  |
| 10  | 79              | 10    | 79 53 27  | 25  | $\overline{46}$ | 15    | $\Lambda$ | 40  | 04              | 15    | 04 50 27  |
| 11  | 01              | 10    | 01 27 53  | 26  | 28              | 15    | 28 46 37  | 41  | 63              | 15    | 63 50 37  |
| 12  | $\overline{36}$ | 11    | $\Lambda$ | 27  | 41              | 15    | 41 46 36  | 42  | 33              | 16    | $\Lambda$ |
| 13  | 18              | 11    | 18 36 27  | 28  | 26              | 15    | 26 46 36  | 43  | 51              | 17    | $\Lambda$ |
| 14  | 19              | 11    | 19 36 53  | 29  | 56              | 15    | 56 46 36  | 44  | $\overline{57}$ | 18    | $\Lambda$ |

redundant  
unit propagation  
lazy data structures

(Here \*\* stands for the previously learned clause  $\overline{50} \overline{26} \overline{47} \overline{35} \overline{41} \overline{32} \overline{38} \overline{44} \overline{27} \overline{45} \overline{55} \overline{65} \overline{60} \overline{70} \overline{30}$ .)

spaced 0s and no ten equally spaced 1s.) Level 18 in the table has just been launched with the decision  $L_{44} = \overline{57}$ ; and that decision will trigger the setting of many more literals 15, 49, 61, 68, 77, 78, 87,  $\overline{96}$ , . . . , eventually leading to a conflict when trying to set  $L_{67}$ . The conflict clause turns out to have length 22:

$$53 \ 27 \ 36 \ \overline{70} \ \overline{35} \ 37 \ \overline{69} \ \overline{21} \ 46 \ \overline{28} \ \overline{56} \ \overline{65} \ \overline{60} \ 50 \ \overline{64} \ \overline{24} \ \overline{42} \ \overline{73} \ \overline{63} \ \overline{33} \ \overline{51} \ 57. \quad (115)$$

(Its literals are shown here in order of the appearance of their complements in the trail.) When we see such a monster clause, we might well question whether we really want to “learn” such an obscure fact!

A closer look, however, reveals that many of the literals in (115) are redundant. For example,  $\overline{70}$  can safely be deleted, because its reason is ‘70 36 53’; both 36 and 53 already appear in (115), hence  $(115) \diamond (70 \ 36 \ 53)$  gets rid of  $\overline{70}$ . Indeed, more than half of the literals in this example are redundant, and (115) can be simplified to the much shorter and more memorable clause

$$53 \ 27 \ 36 \ \overline{35} \ 37 \ 46 \ 50 \ \overline{33} \ \overline{51} \ 57. \quad (116)$$

Exercise 257 explains how to discover such simplifications, which turn out to be quite important in practice. For example, the clauses learned while proving *waerden*(3, 10; 97) unsatisfiable had an average length of 19.9 before simplification, but only 11.2 after; simplification made the algorithm run about 33% faster.

Most of the computation time of a CDCL solver is devoted to unit propagation. Thus we need to know when the value of a literal has been forced by previous assignments, and we hope to know it quickly. The idea of “lazy data structures,” used above in Algorithm D, works nicely for this purpose, in the presence of long clauses, provided that we extend it so that every clause now has *two*

watched literals instead of one. If we know that the first two literals of a clause are not false, then we needn't look at this clause until one of them becomes false, even though other literals in the clause might be repeatedly veering between transient states of true, false, and undefined. And when a watchee does become false, we'll try to swap it with a nonfalse partner that can be watched instead. Propagations or conflicts will arise only when all of the remaining literals are false.

Algorithm C below therefore represents clauses with the following data structures: A monolithic array called MEM is assumed to be large enough to hold all of the literals in all of the clauses, interspersed with control information. Each clause  $c = l_0 \vee l_1 \vee \dots \vee l_{k-1}$  with  $k > 1$  is represented by its starting position in MEM, with  $\text{MEM}[c + j] = l_j$  for  $0 \leq j < k$ . Its two watched literals are  $l_0$  and  $l_1$ , and its size  $k$  is stored in  $\text{MEM}[c - 1]$ . Unit clauses, for which  $k = 1$ , are treated differently; they appear in level 0 of the trail, not in MEM.

A learned clause  $c$  can be distinguished from an initial clause because it has a relatively high number, with  $\text{MINL} \leq c < \text{MAXL}$ . Initially MAXL is set equal to MINL, the smallest cell in MEM that is available for learned clauses; then MAXL grows as new clauses are added to the repertoire. The set of learned clauses is periodically culled, so that the less desirable ones don't clutter up memory and slow things down. Additional information about a learned clause  $c$  is kept in  $\text{MEM}[c - 4]$  and  $\text{MEM}[c - 5]$ , to help with this recycling process (see below).

Individual literals  $x_k$  and  $\bar{x}_k$ , for  $1 \leq k \leq n$ , are represented internally by the numbers  $2k$  and  $2k + 1$  as in (57) above. And each of these  $2n$  literals  $l$  has a list pointer  $W_l$ , which begins a linked list of the clauses in which  $l$  is watched. We have  $W_l = 0$  if there is no such clause; but if  $W_l = c > 0$ , the next link in this "watch list" is in  $\text{MEM}[c - 2]$  if  $l = l_0$ , in  $\text{MEM}[c - 3]$  if  $l = l_1$ . [See Armin Biere, *Journal on Satisfiability, Boolean Modeling and Comp.* 4 (2008), 75–97.]

For example, the first few cells of MEM might contain the following data when we are representing the clauses (9) of *waarden* (3, 3; 9):

```
i = 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 ...
MEM[i] = 9 45 3  2  4  6 15 51 3  4  6  8 21 45 3  6  8 10 ...
```

(Clause 3 is '123', clause 9 is '234', clause 15 is '345', ..., clause 45 is '135', clause 51 is '246', ...; the watch lists for literals  $x_1, x_2, x_3, x_4$  begin respectively at  $W_2 = 3, W_4 = 3, W_6 = 9, W_8 = 15$ .)

The other major data structures of Algorithm C are focused on variables, not clauses. Each variable  $x_k$  for  $1 \leq k \leq n$  has six current attributes  $\text{S}(k)$ ,  $\text{VAL}(k)$ ,  $\text{OVAL}(k)$ ,  $\text{TLOC}(k)$ ,  $\text{HLOC}(k)$ , and  $\text{ACT}(k)$ , which interact as follows:  $\text{S}(k)$  is the "stamp" that's used during clause formation. If neither  $x_k$  nor  $\bar{x}_k$  appears in the current trail, then  $\text{VAL}(k) = -1$ , and we say that  $x_k$  and its two literals are "free." But if  $L_t = l$  is a literal of the trail, belonging to level  $d$ , we have

$$\text{VAL}(|l|) = 2d + (l \& 1) \quad \text{and} \quad \text{TLOC}(|l|) = t, \quad \text{where } |l| = l \gg 1, \quad (117)$$

and we say that  $l$  is "true" and  $\bar{l}$  is "false." Thus a given literal  $l$  is false if and only if  $\text{VAL}(|l|)$  is nonnegative and  $\text{VAL}(|l|) + l$  is odd. In most cases a watched literal is not false; but there are exceptions to this rule (see exercise 261).

```
MEM
Unit clauses
level 0
recycling
watch list
Biere
stamp
free literals and free variables
true literals
false literals
watched literal
```

The attributes  $\text{ACT}(k)$  and  $\text{HLOC}(k)$  tell the algorithm how to select the next decision variable. Each variable  $x_k$  has an *activity* score  $\text{ACT}(k)$ , which heuristically estimates its desirability for branching. All of the free variables, and possibly others, are kept in an array called **HEAP**, which is arranged so that

$$\text{ACT}(\text{HEAP}[j]) \leq \text{ACT}(\text{HEAP}[(j-1) \gg 1]) \quad \text{for } 0 < j < h \quad (118)$$

when it contains  $h$  elements (see Section 5.2.3). Thus  $\text{HEAP}[0]$  will always be a free variable of maximum activity, if it is free; so it's the variable that will be chosen to govern the decision when the trail starts to acquire a new level.

Activity scores help the algorithm to focus on recent conflicts. Suppose, for example, that 100 conflicts have been resolved, hence 100 clauses have been learned. Suppose further that  $x_j$  or  $\bar{x}_j$  was stamped while resolving the conflicts numbered 3, 47, 95, 99, and 100; but  $x_k$  or  $\bar{x}_k$  was stamped during conflicts 41, 87, 94, 95, 96, and 97. We could express their recent activity by computing

$$\text{ACT}(j) = \rho^0 + \rho^1 + \rho^5 + \rho^{53} + \rho^{97}, \quad \text{ACT}(k) = \rho^3 + \rho^4 + \rho^5 + \rho^6 + \rho^{13} + \rho^{59},$$

where  $\rho$  is a damping factor (say  $\rho = .95$ ), because  $100 - 100 = 0$ ,  $100 - 99 = 1$ ,  $100 - 95 = 5$ ,  $\dots$ ,  $100 - 41 = 59$ . In this particular case  $j$  would be considered to be less active than  $k$  unless  $\rho$  is less than about .8744.

In order to update the activity scores according to this measure, we would have to do quite a bit of recomputation whenever a new conflict occurs: The new scores would require us to multiply all  $n$  of the old scores by  $\rho$ , then to increase the activity of every newly stamped variable by 1. But there's a much better way, namely to compute  $\rho^{-100}$  times the scores shown above:

$$\text{ACT}(j) = \rho^{-3} + \rho^{-47} + \rho^{-95} + \rho^{-99} + \rho^{-100}, \quad \text{ACT}(k) = \rho^{-41} + \dots + \rho^{-96} + \rho^{-97}.$$

These newly scaled scores, suggested by Niklas Eén, give us the same information about the relative activity of each variable; and they're updated easily, because we need to do only one addition per stamped variable when resolving conflicts.

The only problem is that the new scores can become really huge, because  $\rho^{-M}$  can cause floating point overflow after the number  $M$  of conflicts becomes large. The remedy is to *divide* them all by  $10^{100}$ , say, whenever any variable gets a score that exceeds  $10^{100}$ . The **HEAP** needn't change, since (118) still holds.

During the algorithm the variable **DEL** holds the current scaling factor  $\rho^{-M}$ , divided by  $10^{100}$  each time all of the activities have been rescaled.

Finally, the parity of  $\text{OVAL}(k)$  is used to control the polarity of each new decision in step C6. Algorithm C starts by simply making each  $\text{OVAL}(k)$  odd, although other initialization schemes are possible. Afterwards it sets  $\text{OVAL}(k) \leftarrow \text{VAL}(k)$  whenever  $x_k$  leaves the trail and becomes free, as recommended by D. Frost and R. Dechter [*AAAI Conf.* **12** (1994), 301–306] and independently by K. Pipatsrisawat and A. Darwiche [*LNCS 4501* (2007), 294–299], because experience has shown that the recently forced polarities tend to remain good. This technique is called “sticking” or “progress saving” or “phase saving.”

Algorithm C is based on the framework of a pioneering CDCL solver called Chaff, and on an early descendant of Chaff called MiniSAT that was developed by N. Eén and N. Sörensson [*LNCS 2919* (2004), 502–518].

activity  
 $\text{ACT}(k)$   
 heuristic  
 heap  
 focus  
 Mathews, Edwin Lee (41)  
 damping factor  
 Eén  
 floating point overflow  
 rescaling  
 Frost  
 Dechter  
 AAAI: American Association for Artificial Intelligence  
 Pipatsrisawat  
 Darwiche  
 polarities  
 sticking values  
 progress saving  
 phase saving  
 Chaff  
 MiniSAT  
 Eén  
 Sörensson



**Algorithm C** (*Satisfiability by CDCL*). Given a set of clauses on  $n$  Boolean variables, this algorithm finds a solution  $L_0L_1 \dots L_{n-1}$  if and only if the clauses are satisfiable, meanwhile discovering  $M$  new ones that are consequences of the originals. After discovering  $M_p$  new clauses, it will purge some of them from its memory and reset  $M_p$ ; after discovering  $M_f$  of them, it will flush part of its trail, reset  $M_f$ , and start over. (Details of purging and flushing will be discussed later.)

purging  
flushing  
breadth-first search  
unit propagation  
Backjump

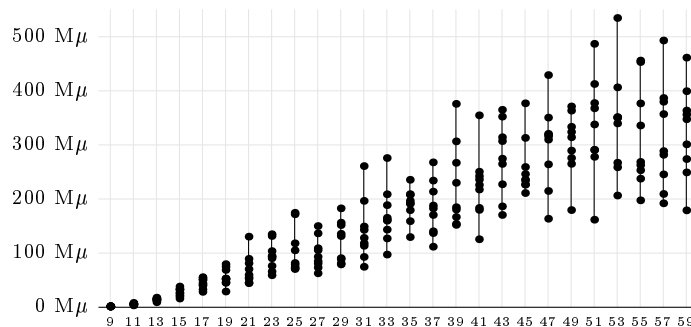
- C1.** [Initialize.] Set  $\text{VAL}(k) \leftarrow \text{OVAL}(k) \leftarrow \text{TLOC}(k) \leftarrow -1$ ,  $\text{ACT}(k) \leftarrow \text{S}(k) \leftarrow 0$ ,  $R_{2k} \leftarrow R_{2k+1} \leftarrow \Lambda$ ,  $\text{HLOC}(k) \leftarrow p_k - 1$ , and  $\text{HEAP}[p_k - 1] \leftarrow k$ , for  $1 \leq k \leq n$ , where  $p_1 \dots p_n$  is a random permutation of  $\{1, \dots, n\}$ . Then input the clauses into MEM and the watch lists, as described above. Put the distinct unit clauses into  $L_0L_1 \dots L_{F-1}$ ; but terminate unsuccessfully if there are contradictory clauses ( $l$ ) and ( $\bar{l}$ ). Set MINL and MAXL to the first available position in MEM. (See exercise 260.) Set  $i_0 \leftarrow d \leftarrow s \leftarrow M \leftarrow G \leftarrow 0$ ,  $h \leftarrow n$ ,  $\text{DEL} \leftarrow 1$ .
- C2.** [Level complete?] (The trail  $L_0 \dots L_{F-1}$  now contains all of the literals that are forced by  $L_0 \dots L_{G-1}$ .) Go to C5 if  $G = F$ .
- C3.** [Advance  $G$ .] Set  $l \leftarrow L_G$  and  $G \leftarrow G + 1$ . Then do step C4 for all  $c$  in the watch list of  $\bar{l}$ , unless that step detects a conflict and jumps to C7. If there is no conflict, return to C2. (See exercise 261.)
- C4.** [Does  $c$  force a unit?] Let  $l_0l_1 \dots l_{k-1}$  be the literals of clause  $c$ , where  $l_1 = \bar{l}$ . (Swap  $l_0 \leftrightarrow l_1$  if necessary.) If  $l_0$  is true, do nothing. Otherwise look for a literal  $l_j$  with  $1 < j < k$  that is not false. If such a literal is found, move  $c$  to the watch list of  $l_j$ . But if  $l_2, \dots, l_{k-1}$  are all false, jump to C7 if  $l_0$  is also false. On the other hand if  $l_0$  is free, make it true by setting  $L_F \leftarrow l_0$ ,  $\text{TLOC}(|l_0|) \leftarrow F$ ,  $\text{VAL}(|l_0|) \leftarrow 2d + (l_0 \& 1)$ ,  $R_{l_0} \leftarrow c$ , and  $F \leftarrow F + 1$ .
- C5.** [New level?] If  $F = n$ , terminate successfully. Otherwise if  $M \geq M_p$ , prepare to purge excess clauses (see below). Otherwise if  $M \geq M_f$ , flush literals as explained below and return to C2. Otherwise set  $d \leftarrow d + 1$  and  $i_d \leftarrow F$ .
- C6.** [Make a decision.] Set  $k \leftarrow \text{HEAP}[0]$  and delete  $k$  from the heap (see exercises 262 and 266). If  $\text{VAL}(k) \geq 0$ , repeat this step. Otherwise set  $l \leftarrow 2k + (\text{OVAL}(k) \& 1)$ ,  $\text{VAL}(k) \leftarrow 2d + (\text{OVAL}(k) \& 1)$ ,  $L_F \leftarrow l$ ,  $\text{TLOC}(|l|) \leftarrow F$ ,  $R_l \leftarrow \Lambda$ , and  $F \leftarrow F + 1$ . (At this point  $F = G + 1$ .) Go to C3.
- C7.** [Resolve a conflict.] Terminate unsuccessfully if  $d = 0$ . Otherwise use the conflict clause  $c$  to construct a new clause  $\bar{l}' \vee \bar{b}_1 \vee \dots \vee \bar{b}_r$  as described above. Set  $\text{ACT}(|l|) \leftarrow \text{ACT}(|l|) + \text{DEL}$  for all literals  $l$  stamped during this process; also set  $d'$  to the maximum level occupied by  $\{b_1, \dots, b_r\}$  in the trail. (See exercise 263. Increasing  $\text{ACT}(|l|)$  may also change HEAP.)
- C8.** [Backjump.] While  $F > i_{d+1}$ , do the following: Set  $F \leftarrow F - 1$ ,  $l \leftarrow L_F$ ,  $k \leftarrow |l|$ ,  $\text{OVAL}(k) \leftarrow \text{VAL}(k)$ ,  $\text{VAL}(k) \leftarrow -1$ ,  $R_l \leftarrow \Lambda$ ; and if  $\text{HLOC}(|l|) < 0$  insert  $k$  into HEAP (see exercise 262). Then set  $G \leftarrow F$  and  $d \leftarrow d'$ .
- C9.** [Learn.] If  $d > 0$ , set  $c \leftarrow \text{MAXL}$ , store the new clause in MEM at position  $c$ , and advance MAXL to the next available position in MEM. (Exercise 263 gives full details.) Set  $M \leftarrow M + 1$ ,  $L_F \leftarrow l'$ ,  $\text{TLOC}(|l'|) \leftarrow F$ ,  $R_{l'} \leftarrow c$ ,  $F \leftarrow F + 1$ ,  $\text{DEL} \leftarrow \text{DEL}/\rho$ , and return to C3. ■

The high-level operations on data structures in this algorithm are spelled out in terms of elementary low-level steps in exercises 260–263. Exercises 266–271 discuss simple enhancements that were made in the experiments reported below.

*Reality check:* Although detailed statistics about the performance of Algorithm C on a wide variety of problems will be presented later, a few examples of typical behavior will help now to clarify how the method actually works in practice. Random choices make the running time of this algorithm more variable than it was in Algorithms A, B, D, or L; sometimes we’re lucky, sometimes we’re not.

In the case of *waarden*(3,10;97), the modest 97-variable-and-2779-clause problem that was considered in Table 3, nine test runs of Algorithm C established unsatisfiability after making between 250 and 300 million memory accesses; the median was 272  $M\mu$ . (This is more than twice as fast as our best previous time, which was obtained with Algorithm L.) The average number of decisions made—namely the number of times  $L_F \leftarrow l$  was done in step C6—was about 63 thousand; this compares to 1701 “nodes” in Algorithm L, step L3, and 100 million nodes in Algorithms A, B, D. About 53 thousand clauses were learned, having an average size of 11.5 literals (after averaging about 19.9 before simplification).

**Fig. 49.** It is not possible to color the edges of the flower snark graph  $J_q$  with three colors, when  $q$  is odd. Algorithm C is able to prove this with amazing speed: Computation times (in megamems) are shown for nine trials at each value of  $q$ .



Algorithm C often speeds things up much more dramatically, in fact. For example, Fig. 49 shows how it whips through a sequence of three-coloring problems that are based on “flower snarks.” Exercise 176 defines *fsnark*( $q$ ), an interesting set of  $42q + 3$  unsatisfiable clauses on  $18q$  variables. The running time of Algorithms A, B, D, and L on *fsnark*( $q$ ) is proportional to  $2^q$ , so it’s way off the chart—well over a gigamem already when  $q = 19$ . But Algorithm C polishes off the case  $q = 99$  in that same amount of time (thus winning by 24 orders of magnitude)! On the other hand, no satisfactory theoretical explanation for the apparently linear behavior in Fig. 49 is presently known.

**Certificates of unsatisfiability.** When a SAT solver reports that a given instance is satisfiable, it also produces a set of distinct literals from which we can easily check that every clause is satisfied. But if its report is *negative*—UNSAT—how confident can we be that such a claim is true? Maybe the implementation contains a subtle error; after all, large and complicated programs are notoriously buggy, and computer hardware isn’t perfect either. A negative answer can therefore leave both programmers and users unsatisfied, as well as the problem.

decisions  
nodes  
three-coloring problems  
flower snarks  
snark graphs  
*fsnark*  
Certificates of unsatisfiability

We've seen that unsatisfiability can be proved rigorously by constructing a resolution refutation, namely a chain of resolution steps that ends with the empty clause  $\epsilon$ , as in Fig. 48. But such refutations amount to the construction of a huge directed acyclic graph.

A much more compact characterization of unsatisfiability is possible. Let's say that the sequence of clauses  $(C_1, C_2, \dots, C_t)$  is a *certificate of unsatisfiability* for a family of clauses  $F$  if  $C_t = \epsilon$ , and if we have

$$F \wedge C_1 \wedge \dots \wedge C_{i-1} \wedge \overline{C}_i \vdash_1 \epsilon \quad \text{for } 1 \leq i \leq t. \quad (119)$$

Here the subscript 1 in ' $G \vdash_1 \epsilon$ ' means that the clauses  $G$  lead to a contradiction by *unit propagation*; and if  $C_i$  is the clause  $(a_1 \vee \dots \vee a_k)$ , then  $\overline{C}_i$  is an abbreviation for the conjunction of unit clauses  $(\overline{a}_1) \wedge \dots \wedge (\overline{a}_k)$ .

For example, let  $F = R$  be Rivest's clauses (6), which were proved unsatisfiable in Fig. 48. Then  $(12, 1, 2, \epsilon)$  is a certificate of unsatisfiability, because

$$\begin{aligned} R \wedge \overline{1} \wedge \overline{2} \vdash_1 \overline{3} \vdash_1 \overline{4} \vdash_1 \epsilon & \quad (\text{using } 12\overline{3}, 23\overline{4}, \text{ and } 341); \\ R \wedge 12 \wedge \overline{1} \vdash_1 2 \vdash_1 \overline{4} \vdash_1 \overline{3} \vdash_1 \epsilon & \quad (\text{using } 12, \overline{4}1\overline{2}, \overline{2}3\overline{4}, \text{ and } 341); \\ R \wedge 12 \wedge 1 \wedge \overline{2} \vdash_1 4 \vdash_1 3 \vdash_1 \epsilon & \quad (\text{using } 41\overline{2}, 23\overline{4}, \text{ and } 34\overline{1}); \\ R \wedge 12 \wedge 1 \wedge 2 \vdash_1 3 \vdash_1 4 \vdash_1 \epsilon & \quad (\text{using } \overline{1}2\overline{3}, \overline{2}3\overline{4}, \text{ and } 34\overline{1}). \end{aligned}$$

A certificate of unsatisfiability gives a convincing proof, since (119) implies that each  $C_i$  must be true whenever  $F, C_1, \dots, C_{i-1}$  are true. And it's easy to check whether or not  $G \vdash_1 \epsilon$ , for any given set of clauses  $G$ , because everything is forced and no choices are involved. Unit propagation is analogous to water flowing downhill; we can be pretty sure that it has been implemented correctly, even if we don't trust the CDCL solver that generated the certificate being checked.

E. Goldberg and Y. Novikov [*Proceedings of DATE: Design, Automation and Test in Europe* 6,1 (2003), 886–891] have pointed out that CDCL solvers actually produce such certificates as a natural byproduct of their operation:

**Theorem G.** *If Algorithm C terminates unsuccessfully, the sequence  $(C_1, C_2, \dots, C_t)$  of clauses that it has learned is a certificate of unsatisfiability.*

*Proof.* It suffices to show that, whenever Algorithm C has learned the clause  $C' = \overline{l}' \vee \overline{b}_1 \vee \dots \vee \overline{b}_r$ , unit propagation will deduce  $\epsilon$  if we append the unit clauses  $(l') \wedge (b_1) \wedge \dots \wedge (b_r)$  to the clauses that the algorithm already knows. The key point is that  $C'$  has essentially been obtained by repeated resolution steps,

$$C' = (\dots((C \diamond R_{l_1}) \diamond R_{l_2}) \diamond \dots) \diamond R_{l_s}, \quad (120)$$

where  $C$  is the original conflict clause and  $R_{l_1}, R_{l_2}, \dots, R_{l_s}$  are the reasons for each literal that was removed while  $C'$  was constructed in step C7. More precisely, we have  $C = A_0$  and  $R_{l_i} = l_i \vee A_i$ , where all literals of  $A_0 \cup A_1 \cup \dots \cup A_s$  are false (their complements appear in the trail); and

$$\begin{aligned} \overline{l}_i \in A_0 \cup \dots \cup A_{i-1}, \text{ for } 1 \leq i \leq s; \\ A_0 \cup A_1 \cup \dots \cup A_s = \{\overline{l}', \overline{l}_1, \dots, \overline{l}_s, \overline{b}_1, \dots, \overline{b}_r\}. \end{aligned} \quad (121)$$

Thus the known clauses, plus  $b_1, \dots, b_r$ , and  $l'$ , will force  $l_s$  using clause  $R_{l_s}$ . And  $l_{s-1}$  will then be forced, using  $R_{l_{s-1}}$ . And so on. ■

resolution refutation  
clausal proofs, see certificates of unsat  
notation  $F \vdash_1 \epsilon$   
unit propagation  
unit clauses  
Rivest  
Goldberg  
Novikov  
learned clauses, sequence of  
resolution  
conflict clause

Since the unit literals in this proof are propagated in reverse order  $l_s, l_{s-1}, \dots, l_1$  from the resolution steps in (120), this certificate-checking procedure has become known as “reverse unit propagation” [see A. Van Gelder, *Proc. Int. Symp. on Artificial Intelligence and Math.* **10** (2008), 9 pages, online as ISAIM2008].

Notice that the proof of Theorem G doesn’t claim that reverse unit propagation will reconstruct the precise reasoning by which Algorithm C learned a clause. Many different downhill paths to  $\epsilon$ , built from  $\vdash_1$  steps, usually exist in a typical situation. We merely have shown that every clause learnable from a single conflict does imply the existence of at least one such downhill path.

Many of the clauses learned during a typical run of Algorithm C will be “shots in the dark,” which turn out to have been aimed in unfruitful directions. Thus the certificates in Theorem G will usually be longer than actually necessary to demonstrate unsatisfiability. For example, Algorithm C learns about 53,000 clauses when refuting *waarden*(3, 10; 97), and about 135,000 when refuting *fsnark*(99); but fewer than 50,000 of the former, and fewer than 47,000 of the latter, were actually used in subsequent steps. Exercise 284 explains how to shorten a certificate of unsatisfiability while checking its validity.

An unexpected difficulty arises, however: We might spend more time verifying a certificate than we needed to generate it! For example, a certificate for *waarden*(3, 10; 97) was found in 272 megamems, but the time needed to check it with straightforward unit-propagations was actually 2.2 *gigamems*. Indeed, this discrepancy becomes significantly worse in larger problems, because a simple program for checking must keep all of the clauses active in its memory. If there are a million active clauses, there are two million literals being watched; hence every change to a literal will require many updates to the data structures.

The solution to this problem is to provide extra hints to the certificate checker. As we are about to see, Algorithm C does *not* keep all of the learned clauses in its memory; it systematically purges its collection, so that the total number stays reasonable. At such times it can also inform the certificate checker that the purged clauses will no longer be relevant to the proof.

Further improvements also allow annotated certificates to accommodate stronger proof rules, such as Tseytin’s extended resolution and techniques based on generalized autarkies; see N. Wetzler, M. J. H. Heule, and W. A. Hunt, Jr., *LNCS 8561* (2014), 422–429.

Whenever a family of clauses has a certificate of unsatisfiability, a variant of Algorithm C will actually find one that isn’t too much longer. (See exercise 386.)

**\*Purging unhelpful clauses.** After thousands of conflicts have occurred, Algorithm C has learned thousands of new clauses. New clauses guide the search by steering us away from unproductive paths; but they also slow down the propagation process, because we have to watch them.

We’ve seen that certificates can usually be shortened; therefore we know that many of the learned clauses will probably never be needed again. For this reason Algorithm C periodically attempts to weed out the ones that appear to be more harmful than helpful, by ranking the clauses that have accumulated.

reverse unit propagation  
Van Gelder  
purges  
Tseytin  
extended resolution  
autarkies  
Wetzler  
Heule  
Hunt  
Purging unhelpful clauses

*I consider that a man's brain originally is like a little empty attic, and you have to stock it with such furniture as you choose. . . . the skilled workman is very careful indeed as to what he takes into his brain-attic. . . . It is a mistake to think that that little room has elastic walls and can distend to any extent. . . . It is of the highest importance, therefore, not to have useless facts elbowing out the useful ones.*

— SHERLOCK HOLMES, in *A Study in Scarlet* (1887)

HOLMES  
Doyle  
*waerden* (3, 10; 97)  
author  
discarding  
reason  
trail  
heuristics  
Audemard  
Simon  
literal block distance  
signature  
literal block distance  
glucose, see literal block distance

Algorithm C initiates a special clause-refinement process as soon as it has learned  $M \geq M_p$  clauses and arrived at a reasonably stable state (step C5). Let's continue our running example, *waerden*(3, 10; 97), in order to make the issues concrete. If  $M_p$  is so huge that no clauses are ever thrown away, a typical run will learn roughly 48 thousand clauses, and do roughly 800 megamems of computation, before proving unsatisfiability. But if  $M_p = 10000$ , it will learn roughly 50 thousand clauses, and the computation time will go down to about 500 megamems. In the latter case the total number of learned clauses in memory will rarely exceed 10 thousand.

Indeed, let's set  $M_p = 10000$  and take a close look at exactly what happened during the author's first experiments. Algorithm C paused to reconnoiter the situation after having learned 10002 clauses. At that point only 6252 of those 10002 clauses were actually present in memory, however, because of the clause-discarding mechanism discussed in exercise 271. Some clauses had length 2, while the maximum size was 24 and the median was 11; here's a complete histogram:

2 9 49 126 216 371 542 719 882 1094 661 540 414 269 176 111 35 20 10 3 1 1 1.

Short clauses tend to be more useful, because they reduce more quickly to units.

A learned clause cannot be purged if it is the reason for one of the literals on the trail. In our example, 12 of the 6252 fell into this category; for instance,  $\overline{30}$  appeared on level 10 of the trail because ' $\overline{30} \overline{33} \overline{39} 41 \overline{42} \overline{45} 46 \overline{48} \overline{54} \overline{57}$ ' had been learned, and we may need to know that clause in a future resolution step.

The purging process will try to remove at least half of the existing learned clauses, so that at most 3126 remain. We aren't allowed to touch the 12 reason-bound ones; hence we want to forget 3114 of the other 6240. Which of them should we expel?

Among many heuristics that have been tried, the most successful in practice are based on what Gilles Audemard and Laurent Simon have called "literal block distance" [see *Proc. Int. Joint Conference on Artificial Intelligence* **21** (2009), 399–404]. They observed that each level of the trail can be considered to be a block of more-or-less related variables; hence a long clause might turn out to be more useful than a short clause, if the literals of the long one all lie on just one or two levels while the literals of the short one belong to three or more.

Suppose all the literals of a clause  $C = l_1 \vee \cdots \vee l_r$  appear in the trail, either positively as  $l_j$  or negatively as  $\bar{l}_j$ . We can group them by level so that exactly  $p + q$  levels are represented, where  $p$  of the levels contain at least one positive  $l_j$  and the other  $q$  contain nothing but  $\bar{l}_j$ 's. Then  $(p, q)$  is the *signature* of  $C$  with respect to the trail, and  $p + q$  is the literal block distance. For example, the very

first clause learned from *waerden*(3, 10; 97) in the author’s test run was

$$\overline{11} \overline{16} \overline{21} \overline{26} \overline{36} \overline{46} \overline{51} 61 66 91; \tag{122}$$

later, when it was time to rank clauses for purging, the values and trail levels of those literals were specified by VAL(11), VAL(16), . . . , VAL(91), which were

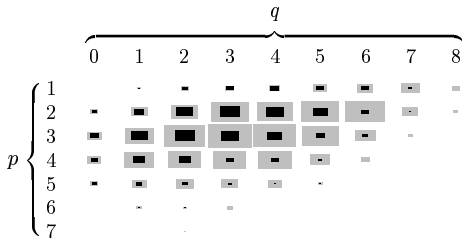
$$20 \ 21 \ 21 \ 21 \ 20 \ 15 \ 16 \ 8 \ 14 \ 20.$$

Thus 61 was true on level  $8 \gg 1 = 4$ ;  $\overline{46}$  and 66 were true on level  $15 \gg 1 = 14 \gg 1 = 7$ ;  $\overline{51}$  was false on level 8; the others were a mixture of true and false on level 10; hence (122) had  $p = 3$  and  $q = 1$  with respect to the current trail.

If  $C$  has signature  $(p, q)$  and  $C'$  has signature  $(p', q')$ , where  $p \leq p'$  and  $q \leq q'$  and  $(p, q) \neq (p', q')$ , we can expect that  $C$  is more likely than  $C'$  to be useful in future propagations. The same conclusion is plausible also when  $p + q = p' + q'$  and  $p < p'$ , because  $C'$  won’t force anything until literals from at least  $p + 1$  different levels change sign. These intuitive expectations are borne out by the following detailed data obtained from *waerden*(3, 10; 97):

$$\left( \begin{array}{cccccccc} 0 & 4 & 17 & 22 & 30 & 54 & 67 & 99 & 17 \\ 17 & 81 & 191 & 395 & 360 & 404 & 438 & 66 & 6 \\ 63 & 232 & 463 & 536 & 521 & 386 & 117 & 6 & 0 \\ 52 & 243 & 291 & 298 & 308 & 112 & 22 & 0 & 0 \\ 18 & 59 & 86 & 77 & 53 & 7 & 0 & 0 & 0 \\ 0 & 8 & 3 & 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \left( \begin{array}{cccccccc} 0 & 1 & 9 & 15 & 21 & 16 & 15 & 3 & 0 \\ 7 & 26 & 74 & 107 & 82 & 57 & 16 & 1 & 0 \\ 20 & 74 & 104 & 86 & 61 & 21 & 9 & 0 & 0 \\ 13 & 40 & 37 & 16 & 14 & 4 & 0 & 0 & 0 \\ 6 & 10 & 9 & 4 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right)$$

The matrix on the left shows how many of the 6240 eligible clauses had a given signature  $(p, q)$ , for  $1 \leq p \leq 7$  and  $0 \leq q \leq 8$ ; the matrix on the right shows how many would have been used to resolve future conflicts, if none of them had been removed. There were, for example, 536 learned clauses with  $p = q = 3$ , of which only 86 actually turned out to be useful. This data is illustrated graphically in Fig. 50, which shows gray rectangles whose areas correspond to the left matrix, overlaid by black rectangles whose areas correspond to the right matrix. We can’t predict the future, but small  $(p, q)$  tends to increase the ratio of black to gray.



**Fig. 50.** Learned clauses that have  $p$  positive and  $q$  all-negative levels. The gray ones will never be used again. Unfortunately, there’s no easy way to distinguish gray from black without being clairvoyant.

An alert reader will be wondering, however, how such signatures were found, because we can’t compute them for all clauses until all variables appear in the trail—and that doesn’t happen until all clauses are satisfied! The answer [see A. Goultiaeva and F. Bacchus, *LNCS 7317* (2012), 30–43] is that it’s quite possible to carry out a “full run” in which *every* variable is assigned a value, by making only a slight change to the normal behavior of Algorithm C: Instead

VAL  
Goultiaeva  
Bacchus  
full run

of resolving conflicts immediately and backjumping, we can carry on after each conflict until all propagations cease, and we can continue to build the trail in the same way until every variable is present on some level. Conflicts may have occurred on several different levels; but we can safely resolve them later, learning new clauses at that time. Meanwhile, a full trail allows us to compute signatures based on VAL fields. And those VAL fields go into the OVAL fields after backjumping, so the variables in each block will tend to maintain their relationships.

backjumping  
VAL  
OVAL  
author  
tie-breakers  
literal block distance  
ACT( $c$ )  
clause activity  
activity score

The author's implementation of Algorithm C assigns an eight-bit value

$$\text{RANGE}(c) \leftarrow \min(\lfloor 16(p + \alpha q) \rfloor, 255) \quad (123)$$

to each clause  $c$ ; here  $\alpha$  is a parameter,  $0 \leq \alpha \leq 1$ . We also set  $\text{RANGE}(c) \leftarrow 0$  if  $c$  is the reason for some literal in the trail;  $\text{RANGE}(c) \leftarrow 256$  if  $c$  is satisfied at level 0. If there are  $m_j$  clauses of range  $j$ , and if we want to keep at most  $T$  clauses in memory, we find the largest  $j \leq 256$  such that

$$m_j > 0 \quad \text{and} \quad s_j = m_0 + m_1 + \cdots + m_{j-1} \leq T. \quad (124)$$

Then we retain all clauses for which  $\text{RANGE}(c) < j$ , together with  $T - s_j$  "tie-breakers" that have  $\text{RANGE}(c) = j$  (unless  $j = 256$ ). When  $\alpha$  has the relatively high value  $\frac{15}{16} = .9375$ , this rule essentially preserves as many clauses of small literal block distance as it can; and for constant  $p + q$  it favors those with small  $p$ .

For example, with  $\alpha = \frac{15}{16}$  and the data from Fig. 50, we save clauses that have  $p = (1, 2, 3, 4, 5)$  when  $q \leq (5, 4, 3, 2, 0)$ , respectively. This gives us  $s_{95} = 12 + 3069$  clauses, just 45 shy of our target  $T = 3126$ . So we also choose 45 tie-breakers from among the 59 clauses that have  $\text{RANGE}(c) = 95$ ,  $(p, q) = (5, 1)$ .

Tie-breaking can be done by using a secondary heuristic  $\text{ACT}(c)$ , "clause activity," which is analogous to the activity score of a variable but it is more easily maintained. If clause  $c$  has been used to resolve the conflicts numbered 3, 47, 95, 99, and 100, say, then

$$\text{ACT}(c) = \varrho^{-3} + \varrho^{-47} + \varrho^{-95} + \varrho^{-99} + \varrho^{-100}. \quad (125)$$

This damping factor  $\varrho$  (normally .999) is independent of the factor  $\rho$  that is used for variable activities. In the case of Fig. 50, if the 59 clauses with  $(p, q) = (5, 1)$  are arranged in order of increasing ACT scores, the gray-and-black pattern is



So if we retain the 45 with highest activity, we pick up 8 of the 10 that turn out to be useful. (Clause activities are imperfect predictors, but they are usually somewhat better than this example implies.)

Exercises 287 and 288 present full details of clause purging in accordance with these ideas. One question remains: After we've completed a purge, when should we schedule the next one? Successful results are obtained by having two parameters,  $\Delta_p$  and  $\delta_p$ . Initially  $M_p = \Delta_p$ ; then after each purge, we set  $\Delta_p \leftarrow \Delta_p + \delta_p$  and  $M_p \leftarrow M_p + \Delta_p$ . For example, if  $\Delta_p = 10000$  and  $\delta_p = 100$ , purging will occur after approximately 10000, 20100, 30300, 40600,  $\dots$ ,  $k\Delta_p + \binom{k}{2}\delta_p$ ,

... clauses have been learned; and the number of clauses at the beginning of the  $k$ th round will be approximately  $20000 + 200k = 2\Delta_p + 2k\delta_p$ . (See exercise 289.)

We've based this discussion on *waarden*(3, 10; 97), which is quite a simple problem. Algorithm C's gain from clause-purging on larger problems is naturally much more substantial. For example, *waarden*(3, 13; 160) is only a bit larger than *waarden*(3, 10; 97). With  $\Delta_p = 10000$  and  $\delta_p = 100$ , it finishes in 132 gigamems, after learning 9.5 million clauses and occupying only 503 thousand MEM cells. Without purging, it proves unsatisfiability after learning only 7.1 million clauses, yet at well over ten times the cost: 4307 gigamems, and 102 million cells of MEM.

Flushing literals and restarting  
van der Tak  
Ramos  
Heule  
activity scores  
**ACT( $k$ )**  
phase-saving  
reusing the trail

**\*Flushing literals and restarting.** Algorithm C interrupts itself in step C5 not only to purge clauses but also to “flush literals” that may not have been the best choices for decisions in the trail. The task of solving a tough satisfiability problem is a delicate balancing act: We don't want to get bogged down in the wrong part of the search space; but we also don't want to lose the fruits of hard work by “throwing out the baby with the bath water.” A nice compromise has been found by Peter van der Tak, Antonio Ramos, and Marijn Heule [*J. Satisfiability, Bool. Modeling and Comp.* **7** (2011), 133–138], who devised a useful way to rejuvenate the trail periodically by following trends in the activity scores **ACT( $k$ )**.

Let's go back to Table 3, to illustrate their method. After learning the clause (116), Algorithm C will update the trail by setting  $L_{44} \leftarrow 57$  on level 17; that will force  $L_{45} \leftarrow \overline{66}$ , because 39, 42, ..., 63 have all become true; and further positive literals 6, 58, 82, 86, 95, 96 will also join the trail in some order. Step C5 might then intervene to suggest that we should contemplate flushing some or all of the  $F = 52$  literals whose values are currently assigned.

The decision literals  $\overline{53}$ , 55, 44, ..., 51 on levels 1, 2, 3, ..., 17 each were selected because they had the greatest current activity scores when their level began. But activity scores are continually being updated, so the old ones might be considerably out of touch with present realities. For example, we've just boosted **ACT(53)**, **ACT(27)**, **ACT(36)**, **ACT(70)**, ..., in the process of learning (116) — see (115). Thus it's quite possible that several of the first 17 decisions no longer seem wise, because those literals haven't participated in any recent conflicts.

Let  $x_k$  be a variable with maximum **ACT( $k$ )**, among all of the variables not in the current trail. It's easy to find such a  $k$  (see exercise 290). Now consider, as a thought experiment, what would happen if we were to jump back all the way to level 0 at this point and start over. Recall that our phase-saving strategy dictates that we would set **OVAL( $j$ )**  $\leftarrow$  **VAL( $j$ )** just before setting **VAL( $j$ )**  $\leftarrow$   $-1$ , as the variables become unassigned.

If we now restart at step C6 with  $d \leftarrow 1$ , all variables whose activity exceeds **ACT( $k$ )** will receive their former values (although not necessarily in the same order), because the corresponding literals will enter the trail either as decisions or as forced propagations. History will more or less repeat itself, because the old assignments did not cause any conflicts, and because phases were saved.

We might as well therefore avoid most of this back-and-forth unsettling and resetting, by reusing the trail and jumping back only partway, to the first level



where the current activity scores significantly change the picture:

$$\begin{aligned} \text{Set } d' \leftarrow 0. \text{ While } \text{ACT}(|L_{i_{d'+1}}|) \geq \text{ACT}(k), \text{ set } d' \leftarrow d' + 1. \\ \text{Then if } d' < d, \text{ jump back to level } d'. \end{aligned} \tag{126}$$

This is the technique called “literal flushing,” because it removes the literals on levels  $d' + 1$  through  $d$  and leaves the others assigned. It effectively redirects the search into new territory, without being as drastic as a full restart.

In Table 3, for example,  $\text{ACT}(49)$  might exceed the activity score of every other unassigned variable; and it might also exceed  $\text{ACT}(46)$ , the activity of the decision literal  $\overline{46}$  on level 15. If the previous 14 decision-oriented activities  $\text{ACT}(53), \text{ACT}(55), \dots, \text{ACT}(37)$  are all  $\geq \text{ACT}(49)$ , we would flush all the literals  $L_{25}, L_{26}, \dots$  above level  $d' = 14$ , and commence a new level 15.

Notice that some of the flushed literals other than  $\overline{46}$  might actually have the largest activities of all. In such cases they will re-insert themselves, before 49 ever enters the scene. Eventually, though, the literal 49 will inaugurate a new level before a new conflict arises. (See exercise 291.)

Experience shows that flushing can indeed be extremely helpful. On the other hand, it can be harmful if it causes us to abandon a fruitful line of attack. When the solver is perking along and learning useful clauses by the dozen, we don't want to upset the applecart by rocking the boat. Armin Biere has therefore introduced a useful statistic called *agility*, which tends to be correlated with the desirability of flushing at any given moment. His idea [LNCS 4996 (2008), 28–33] is beautifully simple: We maintain a 32-bit integer variable called **AGILITY**, initially zero. Whenever a literal  $l$  is placed on the trail in steps C4, C6, or C9, we update the agility by setting

$$\text{AGILITY} \leftarrow \text{AGILITY} - (\text{AGILITY} \gg 13) + (((\text{OVAL}(|l|) - \text{VAL}(|l|)) \& 1) \ll 19). \tag{127}$$

In other words, the fraction  $\text{AGILITY}/2^{32}$  is essentially multiplied by  $1 - \delta$ , then increased by  $\delta$  if the new polarity of  $l$  differs from its previous polarity, where  $\delta = 2^{-13} \approx .0001$ . High agility means that lots of the recent propagations are flipping the values of variables and trying new possibilities; low agility means that the algorithm is basically in a rut, spinning its wheels and getting nowhere.

**Table 4**

TO FLUSH OR NOT TO FLUSH?

| Let $a = \text{AGILITY}/2^{32}$ when setting $M_f \leftarrow M + \Delta_f$ , and let $\psi = 1/6, \theta = 17/16$ . |                                    |                  |                                    |                  |                                       |
|---|------------------------------------|------------------|------------------------------------|------------------|---------------------------------------|
| If $\Delta_f$ is  | then flush if                      | If $\Delta_f$ is | then flush if                      | If $\Delta_f$ is | then flush if                         |
| 1   | $a \leq \psi \approx .17$          | 32               | $a \leq \theta^5 \psi \approx .23$ | 1024             | $a \leq \theta^{10} \psi \approx .31$ |
| 2   | $a \leq \theta \psi \approx .18$   | 64               | $a \leq \theta^6 \psi \approx .24$ | 2048             | $a \leq \theta^{11} \psi \approx .32$ |
| 4   | $a \leq \theta^2 \psi \approx .19$ | 128              | $a \leq \theta^7 \psi \approx .25$ | 4096             | $a \leq \theta^{12} \psi \approx .34$ |
| 8   | $a \leq \theta^3 \psi \approx .20$ | 256              | $a \leq \theta^8 \psi \approx .27$ | 8192             | $a \leq \theta^{13} \psi \approx .37$ |
| 16  | $a \leq \theta^4 \psi \approx .21$ | 512              | $a \leq \theta^9 \psi \approx .29$ | 16384            | $a \leq \theta^{14} \psi \approx .39$ |

Armed with the notion of agility, we can finally state what Algorithm C does when step C5 finds  $M \geq M_f$ : First  $M_f$  is reset to  $M + \Delta_f$ , where  $\Delta_f$  is

literal flushing  
activity score  
clichés  
Biere  
agility  
damping factor  
polarity

a power of two determined by the “reluctant doubling” sequence  $\langle 1, 1, 2, 1, 1, 2, 4, 1, \dots \rangle$ ; that sequence is discussed below and in exercise 293. Then the agility is compared to a threshold, depending on  $\Delta_f$ , according to the schedule in Table 4. (The parameter  $\psi$  in that table can be raised or lowered, if you want to increase or decrease the amount of flushing.) If the agility is sufficiently small,  $x_k$  is found and (126) is performed. Nothing changes if the agility is large or if  $d' = d$ ; otherwise (126) has flushed some literals, using the operations of step C8.

reluctant doubling  
 heuristic  
 randomized methods  
 stochastic local search  
 SLS: Stochastic local search  
 Gu  
 Papadimitriou  
 debug  
 random walks–  
 Papadimitriou

**Monte Carlo methods.** Let’s turn now to a completely different way to approach satisfiability problems, based on finding solutions by totally heuristic and randomized methods, often called *stochastic local search*. We often use such methods in our daily lives, even though there’s no guarantee of success. The simplest satisfiability-oriented technique of this kind was introduced by Jun Gu [see *SIGART Bulletin* **3**,1 (January 1992), 8–12] and by Christos Papadimitriou [*FOCS* **32** (1991), 163–169] as a byproduct of more general studies:

“Start with any truth assignment. While there are unsatisfied clauses, pick any one, and flip a random literal in it.”

Some programmers are known to debug their code in a haphazard manner, somewhat like this approach; and we know that such “blind” changes are foolish because they usually introduce new bugs. Yet this idea does have merit when it is applied to satisfiability, so we shall formulate it as an algorithm:

**Algorithm P** (*Satisfiability by random walk*). Given  $m$  nonempty clauses  $C_1 \wedge \dots \wedge C_m$  on  $n$  Boolean variables  $x_1 \dots x_n$ , this algorithm either finds a solution or terminates unsuccessfully after making  $N$  trials.

- P1.** [Initialize.] Assign random Boolean values to  $x_1 \dots x_n$ . Set  $j \leftarrow 0$ ,  $s \leftarrow 0$ , and  $t \leftarrow 0$ . (We know that  $s$  clauses are satisfied after having made  $t$  flips.)
- P2.** [Success?] If  $s = m$ , terminate successfully with solution  $x_1 \dots x_n$ . Otherwise set  $j \leftarrow (j \bmod m) + 1$ . If clause  $C_j$  is satisfied by  $x_1 \dots x_n$ , set  $s \leftarrow s + 1$  and repeat this step.
- P3.** [Done?] If  $t = N$ , terminate unsuccessfully.
- P4.** [Flip one bit.] Let clause  $C_j$  be  $(l_1 \vee \dots \vee l_k)$ . Choose a random index  $i \in \{1, \dots, k\}$ , and change variable  $|l_i|$  so that literal  $l_i$  becomes true. Set  $s \leftarrow s - 1$ ,  $t \leftarrow t + 1$ , and return to P2. ■

Suppose, for example, that we’re given the seven clauses  $R'$  of (7). Thus  $m = 7$ ,  $n = 4$ ; and there are two solutions, 01\*1. In this case every nonsolution violates a unique clause; for example, 1100 violates the clause  $\bar{1}\bar{2}3$ , so step P4 is equally likely to change 1100 to 0100, 1000, or 1110, only one of which is closer to a solution. An exact analysis (see exercise 294) shows that Algorithm P will find a solution after making 8.25 flips, on the average. That’s no improvement over a brute-force search through all  $2^n = 16$  possibilities; but a small example like this doesn’t tell us much about what happens when  $n$  is large.

Papadimitriou observed that Algorithm P is reasonably effective when it’s applied to 2SAT problems, because each flip has roughly a 50-50 chance of making

progress in that case. Several years later, Uwe Schöning [Algorithmica **32** (2002), 615–623] discovered that the algorithm also does surprisingly well on instances of 3SAT, even though the flips when  $k > 2$  in step P4 tend to go “the wrong way”:

Schöning  
ballot number

**Theorem U.** *If the given clauses are satisfiable, and if each clause has at most three literals, Algorithm P will succeed with probability  $\Omega((3/4)^n/n)$  after making at most  $n$  flips.*

*Proof.* By complementing variables, if necessary, we can assume that  $0\dots 0$  is a solution; under this assumption, every clause has at least one negative literal. Let  $X_t = x_1 + \dots + x_n$  be the number of 1s after  $t$  flips have been made. Each flip changes  $X_t$  by  $\pm 1$ , and we want to show that there’s a nontrivial chance that  $X_t$  will become 0. After step P1, the random variable  $X_0$  will be equal to  $q$  with probability  $\binom{n}{q}/2^n$ .

A clause that contains three negative literals is good news for Algorithm P, because it is violated only when all three variables are 1; a flip will *always* decrease  $X_t$  in such a case. Similarly, a violated clause with two negatives and one positive will invoke a flip that makes progress  $2/3$  of the time. The worst case occurs only when a problematic clause has only one negative literal. Unfortunately, every clause might belong to this worst case, for all we know.

Instead of studying  $X_t$ , which depends on the pattern of clauses, it’s much easier to study another random variable  $Y_t$  defined as follows: Initially  $Y_0 = X_0$ ; but  $Y_{t+1} = Y_t - 1$  only when step P4 flips the negative literal that has the smallest subscript; otherwise  $Y_{t+1} = Y_t + 1$ . For example, after taking care of a violated clause such as  $x_3 \vee \bar{x}_5 \vee \bar{x}_8$ , we have  $X_{t+1} = X_t + (+1, -1, -1)$  but  $Y_{t+1} = Y_t + (+1, -1, +1)$  in the three possible cases. Furthermore, if the clause contains fewer than three literals, we penalize  $Y_{t+1}$  even more, by allowing it to be  $Y_t - 1$  only with probability  $1/3$ . (After a clause such as  $x_4 \vee \bar{x}_6$ , for instance, we put  $Y_{t+1} = Y_t - 1$  in only  $2/3$  of the cases when  $x_6$  is flipped; otherwise  $Y_{t+1} = Y_t + 1$ .)

We clearly have  $X_t \leq Y_t$  for all  $t$ . Therefore  $\Pr(X_t = 0) \geq \Pr(Y_t = 0)$ , after  $t$  flips have been made; and we’ve defined things so that it’s quite easy to calculate  $\Pr(Y_t = 0)$ , because  $Y_t$  doesn’t depend on the current clause  $j$ :

$$\Pr(Y_{t+1} = Y_t - 1) = 1/3 \quad \text{and} \quad \Pr(Y_{t+1} = Y_t + 1) = 2/3 \quad \text{when } Y_t > 0.$$

Indeed, the theory of random walks developed in Section 7.2.1.6 tells us how to count the number of scenarios that begin with  $Y_0 = q$  and end with  $Y_t = 0$ , after  $Y_t$  has increased  $p$  times and decreased  $p + q$  times while remaining positive for  $0 \leq t < 2p + q$ . It is the “ballot number” of Eq. 7.2.1.6–(23),

$$C_{p,p+q-1} = \frac{q}{2p+q} \binom{2p+q}{p}. \quad (128)$$

The probability that  $Y_0 = q$  and that  $Y_t = 0$  for the first time when  $t = 2p + q$  is therefore exactly

$$f(p, q) = \frac{1}{2^n} \binom{n}{q} \frac{q}{2p+q} \binom{2p+q}{p} \left(\frac{1}{3}\right)^{p+q} \left(\frac{2}{3}\right)^p. \quad (129)$$

Every value of  $p$  and  $q$  gives a lower bound for the probability that Algorithm P succeeds; and exercise 296 shows that we get the result claimed in Theorem U by choosing  $p = q \approx n/3$ . ■

Theorem U might seem pointless, because it predicts success only with exponentially small probability when  $N = n$ . But if at first we don't succeed, we can try and try again, by repeating Algorithm P with different random choices. And if we repeat it  $Kn(4/3)^n$  times, for large enough  $K$ , we're almost certain to find a solution unless the clauses can't all be satisfied.

In fact, even more is true, because the proof of Theorem U doesn't exploit the full power of Eq. (129). Exercise 297 carries the analysis further, in a particularly instructive way, and proves a much sharper result:

**Corollary W.** *When Algorithm P is applied  $K(4/3)^n$  times with  $N = 2n$  to a set of satisfiable ternary clauses, its success probability exceeds  $1 - e^{-K/2}$ .* ■

If the clauses  $C_1 \wedge \dots \wedge C_m$  are unsatisfiable, Algorithm P will never demonstrate that fact conclusively. But if we repeat it  $100(4/3)^n$  times and get no solution, Corollary W tells us that the chances of satisfiability are incredibly small (less than  $10^{-21}$ ). So it's a safe bet that no solution exists in such a case.

Thus Algorithm P has a surprisingly good chance of finding solutions “with its eyes closed,” while walking at random in the gigantic space of all  $2^n$  binary vectors; and we can well imagine that even better results are possible if we devise randomized walking methods that proceed with eyes wide open. Therefore many people have experimented with strategies that try to make intelligent choices about which direction to take at each flip-step. One of the simplest and best of these improvements, popularly known as WalkSAT, was devised by B. Selman, H. A. Kautz, and B. Cohen [Nat. Conf. Artificial Intelligence 12 (1994), 337–343]:

**Algorithm W (WalkSAT).** Given  $m$  nonempty clauses  $C_1 \wedge \dots \wedge C_m$  on  $n$  Boolean variables  $x_1 \dots x_n$ , and a “greed-avoidance” parameter  $p$ , this algorithm either finds a solution or terminates unsuccessfully after making  $N$  trials. It uses auxiliary arrays  $c_1 \dots c_n$ ,  $f_0 \dots f_{m-1}$ ,  $k_1 \dots k_m$ , and  $w_1 \dots w_m$ .

- W1.** [Initialize.] Assign random Boolean values to  $x_1 \dots x_n$ . Also set  $r \leftarrow t \leftarrow 0$  and  $c_1 \dots c_n \leftarrow 0 \dots 0$ . Then, for  $1 \leq j \leq m$ , set  $k_j$  to the number of true literals in  $C_j$ ; and if  $k_j = 0$ , set  $f_r \leftarrow j$ ,  $w_j \leftarrow r$ , and  $r \leftarrow r + 1$ ; or if  $k_j = 1$  and the only true literal of  $C_j$  is  $x_i$  or  $\bar{x}_i$ , set  $c_i \leftarrow c_i + 1$ . (Now  $r$  is the number of unsatisfied clauses, and the  $f$  array lists them. The number  $c_i$  is the “cost” or “break count” for variable  $x_i$ , namely the number of additional clauses that will become false if  $x_i$  is flipped.)
- W2.** [Done?] If  $r = 0$ , terminate successfully with solution  $x_1 \dots x_n$ . Otherwise, if  $t = N$ , terminate unsuccessfully.
- W3.** [Choose  $j$ .] Set  $j \leftarrow f_q$ , where  $q$  is uniformly random in  $\{0, 1, \dots, r - 1\}$ . (In other words, choose an unsatisfied clause  $C_j$  at random, considering every such clause to be equally likely; exercise 3.4.1–3 discusses the best way to compute  $q$ .) Let clause  $C_j$  be  $(l_1 \vee \dots \vee l_k)$ .

n-cube  
WalkSAT-  
Selman  
Kautz  
Cohen  
break count

**W4.** [Choose  $l$ .] Let  $c$  be the smallest cost among the literals  $\{l_1, \dots, l_k\}$ . If  $c = 0$ , or if  $c \geq 1$  and  $U \geq p$  where  $U$  is uniform in  $[0..1)$ , choose  $l$  randomly from among the literals of cost  $c$ . (We call this a “greedy” choice, because flipping  $l$  will minimize the number of newly false clauses.) Otherwise choose  $l$  randomly in  $\{l_1, \dots, l_k\}$ .

**W5.** [Flip  $l$ .] Change the value of variable  $|l|$ , and update  $r, c_1 \dots c_n, f_0 \dots f_{r-1}, k_1 \dots k_m, w_1 \dots w_m$  to agree with this new value. (Exercise 302 explains how to implement steps W4 and W5 efficiently, with computer-friendly changes to the data structures.) Set  $t \leftarrow t + 1$  and return to W2. ■

Seitz  
Alava  
Orponen  
3SAT  
2SAT  
reluctant doubling  
Luby  
Sinclair  
Zuckerman

If, for example, we try to satisfy the seven clauses of (7) with Algorithm W, as we did earlier with Algorithm P, the choice  $x_1 x_2 x_3 x_4 = 0110$  violates  $\bar{2}34$ ; and  $c_1 c_2 c_3 c_4$  turns out to be 0110 in this situation. So step W4 will choose to flip  $x_4$ , and we’ll have the solution 0111. (See exercise 303.)

Notice that step W3 focuses attention on variables that need to change. Furthermore, a literal that appears in the most unsatisfied clauses is most likely to appear in the chosen clause  $C_j$ .

If no cost-free flip is available, step W4 makes nongreedy choices with probability  $p$ . This policy keeps the algorithm from getting stuck in an unsatisfiable region from which there’s no greedy exit. Extensive experiments by S. Seitz, M. Alava, and P. Orponen [*J. Statistical Mechanics* (June 2005), P06006:1–27] indicate that the best choice of  $p$  is .57 when large random 3SAT problems are being tackled. For example, with this setting of  $p$ , and with  $m = 4.2n$  random 3-literal clauses, Algorithm W works fantastically well: It tends to find solutions after making fewer than  $10,000n$  flips when  $n = 10^4$ , and fewer than  $2500n$  flips when  $10^5 \leq n \leq 10^6$ .

What about the parameter  $N$ ? Should we set it equal to  $2n$  (as recommended for 3SAT problems with respect to Algorithm P), or perhaps to  $n^2$  (as recommended for 2SAT in exercise 299), or to  $2500n$  (as just mentioned for 3SAT in Algorithm W), or to something else? When we use an algorithm like WalkSAT, whose behavior can vary wildly depending on random choices and on unknown characteristics of the data, it’s often wise to “cut our losses” and to start afresh with a brand new pattern of random numbers.

Exercise 306 proves that such an algorithm always has an optimum cutoff value  $N = N^*$ , which minimizes the expected time to success when the algorithm is restarted after each failure. Sometimes  $N^* = \infty$  is the best choice, meaning that we should always keep plowing ahead; in other cases  $N^*$  is quite small.

But  $N^*$  exists only in theory, and the theory requires perfect knowledge of the algorithm’s behavior. In practice we usually have little or no information about how  $N$  should best be specified. Fortunately there’s still an effective way to proceed, by using the notion of *reluctant doubling* introduced by M. Luby, A. Sinclair, and D. Zuckerman [*Information Proc. Letters* **47** (1993), 173–180], who defined the interesting sequence

$$S_1, S_2, \dots = 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, \dots \quad (130)$$

The elements of this sequence are all powers of 2. Furthermore we have  $S_{n+1} = 2S_n$  if the number  $S_n$  has already occurred an even number of times, otherwise  $S_{n+1} = 1$ . A convenient way to generate this sequence is to work with two integers  $(u, v)$ , and to start with  $(u_1, v_1) = (1, 1)$ ; then

$$(u_{n+1}, v_{n+1}) = (u_n \& -u_n = v_n? (u_n + 1, 1): (u_n, 2v_n)). \quad (131)$$

The successive pairs are  $(1, 1), (2, 1), (2, 2), (3, 1), (4, 1), (4, 2), (4, 4), (5, 1), \dots$ , and we have  $S_n = v_n$  for all  $n \geq 1$ .

The reluctant doubling strategy is to run Algorithm W repeatedly with  $N = cS_1, cS_2, cS_3, \dots$ , until success is achieved, where  $c$  is some constant. Exercise 308 proves that the expected running time  $X$  obtained in this way exceeds the optimum by at most a factor of  $O(\log X)$ . Other sequences besides  $\langle S_n \rangle$  also have this property, and they're sometimes better (see exercise 311). The best policy is probably to use  $\langle cS_n \rangle$ , where  $c$  represents our best guess about the value of  $N^*$ ; in this way we hedge our bets in case  $c$  is too small.

**The Local Lemma.** The existence of particular combinatorial patterns is often established by using a nonconstructive proof technique called the “probabilistic method,” pioneered by Paul Erdős. If we can show that  $\Pr(X) > 0$ , in some probability space, then  $X$  must be true in at least one case. For example [*Bull. Amer. Math. Soc.* **53** (1947), 292–294], Erdős famously observed that there is a graph  $G$  on  $n$  vertices such that neither  $G$  nor  $\overline{G}$  contains a  $k$ -clique, whenever

$$\binom{n}{k} < 2^{k(k-1)/2-1}. \quad (132)$$

For if we consider a random graph  $G$ , each of whose  $\binom{n}{2}$  edges is present with probability  $1/2$ , and if  $U$  is any particular subset of  $k$  vertices in  $G$ , the probability that either  $G|U$  or  $\overline{G}|U$  is a complete graph is clearly  $2/2^{k(k-1)/2}$ . Hence the probability that this doesn't happen for any of the  $\binom{n}{k}$  subsets  $U$  is at least  $1 - \binom{n}{k}2^{1-k(k-1)/2}$ . This probability is positive; so such a graph must exist.

The proof just given does not provide any explicit construction. But it does show that we can find such a graph by making at most  $1/(1 - \binom{n}{k}2^{1-k(k-1)/2})$  random trials, on the average, provided that  $n$  and  $k$  are small enough that we are able to test all  $\binom{n}{k}$  subgraphs in a reasonable amount of time.

Probability calculations of this kind are often complicated by dependencies between the random events being considered. For example, the presence of a clique in one part of a graph affects the likelihood of many other cliques that share some of the same vertices. But the interdependencies are often highly localized, so that “remote” events are essentially independent of each other. László Lovász introduced an important way to deal with such situations early in the 1970s, and his approach has become known as the “Local Lemma” because it has been used to establish many theorems. First published as a lemma on pages 616–617 of a longer paper [Erdős and Lovász, *Infinite and Finite Sets, Colloquia Math. Soc. János Bolyai* **10** (1975), 609–627], and subsequently extended to a “lopsided” form [P. Erdős and J. Spencer, *Discrete Applied Math.* **30** (1991), 151–154], it can be stated as follows:

restart schedule  
Local Lemma–  
probabilistic method  
Erdős  
clique  
random graph  
Ramsey's theorem  
Lovász  
Erdős  
Lovász  
Spencer

**Lemma L.** Let  $A_1, \dots, A_m$  be events in some probability space. Let  $G$  be a graph on vertices  $\{1, \dots, m\}$ , and let  $(p_1, \dots, p_m)$  be numbers such that

$$\Pr(A_i \mid \bar{A}_{j_1} \cap \dots \cap \bar{A}_{j_k}) \leq p_i \text{ whenever } k \geq 0 \text{ and } i \not\sim j_1, \dots, i \not\sim j_k. \quad (133)$$

Then  $\Pr(\bar{A}_1 \cap \dots \cap \bar{A}_m) > 0$  whenever  $(p_1, \dots, p_m)$  lies in a certain set  $\mathcal{R}(G)$ . ■

In applications we think of the  $A_j$  as “bad” events, which are undesirable conditions that interfere with whatever we’re trying to find. The graph  $G$  is called a “lopsidependency graph” for our application; this name was coined as an extension of Lovász’s original term “dependency graph,” for which the strict condition ‘ $= p_i$ ’ was assumed in place of ‘ $\leq p_i$ ’ in (133).

The set  $\mathcal{R}(G)$  of probability bounds for which we can guarantee that all bad events can simultaneously be avoided, given (133), will be discussed further below. If  $G$  is the complete graph  $K_m$ , so that (133) simply states that  $\Pr(A_i) \leq p_i$ ,  $\mathcal{R}(G)$  is clearly  $\{(p_1, \dots, p_m) \mid (p_1, \dots, p_m) \geq (0, \dots, 0) \text{ and } p_1 + \dots + p_m < 1\}$ ; this is the smallest possible  $\mathcal{R}(G)$ . At the other extreme, if  $G$  is the empty graph  $\bar{K}_m$ , we get  $\{(p_1, \dots, p_m) \mid 0 \leq p_j < 1 \text{ for } 1 \leq j \leq m\}$ , the largest possible  $\mathcal{R}(G)$ . Adding an edge to  $G$  makes  $\mathcal{R}(G)$  smaller. Notice that, if  $(p_1, \dots, p_m)$  is in  $\mathcal{R}(G)$  and  $0 \leq p'_j \leq p_j$  for  $1 \leq j \leq m$ , then also  $(p'_1, \dots, p'_m) \in \mathcal{R}(G)$ .

Lovász discovered an elegant local condition that suffices to make Lemma L widely applicable [see J. Spencer, *Discrete Math.* **20** (1977), 69–76]:

**Theorem L.** The probability vector  $(p_1, \dots, p_m)$  is in  $\mathcal{R}(G)$  when there are numbers  $0 \leq \theta_1, \dots, \theta_m < 1$  such that

$$p_i = \theta_i \prod_{i-j \text{ in } G} (1 - \theta_j). \quad (134)$$

*Proof.* Exercise 344(e) proves that  $\Pr(\bar{A}_1 \cap \dots \cap \bar{A}_m) \geq (1 - \theta_1) \dots (1 - \theta_m)$ . ■

James B. Shearer [*Combinatorica* **5** (1985), 241–245] went on to determine the exact maximum extent of  $\mathcal{R}(G)$  for all graphs  $G$ , as we’ll see later; and he also established the following important special case:

**Theorem J.** Suppose every vertex of  $G$  has degree  $\leq d$ , where  $d > 1$ . Then  $(p, \dots, p) \in \mathcal{R}(G)$  when  $p \leq (d - 1)^{d-1}/d^d$ .

*Proof.* See the interesting inductive argument in exercise 317. ■

This condition on  $p$  holds whenever  $p \leq 1/(ed)$  (see exercise 319).

Further study led to a big surprise: The Local Lemma proves only that desirable combinatorial patterns *exist*, although they might be rare. But Robin Moser and Gábor Tardos discovered [*JACM* **57** (2010), 11:1–11:15] that we can efficiently *compute* a pattern that avoids all of the bad  $A_j$ , using an almost unbelievably simple algorithm analogous to WalkSAT!

**Algorithm M (Local resampling).** Given  $m$  events  $\{A_1, \dots, A_m\}$  that depend on  $n$  Boolean variables  $\{x_1, \dots, x_n\}$ , this algorithm either finds a vector  $x_1 \dots x_n$  for which none of the events is true, or loops forever. We assume that  $A_j$  is a function of the variables  $\{x_k \mid k \in \Xi_j\}$  for some given subset  $\Xi_j \subseteq \{1, \dots, n\}$ .

$\mathcal{R}(G)$   
lopsidependency graph  
dependency graph  
Lovász  
Spencer  
Shearer  
Moser  
Tardos

Whenever the algorithm assigns a value to  $x_k$ , it sets  $x_k \leftarrow 1$  with probability  $\xi_k$  and  $x_k \leftarrow 0$  with probability  $1 - \xi_k$ , where  $\xi_k$  is another given parameter.

**M1.** [Initialize.] For  $1 \leq k \leq n$ , set  $x_k \leftarrow [U < \xi_k]$ , where  $U$  is uniform in  $[0..1)$ .

**M2.** [Choose  $j$ .] Set  $j$  to the index of any event such that  $A_j$  is true. If no such  $j$  exists, terminate successfully, having found a solution  $x_1 \dots x_n$ .

**M3.** [Resample for  $A_j$ .] For each  $k \in \Xi_j$ , set  $x_k \leftarrow [U < \xi_k]$ , where  $U$  is uniform in  $[0..1)$ . Return to M2. ■

(We have stated Algorithm M in terms of binary variables  $x_k$  purely for convenience. The same ideas apply when each  $x_k$  has a discrete probability distribution on *any* set of values, possibly different for each  $k$ .)

To tie this algorithm to the Local Lemma, we assume that event  $A_i$  holds with probability  $\leq p_i$  whenever the variables it depends on have the given distribution. For example, if  $A_i$  is the event “ $x_3 \neq x_5$ ” then  $p_i$  must be at least  $\xi_3(1 - \xi_5) + (1 - \xi_3)\xi_5$ .

We also assume that there’s a graph  $G$  on vertices  $\{1, \dots, m\}$  such that condition (133) is true, and that  $i - j$  whenever  $i \neq j$  and  $\Xi_i \cap \Xi_j \neq \emptyset$ . Then  $G$  is a suitable dependency graph for  $\{A_1, \dots, A_m\}$ , because the events  $A_{j_1}, \dots, A_{j_k}$  can’t possibly influence  $A_i$  when  $i \not- j_1, \dots, i \not- j_k$ . (Those events share no common variables with  $A_i$ .) We can also sometimes get by with fewer edges by making  $G$  a *lopsidependency* graph; see exercise 351.

Algorithm M might succeed with *any* given events, purely by chance. But if the conditions of the Local Lemma are satisfied, success can be guaranteed:

**Theorem M.** *If (133) holds with probabilities that satisfy condition (134) of Theorem L, step M3 is performed for  $A_j$  at most  $\theta_j/(1 - \theta_j)$  times, on average.*

*Proof.* Exercise 352 shows that this result is a corollary of the more general analysis that is carried out below. The stated upper bound is good news, because  $\theta_j$  is usually quite small. ■

**Traces and pieces.** The best way to understand why Algorithm M is so efficient is to view it algebraically in terms of “traces.” The theory of traces is a beautiful area of mathematics in which amazingly simple proofs of profound results have been discovered. Its basic ideas were first formulated by P. Cartier and D. Foata [*Lecture Notes in Math.* **85** (1969)], then independently developed from another point of view by R. M. Keller [*JACM* **20** (1973), 514–537, 696–710] and A. Mazurkiewicz [“Concurrent program schemes and their interpretations,” DAIMI Report PB 78 (Aarhus University, July 1977)]. Significant advances were made by G. X. Viennot [*Lecture Notes in Math.* **1234** (1985), 321–350], who presented many wide-ranging applications and explained how the theory could readily be visualized in terms of what he called “heaps of pieces.”

Trace theory is the study of algebraic products whose variables are not necessarily commutative. Thus it forms a bridge between the study of strings (in which, for example, *acbbaca* is quite distinct from *baccaab*) and the study of ordinary commutative algebra (in which both of those examples are equal to

reliability polynomial  
lopsidependency  
traces–  
Cartier  
Foata  
Keller  
Mazurkiewicz  
Viennot  
heaps of pieces  
commutativity, partial  
strings



$aaabbcc = a^3b^2c^2$ ). Each adjacent pair of letters  $\{a, b\}$  either *commutes*, meaning that  $ab = ba$ , or *clashes*, meaning that  $ab$  is different from  $ba$ . If, for instance, we specify that  $a$  commutes with  $c$  but that  $b$  clashes with both  $a$  and  $c$ , then  $acbbaca$  is equal to  $cabbaac$ , and it has six variants altogether; similarly, there are ten equally good ways to write  $baccaab$ .

Formally speaking, a *trace* is an equivalence class of strings that can be converted to each other by repeatedly interchanging pairs of adjacent letters that don't clash. But we don't need to fuss about the fact that equivalence classes are present; we can simply represent a trace by any one of its equivalent strings, just as we don't distinguish between equivalent fractions such as  $1/2$  and  $3/6$ .

Every graph whose vertices represent distinct letters defines a family of traces on those letters, when we stipulate that two letters clash if and only if they are adjacent in the graph. For example, the path graph  $a - b - c$  corresponds to the rules stated above. The distinct traces for this graph are

$$\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, cb, cc, aaa, aab, \dots, ccb, ccc, aaaa, \dots \quad (135)$$

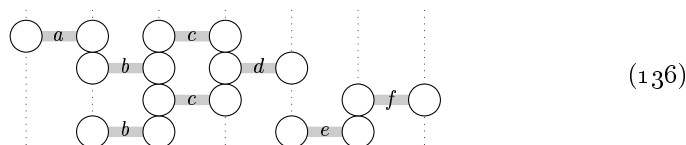
if we list them first by size and then in lexicographic order. (Notice that  $ca$  is absent, because  $ac$  has already appeared.) The complete graph  $K_n$  defines traces that are the same as strings, when *nothing* commutes; the empty graph  $\overline{K}_n$  defines traces that are the same as monomials, when *everything* commutes. If we use the path  $a - b - c - d - e - f$  to define clashes, the traces  $bcebafdc$  and  $efcdbca$  turn out to be the same.

Viennot observed that partial commutativity is actually a familiar concept, if we regard the letters as "pieces" that occupy "territory." Pieces clash if and only if their territories overlap; pieces commute if and only if their territories are disjoint. A trace corresponds to stacking the pieces on top of one another, from left to right, letting each new piece "fall" until it either rests on the ground or on another piece. In the latter case, it must rest on the most recent piece with which it clashes. He called this configuration an *empilement* — a nice French word.

More precisely, each piece  $a$  is assigned a nonempty subset  $T(a)$  of some universe, and we say that  $a$  clashes with  $b$  if and only if  $T(a) \cap T(b) \neq \emptyset$ . For example, the constraints of the graph  $a - b - c - d - e - f$  arise when we let

$$T(a) = \{1, 2\}, \quad T(b) = \{2, 3\}, \quad T(c) = \{3, 4\}, \quad \dots, \quad T(f) = \{6, 7\};$$

then the traces  $bcebafdc$  and  $efcdbca$  both have



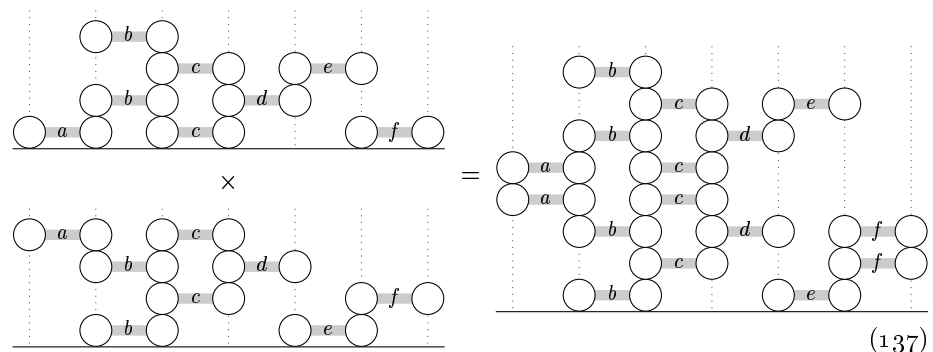
as their empilement. (Readers who have played the game of Tetris® will immediately understand how such diagrams are formed, although the pieces in trace theory differ from those of Tetris because they occupy only a single horizontal level. Furthermore, each type of piece always falls in exactly the same place; and a piece's territory  $T(a)$  might have "holes" — it needn't be connected.)

clashing pairs of letters  
equivalence class  
path graph  
lexicographic order  
Viennot  
territory  
pieces-  
territories  
stacking the pieces  
empilement  
intersection graph  
diagram of a trace  
Tetris

Two traces are the same if and only if they have the same empilement. In fact, the diagram implicitly defines a partial ordering on the pieces that appear; and the number of different strings that represent any given trace is the number of ways to sort that ordering topologically (see exercise 324).

Every trace  $\alpha$  has a *length*, denoted by  $|\alpha|$ , which is the number of letters in any of its equivalent strings. It also has a *height*, written  $h(\alpha)$ , which is the number of levels in its empilement. For example,  $|bcebafdc| = 8$  and  $h(bcebafdc) = 4$ .

**Arithmetic on traces.** To multiply traces, we simply concatenate them. If, for example,  $\alpha = bcebafdc$  is the trace corresponding to (136), then  $\alpha\alpha^R = bcebafdccdfabecb$  has the following empilement:



topological sorting  
length of a trace  
height of a trace  
multiplication of traces  
division of traces  
right division of traces  
polynomials  
generating function  
empty string

The algorithm in exercise 327 formulates this procedure precisely. A moment's thought shows that  $|\alpha\beta| = |\alpha| + |\beta|$ ,  $h(\alpha\beta) \leq h(\alpha) + h(\beta)$ , and  $h(\alpha\alpha^R) = 2h(\alpha)$ .

Traces can also be *divided*, in the sense that  $\alpha = (\alpha\beta)/\beta$  can be determined uniquely when  $\alpha\beta$  and  $\beta$  are given. All we have to do is remove the pieces of  $\beta$  from the pieces of  $\alpha\beta$ , one by one, working our way down from the top of the empilements. Similarly, the value of  $\beta = \alpha \setminus (\alpha\beta)$  can be computed from the traces  $\alpha$  and  $\alpha\beta$ . (See exercises 328 and 329.)

Notice that we could rotate diagrams like (136) and (137) by 90 degrees, thereby letting the pieces “fall” to the *left* instead of downwards. (We’ve used a left-to-right approach for similar purposes in Section 5.3.4, Fig. 50.) Or we could let them fall upwards, or to the right. Different orientations are sometimes more natural, depending on what we’re trying to do.

We can also add and subtract traces, thereby obtaining polynomials in variables that are only partially commutative. Such polynomials can be multiplied in the normal way; for example,  $(\alpha + \beta)(\gamma - \delta) = \alpha\gamma - \alpha\delta + \beta\gamma - \beta\delta$ . Indeed, we can even work with *infinite* sums, at least formally: The generating function for all traces that belong to the graph  $a - b - c$  is

$$1 + a + b + c + aa + ab + ac + ba + bb + bc + cb + cc + aaa + \dots + ccc + aaaa + \dots \quad (138)$$

(Compare with (135); we now use 1, not  $\epsilon$ , to stand for the empty string.)

The infinite sum (138) can actually be expressed in closed form: It equals

$$\frac{1}{1 - a - b - c + ac} = 1 + (a + b + c - ac) + (a + b + c - ac)^2 + \dots, \quad (139)$$

an identity that is correct not only when the variables are commutative, but also in the algebra of traces, when variables commute only when they don't clash.

In their original monograph of 1969, Cartier and Foata showed that the sum of all traces with respect to *any* graph can be expressed in a remarkably simple way that generalizes (139). Let's define the *Möbius function* of a trace  $\alpha$  with respect to a graph  $G$  by the rule

$$\mu_G(\alpha) = \begin{cases} 0, & \text{if } h_G(\alpha) > 1; \\ (-1)^{|\alpha|}, & \text{otherwise.} \end{cases} \tag{140}$$

(The classical Möbius function  $\mu(n)$  for integers, defined in exercise 4.5.2–10, is analogous.) Then the *Möbius series* for  $G$  is defined to be

$$M_G = \sum_{\alpha} \mu_G(\alpha)\alpha, \tag{141}$$

where the sum is over all traces. This sum is a polynomial, when  $G$  is finite, because it contains exactly one nonzero term for every independent set of vertices in  $G$ ; therefore we might call it the *Möbius polynomial*. For example, when  $G$  is the path  $a - b - c$ , we have  $M_G = 1 - a - b - c + ac$ , the denominator in (139). Cartier and Foata's generalization of (139) has a remarkably simple proof:

**Theorem F.** *The generating function  $T_G$  for the sum of all traces, with respect to any graph  $G$ , is  $1/M_G$ .*

*Proof.* We want to show that  $M_G T_G = 1$ , in the (partially commutative) algebra of traces. This infinite product is  $\sum_{\alpha, \beta} \mu_G(\alpha) = \sum_{\gamma} \sum_{\alpha, \beta} \mu_G(\alpha)[\gamma = \alpha\beta]$ . Hence we want to show that the sum of  $\mu_G(\alpha)$ , over all ways to factorize  $\gamma = \alpha\beta$  as the product of two traces  $\alpha$  and  $\beta$ , is zero whenever  $\gamma$  is nonempty.

But that's easy. We can assume that the letters are ordered in some arbitrary fashion. Let  $a$  be the smallest letter in the bottom level of  $\gamma$ 's empilement. We can restrict attention to cases where  $\alpha$  consists of independent (commuting) letters (pieces), because  $\mu_G(\alpha) = 0$  otherwise. Now if  $\alpha = a\alpha'$  for some trace  $\alpha'$ , let  $\beta' = a\beta$ ; otherwise we must have  $\beta = a\beta'$  for some trace  $\beta'$ , and we let  $\alpha' = a\alpha$ . In both cases  $\alpha\beta = \alpha'\beta'$ ,  $(\alpha')' = \alpha$ ,  $(\beta')' = \beta$ , and  $\mu_G(\alpha) + \mu_G(\alpha') = 0$ . So we've grouped all possible factorizations of  $\gamma$  into pairs that cancel out in the sum. ■

The Möbius series for any graph can be computed recursively via the formula

$$M_G = M_{G \setminus a} - aM_{G \setminus a^*}, \quad a^* = \{a\} \cup \{b \mid a - b\}, \tag{142}$$

where  $a$  is any letter (vertex) of  $G$ , because we have  $a \notin I$  or  $a \in I$  whenever  $I$  is independent. For example, if  $G$  is the path  $a - b - c - d - e - f$ , then  $G \setminus a^* = G \setminus \{c, d, e, f\}$  is the path  $c - d - e - f$ ; repeated use of (142) yields

$$M_G = 1 - a - b - c - d - e - f + ac + ad + ae + af + bd + be + bf + ce + cf + df - ace - acf - adf - bdf \tag{143}$$

in this case. Since  $M_G$  is a polynomial, we can indicate its dependence on the variables by writing  $M_G(a, b, c, d, e, f)$ . Notice that  $M_G$  is always multilinear (this is, linear in each variable); and  $M_{G \setminus a}(b, c, d, e, f) = M_G(0, b, c, d, e, f)$ .

Cartier  
Foata  
Möbius function of a trace  
Möbius function  
Möbius series  
Möbius polynomial  
multilinear

In applications we often want to replace each letter in the polynomial by a single variable, such as  $z$ , and write  $M_G(z)$ . The polynomial in (143) then becomes  $M_G(z) = 1 - 6z + 10z^2 - 4z^3$ ; and we can conclude from Theorem F that the number of traces of length  $n$  with respect to  $G$  is  $[z^n] 1/(1 - 6z + 10z^2 - 4z^3) = \frac{1}{4}(2 + \sqrt{2})^{n+2} + \frac{1}{4}(2 - \sqrt{2})^{n+2} - 2^{n+1}$ .

Although (142) is a simple recurrence for  $M_G$ , we can't conclude that  $M_G$  is easy to compute when  $G$  is a large and complicated graph. Indeed, the degree of  $M_G$  is the size of a maximum independent set in  $G$ ; and it's NP-hard to determine that number! On the other hand, there are many classes of graphs, such as interval graphs and forests, for which  $M_G$  can be computed in linear time.

If  $\alpha$  is any trace, the letters that can occur first in a string that represents it are called the *sources* of  $\alpha$ ; these are the pieces on the bottom level of  $\alpha$ 's empilement, also called its minimal pieces. Dually, the letters that can occur last are the *sinks* of  $\alpha$ , its maximal pieces. A trace that has only one source is called a *cone*; in this case all pieces are ultimately supported by a single piece at the bottom. A trace that has only one sink is, similarly, called a *pyramid*. Viennot proved a nice generalization of Theorem F in his lecture notes:

$$M_{G \setminus A} / M_G \text{ is the sum of all traces whose sources are contained in } A. \quad (144)$$

(See exercise 338; Theorem F is the special case where  $A$  is the set of all vertices.) In particular, the cones for which  $a$  is the only source are generated by

$$M_{G \setminus a} / M_G - 1 = a M_{G \setminus a^*} / M_G. \quad (145)$$

**\*Traces and the Local Lemma.** Now we're ready to see why the theory of traces is intimately connected with the Local Lemma. If  $G$  is any graph on the vertices  $\{1, \dots, m\}$ , we say that  $\mathcal{R}(G)$  is the set of all nonnegative vectors  $(p_1, \dots, p_m)$  such that  $M_G(p'_1, \dots, p'_m) > 0$  whenever  $0 \leq p'_j \leq p_j$  for  $1 \leq j \leq m$ . This definition of  $\mathcal{R}(G)$  is consistent with the implicit definition already given in Lemma L, because of the following characterization found by J. B. Shearer:

**Theorem S.** Under condition (133) of Lemma L,  $(p_1, \dots, p_m) \in \mathcal{R}(G)$  implies

$$\Pr(\overline{A}_1 \cap \dots \cap \overline{A}_m) \geq M_G(p_1, \dots, p_m) > 0. \quad (146)$$

Conversely, if  $(p_1, \dots, p_m) \notin \mathcal{R}(G)$ , there are events  $B_1, \dots, B_m$  such that

$$\Pr(B_i \mid \overline{B}_{j_1} \cap \dots \cap \overline{B}_{j_k}) = p_i \text{ whenever } k \geq 0 \text{ and } i \not\sim j_1, \dots, i \not\sim j_k, \quad (147)$$

and  $\Pr(\overline{B}_1 \cap \dots \cap \overline{B}_m) = 0$ .

*Proof.* When  $(p_1, \dots, p_m) \in \mathcal{R}(G)$ , exercise 344 proves that there's a unique distribution for events  $B_1, \dots, B_m$  such that they satisfy (147) and also

$$\Pr\left(\bigcap_{j \in J} \overline{A}_j\right) \geq \Pr\left(\bigcap_{j \in J} \overline{B}_j\right) = M_G(p_1[1 \in J], \dots, p_m[m \in J]) \quad (148)$$

for every subset  $J \subseteq \{1, \dots, m\}$ . In this "extreme" worst-possible distribution,  $\Pr(B_i \cap B_j) = 0$  whenever  $i \sim j$  in  $G$ . Exercise 345 proves the converse. ■

maximum independent set  
NP-hard  
interval graphs  
forests  
sources  
sinks  
cone  
pyramid  
Viennot  
 $\mathcal{R}(G)$   
Shearer

Given a probability vector  $(p_1, \dots, p_m)$ , let

$$M_G^*(z) = M_G(p_1 z, \dots, p_m z). \tag{149}$$

Pringsheim  
slack  
consecutive 1s

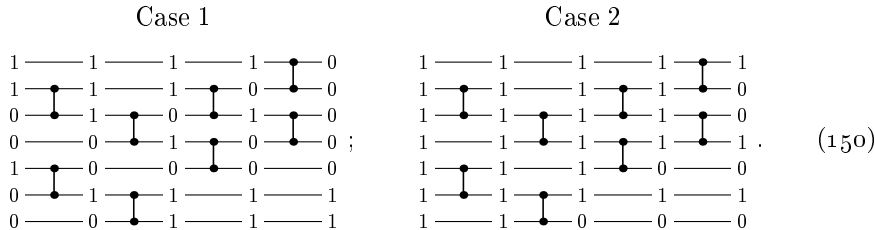
Theorem F tells us that the coefficient of  $z^n$  in the power series  $1/M_G^*(z)$  is the sum of all traces of length  $n$  for  $G$ . Since this coefficient is nonnegative, we know by Pringsheim’s theorem (see exercise 348) that the power series converges for all  $z < 1 + \delta$ , where  $1 + \delta$  is the smallest real root of the polynomial equation  $M_G^*(z) = 0$ ; this number  $\delta$  is called the *slack* of  $(p_1, \dots, p_m)$  with respect to  $G$ .

It’s easy to see that  $(p_1, \dots, p_m) \in \mathcal{R}(G)$  if and only if the slack is positive. For if  $\delta \leq 0$ , the probabilities  $(p'_1, \dots, p'_m)$  with  $p'_j = (1 + \delta)p_j$  make  $M_G = 0$ . But if  $\delta > 0$ , the power series converges when  $z = 1$ . And (since it represents the sum of all traces) it also converges to the positive number  $1/M_G$  if any  $p_j$  is decreased; hence  $(p_1, \dots, p_m)$  lies in  $\mathcal{R}(G)$  by definition. Indeed, this argument shows that, when  $(p_1, \dots, p_m) \in \mathcal{R}(G)$ , we can actually *increase* the probabilities to  $((1 + \epsilon)p_1, \dots, (1 + \epsilon)p_m)$ , and they will still lie in  $\mathcal{R}(G)$  whenever  $\epsilon < \delta$ .

Let’s return now to Algorithm M. Suppose the successive bad events  $A_j$  that step M3 tries to quench are  $X_1, X_2, \dots, X_N$ , where  $N$  is the total number of times step M3 is performed (possibly  $N = \infty$ ). To prove that Algorithm M is efficient, we shall show that this random variable  $N$  has a small expected value, in the probability space of the independent uniform deviates  $U$  that appear in steps M1 and M3. The main idea is that  $X_1 X_2 \dots X_N$  is essentially a trace for the underlying graph; hence we can consider it as an empilement of pieces.

Some simple and concrete examples will help to develop our intuition; we shall consider two case studies. In both cases there are  $m = 6$  events  $A, B, C, D, E, F$ , and there are  $n = 7$  variables  $x_1 \dots x_7$ . Each variable is a random bit; thus  $\xi_1 = \dots = \xi_7 = 1/2$  in the algorithm. Event  $A$  depends on  $x_1 x_2$ , while  $B$  depends on  $x_2 x_3, \dots$ , and  $F$  depends on  $x_6 x_7$ . Furthermore, each event occurs with probability  $1/4$ . In Case 1, each event is true when its substring is ‘10’; thus all events are false if and only if  $x_1 \dots x_7$  is sorted—that is,  $x_1 \leq x_2 \leq \dots \leq x_7$ . In Case 2, each event is true when its substring is ‘11’; thus all events are false if and only if  $x_1 \dots x_7$  has no two consecutive 1s.

What happens when we apply Algorithm M to those two cases? One possible scenario is that step M3 is applied  $N = 8$  times, with  $X_1 X_2 \dots X_8 = BCEBAFDC$ . The actual changes to the bits  $x_1 \dots x_7$  might then be



(Read  $x_1 \dots x_7$  from top to bottom in these diagrams, and scan from left to right. Each module ‘ $\bullet$ ’ means “replace the two bad bits at the left by two random bits

at the right.” In examples such as this, any valid solution  $x_1 \dots x_7$  can be placed at the far right; all values to the left of the modules are then forced.)

Notice that these diagrams are like the empilement (136), except that they’ve been rotated  $90^\circ$ . We know from (136) that the same diagram applies to the scenario *EFBCDBCA* as well as to *BCEBAFDC*, because they’re the same, as traces. Well . . . , not quite! In truth, *EFBCDBCA* doesn’t give exactly the same result as *BCEBAFDC* in Algorithm M, if we execute that algorithm as presently written. But the results *would* be identical if we used *separate* streams of independent random numbers  $U_k$  for each variable  $x_k$ . Thus we can legitimately *equate* equivalent traces, in the probability space of our random events.

The algorithm runs much faster in practice when it’s applied to Case 1 than when it’s applied to Case 2. How can that be? Both of the diagrams in (150) occur with the same probability, namely  $(1/2)^7(1/4)^8$ , as far as the random numbers are concerned. And every diagram for Case 1 has a corresponding diagram for Case 2; so we can’t distinguish the cases by the number of different diagrams. The real difference comes from the fact that, in Case 1, we never have two events to choose from in step M2, unless they are disjoint and can be handled in either order. In Case 2, by contrast, we are deluged at almost every step with events that need to be snuffed out. Therefore the scenario at the right of (150) is actually quite unlikely; why should the algorithm pick *B* as the first event to correct, and then *C*, rather than *A*? Whatever method is used in step M2, we’ll find that the diagrams for Case 2 will occur less frequently than dictated by the strict probabilities, because of the decreasing likelihood that any particular event will be worked on next, in the presence of competing choices. (See also exercise 353.)

Worst-case upper bounds on the running time of Algorithm M therefore come from situations like Case 1. In general, the empilement *BCEBAFDC* in (150) will occur in a run of Algorithm M with probability at most *bcebafdc*, if we write ‘*a*’ for the probabilistic upper bound for event *A* that is denoted by ‘ $p_i$ ’ in (133) when *A* is  $A_i$ , and if ‘*b*’, . . . , ‘*f*’ are similar for *B*, . . . , *F*. The reason is that *bcebafdc* is clearly the probability that those events are produced by the independent random variables  $x_k$  set by the algorithm, if the layers of the corresponding empilement are defined by dependencies between the variable sets  $\Xi_j$ . And even if events in the same layer are dependent (by shared variables) yet not lopsided (in the sense of exercise 351), such events are positively correlated; so the FKG inequality of exercise MPR-61, which holds for the Bernoulli-distributed variables of Algorithm M, shows that *bcebafdc* is an upper bound. Furthermore the probability that step M2 actually chooses *B*, *C*, *E*, *B*, *A*, *F*, *D*, and *C* to work on is at most 1.

Therefore, when  $(p_1, \dots, p_m) \in \mathcal{R}(G)$ , Algorithm M’s running time is maximized when it is applied to events  $B_1, \dots, B_m$  that have the extreme distribution (148) of exercise 344. And we can actually write down the *generating function* for the running time with respect to those extreme events: We have

$$\sum_{N \geq 0} \Pr(\text{Algorithm M on } B_1, \dots, B_m \text{ does } N \text{ resamplings}) z^N = \frac{M_G^*(1)}{M_G^*(z)}, \quad (151)$$

handwaving  
FKG inequality  
Bernoulli-distributed  
extreme distribution  
generating function

where  $M_G^*(z)$  is defined in (149), because the coefficient of  $z^N$  in  $1/M_G^*(z)$  is the sum of the probabilities of all the traces of length  $N$ . Theorem F describes the meaning of  $1/M_G^*(1)$  as a “formal” power series in the variables  $p_i$ ; we proved it without considering whether or not the infinite sum converges when those variables receive numerical values. But when  $(p_1, \dots, p_m) \in \mathcal{R}(G)$ , this series is indeed convergent (it even has a positive “slack”).

This reasoning leads to the following theorem of K. Kolipaka and M. Szegedy [*STOC* **43** (2011), 235–243]:

**Theorem K.** *If  $(p_1, \dots, p_m) \in \mathcal{R}(G)$ , Algorithm M resamples  $\Xi_j$  at most*

$$E_j = p_j M_{G \setminus A_j^*}(p_1, \dots, p_m) / M_G(p_1, \dots, p_m) \quad (152)$$

*times, on the average. In particular, the expected number of iterations of step M3 is at most  $E_1 + \dots + E_m \leq m/\delta$ , where  $\delta$  is the slack of  $(p_1, \dots, p_m)$ .*

*Proof.* The extreme distribution  $B_1, \dots, B_m$  maximizes the number of times  $\Xi_j$  is resampled, and the generating function for that number in the extreme case is

$$\frac{M_G(p_1, \dots, p_{j-1}, p_j, p_{j+1}, \dots, p_m)}{M_G(p_1, \dots, p_{j-1}, p_j z, p_{j+1}, \dots, p_m)}. \quad (153)$$

Differentiating with respect to  $z$ , then setting  $z \leftarrow 1$ , gives (152), because the derivative of the denominator is  $-p_j M_{G \setminus A_j^*}(p_1, \dots, p_m)$  by (141).

The stated upper bound on  $E_1 + \dots + E_m$  is proved in exercise 355. ■

**\*Message passing.** Physicists who study statistical mechanics have developed a significantly different way to apply randomization to satisfiability problems, based on their experience with the behavior of large systems of interacting particles. From their perspective, a set of Boolean variables whose values are 0 or 1 is best viewed as an ensemble of particles that have positive or negative “spin”; these particles affect each other and change their spins according to local attractions and repulsions, analogous to laws of magnetism. A satisfiability problem can be formulated as a joint probability distribution on spins for which the states of minimum “energy” are achieved precisely when the spins satisfy as many clauses as possible.

In essence, their approach amounts to considering a bipartite structure in which each variable is connected to one or more clauses, and each clause is connected to one or more variables. We can regard both variables and clauses as active agents, who continually tweet to their neighbors in this social network. A variable might inform its clauses that “I think I should probably be true”; but several of those clauses might reply, “I really wish you were false.” By carefully balancing these messages against each other, such local interactions can propagate and build up more and more knowledge of distant connections, often converging to a state where the whole network is reasonably happy.

A particular message-passing strategy called *survey propagation* [A. Braunstein, M. Mézard, and R. Zecchina, *Random Structures & Algorithms* **27** (2005),

Kolipaka  
Szegedy  
message passing–  
statistical mechanics  
bipartite structure  
survey propagation–  
Braunstein  
Mézar  
Zecchina

201–226] has proved to be astonishingly good at solving random satisfiability problems in the “hard” region just before the threshold of unsatisfiability.

Let  $C$  be a clause and let  $l$  be one of its literals. A “survey message”  $\eta_{C \rightarrow l}$  is a fraction between 0 and 1 that represents how urgently  $C$  wants  $l$  to be true. If  $\eta_{C \rightarrow l} = 1$ , the truth of  $l$  is desperately needed, lest  $C$  be false; but if  $\eta_{C \rightarrow l} = 0$ , clause  $C$  isn’t the least bit worried about the value of variable  $|l|$ . Initially we set each  $\eta_{C \rightarrow l}$  to a completely random fraction.

We shall consider an extension of the original survey propagation method [see J. Chavas, C. Furtlehner, M. Mézard, and R. Zecchina, *J. Statistical Mechanics* (November 2005), P11016:1–25; A. Braunstein and R. Zecchina, *Physical Review Letters* **96** (27 January 2006), 030201:1–4], which introduces additional “reinforcement messages”  $\eta_l$  for each literal  $l$ . These new messages, which are initially all zero, represent an external force that acts on  $l$ . They help to focus the network activity by reinforcing decisions that have turned out to be fruitful.

Suppose  $v$  is a variable that appears in just three clauses: positively in  $A$  and  $B$ , negatively in  $C$ . This variable will respond to its incoming messages  $\eta_{A \rightarrow v}$ ,  $\eta_{B \rightarrow v}$ ,  $\eta_{C \rightarrow \bar{v}}$ ,  $\eta_v$ , and  $\eta_{\bar{v}}$  by computing two “flexibility coefficients,”  $\pi_v$  and  $\pi_{\bar{v}}$ , using the following formulas:

$$\pi_v = (1 - \eta_v)(1 - \eta_{A \rightarrow v})(1 - \eta_{B \rightarrow v}), \quad \pi_{\bar{v}} = (1 - \eta_{\bar{v}})(1 - \eta_{C \rightarrow \bar{v}}).$$

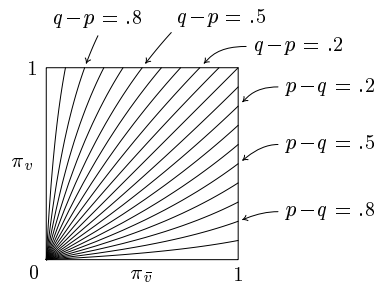
If, for instance,  $\eta_v = \eta_{\bar{v}} = 0$  while  $\eta_{A \rightarrow v} = \eta_{B \rightarrow v} = \eta_{C \rightarrow \bar{v}} = 2/3$ , then  $\pi_v = 1/9$ ,  $\pi_{\bar{v}} = 1/3$ . The  $\pi$ ’s are essentially dual to the  $\eta$ ’s, because high urgency corresponds to low flexibility and vice versa. The general formula for each literal  $l$  is

$$\pi_l = (1 - \eta_l) \prod_{l \in C} (1 - \eta_{C \rightarrow l}). \tag{154}$$

Survey propagation uses these coefficients to estimate variable  $v$ ’s tendency to be either 1 (true), 0 (false), or \* (wild), by computing three numbers

$$p = \frac{(1 - \pi_v)\pi_{\bar{v}}}{\pi_v + \pi_{\bar{v}} - \pi_v\pi_{\bar{v}}}, \quad q = \frac{(1 - \pi_{\bar{v}})\pi_v}{\pi_v + \pi_{\bar{v}} - \pi_v\pi_{\bar{v}}}, \quad r = \frac{\pi_v\pi_{\bar{v}}}{\pi_v + \pi_{\bar{v}} - \pi_v\pi_{\bar{v}}}; \tag{155}$$

then  $p + q + r = 1$ , and  $(p, q, r)$  is called the “field” of  $v$ , representing respectively (truth, falsity, wildness). The field turns out to be  $(8/11, 2/11, 1/11)$  in our example above, indicating that  $v$  should probably be assigned the value 1. But if  $\eta_{A \rightarrow v}$  and  $\eta_{B \rightarrow v}$  had been only  $1/3$  instead of  $2/3$ , the field would have been  $(5/17, 8/17, 4/17)$ , and we would probably want  $v = 0$  in order to satisfy clause  $C$ . Figure 51 shows lines of constant  $p - q$  as a function of  $\pi_v$  and  $\pi_{\bar{v}}$ ; the most decisive cases ( $|p - q| \approx 1$ ) occur at the lower right and upper left.



**Fig. 51.** Lines of constant bias in a variable’s “field.”

random satisfiability problems  
 threshold of unsatisfiability  
 Chavas  
 Furtlehner  
 Mézard  
 Zecchina  
 Braunstein  
 focus  
 flexibility coefficients  
 field



If  $\pi_v = \pi_{\bar{v}} = 0$ , there's no flexibility at all: Variable  $v$  is being asked to be both true and false. The field is undefined in such cases, and the survey propagation method hopes that this doesn't happen.

bias message  
monus

After each literal  $l$  has computed its flexibility, the clauses that involve  $l$  or  $\bar{l}$  can use  $\pi_l$  and  $\pi_{\bar{l}}$  to refine their survey messages. Suppose, for example, that  $C$  is the clause  $u \vee \bar{v} \vee w$ . It will replace the former messages  $\eta_{C \rightarrow u}$ ,  $\eta_{C \rightarrow \bar{v}}$ ,  $\eta_{C \rightarrow w}$  by

$$\eta'_{C \rightarrow u} = \gamma_{\bar{v} \rightarrow C} \gamma_{w \rightarrow C}, \quad \eta'_{C \rightarrow \bar{v}} = \gamma_{u \rightarrow C} \gamma_{w \rightarrow C}, \quad \eta'_{C \rightarrow w} = \gamma_{u \rightarrow C} \gamma_{\bar{v} \rightarrow C},$$

where each  $\gamma_{l \rightarrow C}$  is a “bias message” received from literal  $l$ ,

$$\gamma_{l \rightarrow C} = \frac{(1 - \pi_{\bar{l}}) \pi_l / (1 - \eta_{C \rightarrow l})}{\pi_{\bar{l}} + (1 - \pi_{\bar{l}}) \pi_l / (1 - \eta_{C \rightarrow l})}, \tag{156}$$

reflecting  $l$ 's propensity to be false in clauses other than  $C$ . In general we have

$$\eta'_{C \rightarrow l} = \left( \prod_{l' \in C} \gamma_{l' \rightarrow C} \right) / \gamma_{l \rightarrow C}. \tag{157}$$

(Appropriate conventions must be used to avoid division by zero in formulas (156) and (157); see exercise 359.)

New reinforcement messages  $\eta'_l$  can also be computed periodically, using the formula

$$\eta'_l = \frac{\kappa(\pi_{\bar{l}} \div \pi_l)}{\pi_l + \pi_{\bar{l}} - \pi_l \pi_{\bar{l}}} \tag{158}$$

for each literal  $l$ ; here  $x \div y$  denotes  $\max(x - y, 0)$ , and  $\kappa$  is a reinforcement parameter specified by the algorithm. Notice that  $\eta'_l > 0$  only if  $\eta_{\bar{l}} = 0$ .

For example, here are messages that might be passed when we want to satisfy the seven clauses of (7):

| $l_1$     | $l_2$     | $l_3$     | $\eta_{C \rightarrow l_1}$ | $\eta_{C \rightarrow l_2}$ | $\eta_{C \rightarrow l_3}$ | $\gamma_{l_1 \rightarrow C}$ | $\gamma_{l_2 \rightarrow C}$ | $\gamma_{l_3 \rightarrow C}$ | $l$       | $\pi_l$ | $\eta_l$ |
|-----------|-----------|-----------|----------------------------|----------------------------|----------------------------|------------------------------|------------------------------|------------------------------|-----------|---------|----------|
| 1         | 2         | $\bar{3}$ | 0                          | 0                          | 0                          | 3/5                          | 0                            | 0                            | 1         | 1       | 0        |
| $\bar{1}$ | $\bar{2}$ | 3         | 1/5                        | 0                          | 0                          | 0                            | 3/5                          | 1/3                          | $\bar{1}$ | 2/5     | 1/2      |
| 2         | 3         | $\bar{4}$ | 1/5                        | 0                          | 0                          | 0                            | 1/3                          | 3/5                          | 2         | 2/5     | 1/2      |
| $\bar{2}$ | $\bar{3}$ | 4         | 0                          | 0                          | 0                          | 3/5                          | 0                            | 0                            | $\bar{2}$ | 1       | 0        |
| 1         | 3         | 4         | 0                          | 0                          | 1/5                        | 3/5                          | 1/3                          | 0                            | 3         | 1       | 0        |
| $\bar{1}$ | $\bar{3}$ | $\bar{4}$ | 0                          | 0                          | 0                          | 0                            | 0                            | 3/5                          | $\bar{3}$ | 2/3     | 1/3      |
| $\bar{1}$ | 2         | 4         | 0                          | 0                          | 0                          | 0                            | 0                            | 0                            | 4         | 2/5     | 1/2      |
|           |           |           |                            |                            |                            |                              |                              |                              | $\bar{4}$ | 1       | 0        |

(159)

(Recall that the only solutions to these clauses are  $\bar{1} 2 3 4$  and  $\bar{1} 2 \bar{3} 4$ .) In this case the reader may verify that the messages of (159) constitute a “fixed point”: The  $\eta$  messages determine the  $\pi$ 's; conversely, we also have  $\eta'_{C \rightarrow l} = \eta_{C \rightarrow l}$  for all clauses  $C$  and all literals  $l$ , if the reinforcement messages  $\eta_l$  remain constant.

Exercise 361 proves that every solution to a satisfiable set of clauses yields a fixed point of the simultaneous equations (154), (156), (157), with the property that  $\eta_l = [l \text{ is true in the solution}]$ .

Experiments with this message-passing strategy have shown, however, that the best results are obtained by using it only for preliminary screening, with the goal of discovering variables whose settings are most critical; we needn't continue to transmit messages until every clause is fully satisfied. Once we've assigned suitable values to the most delicate variables, we're usually left with a residual problem that can readily be solved by other algorithms such as WalkSAT.

The survey, reinforcement, and bias messages can be exchanged using a wide variety of different protocols. The following procedure incorporates two ideas from an implementation prepared by C. Baldassi in 2012: (1) The reinforcement strength  $\kappa$  begins at zero, but approaches 1 exponentially. (2) Variables are rated 1, 0, or \* after each reinforcement, according as  $\max(p, q, r)$  in their current field is  $p$ ,  $q$ , or  $r$ . If every clause then has at least one literal that is true or \*, message passing will cease even though some surveys might still be fluctuating.

**Algorithm S** (*Survey propagation*). Given  $m$  nonempty clauses on  $n$  variables, this algorithm tries to assign values to most of the variables in such a way that the still-unsatisfied clauses will be relatively easy to satisfy. It maintains arrays  $\pi_l$  and  $\eta_l$  of floating-point numbers for each literal  $l$ , as well as  $\eta_{C \rightarrow l}$  for each clause  $C$  and each  $l \in C$ . It has a variety of parameters:  $\rho$  (the damping factor for reinforcement),  $N_0$  and  $N$  (the minimum and maximum iteration limits),  $\epsilon$  (the tolerance for convergence), and  $\psi$  (the confidence level).

- S1.** [Initialize.] Set  $\eta_l \leftarrow \pi_l \leftarrow 0$  for all literals  $l$ , and  $\eta_{C \rightarrow l} \leftarrow U$  for all clauses  $C$  and  $l \in C$ , where  $U$  is uniformly random in  $[0..1)$ . Also set  $i \leftarrow 0$ ,  $\phi \leftarrow 1$ .
- S2.** [Done?] Terminate unsuccessfully if  $i \geq N$ . If  $i$  is even or  $i < N_0$ , go to S5.
- S3.** [Reinforce.] Set  $\phi \leftarrow \rho\phi$  and  $\kappa \leftarrow 1 - \phi$ . Replace  $\eta_l$  by  $\eta_l'$  for all literals  $l$ , using (158); but terminate unsuccessfully if  $\pi_l = \pi_{\bar{l}} = 0$ .
- S4.** [Test pseudo-satisfiability.] Go to S5 if there is at least one clause whose literals  $l$  all appear to be false, in the sense that  $\pi_{\bar{l}} < \pi_l$  and  $\pi_{\bar{l}} < \frac{1}{2}$  (see exercise 358). Otherwise go happily to S8.
- S5.** [Compute the  $\pi$ 's.] Compute each  $\pi_l$ , using (154); see also exercise 359.
- S6.** [Update the surveys.] Set  $\delta \leftarrow 0$ . For all clauses  $C$  and literals  $l \in C$ , compute  $\eta'_{C \rightarrow l}$  using (157), and set  $\delta \leftarrow \max(\delta, |\eta'_{C \rightarrow l} - \eta_{C \rightarrow l}|)$ ,  $\eta_{C \rightarrow l} \leftarrow \eta'_{C \rightarrow l}$ .
- S7.** [Loop on  $i$ .] If  $\delta \geq \epsilon$ , set  $i \leftarrow i + 1$  and return to S2.
- S8.** [Reduce the problem.] Assign a value to each variable whose field satisfies  $|p - q| \geq \psi$ . (Exercise 362 has further details.) ■

Computational experience — otherwise known as trial and error — suggests suitable parameter values. The defaults  $\rho = .995$ ,  $N_0 = 5$ ,  $N = 1000$ ,  $\epsilon = .01$ , and  $\psi = .50$  seem to provide a decent starting point for problems of modest size. They worked well, for instance, when the author first tried a random 3SAT problem with 42,000 clauses and 10,000 variables: These clauses were pseudo-satisfiable when  $i = 143$  (although  $\delta \approx .43$  was still rather large); then step S8 fixed the values of 8,282 variables with highly biased fields, and unit propagation gave values to 57 variables more. This process needed only about 218 megamems of calculation. The reduced problem had 1526 2-clauses and 196 3-clauses on

WalkSAT+  
Baldassi  
damping factor  
parameter  
defaults  
author  
3SAT  
unit propagation

1464 variables (because many other variables were no longer needed); 626 steps of WalkSAT polished it off after an additional 42 kilomems. By contrast, when WalkSAT was presented with the original problem (using  $p = .57$ ), it needed more than 31 million steps to find a solution after 3.4 gigamems of computation.

author

Similarly, the author's first experience applying survey propagation to a random 3SAT problem on  $n = 10^6$  variables with  $m = 4.2n$  clauses was a smashing success: More than 800,000 variables were eliminated after only 32.8 gigamems of computation, and WalkSAT solved the residual clauses after 8.5 megamems more. By contrast, pure WalkSAT needed 237 gigamems to perform 2.1 billion steps.

A million-variable problem with 4,250,000 clauses proved to be more challenging. These additional 50,000 clauses put the problem well beyond WalkSAT's capability; and Algorithm S failed too, with its default parameters. However, the settings  $\rho = .9999$  and  $N_0 = 9$  slowed the reinforcement down satisfactorily, and produced some instructive behavior. Consider the matrix

$$\begin{pmatrix} 3988 & 3651 & 3071 & 2339 & 1741 & 1338 & 946 & 702 & 508 & 329 \\ 5649 & 5408 & 4304 & 3349 & 2541 & 2052 & 1448 & 1050 & 666 & 510 \\ 8497 & 7965 & 6386 & 4918 & 3897 & 3012 & 2248 & 1508 & 1075 & 718 \\ 11807 & 11005 & 8812 & 7019 & 5328 & 4135 & 3117 & 2171 & 1475 & 1063 \\ 15814 & 14789 & 11726 & 9134 & 7188 & 5425 & 4121 & 3024 & 2039 & 1372 \\ 20437 & 19342 & 15604 & 12183 & 9397 & 7263 & 5165 & 3791 & 2603 & 1781 \\ 26455 & 24545 & 19917 & 15807 & 12043 & 9161 & 6820 & 5019 & 3381 & 2263 \\ 33203 & 31153 & 25052 & 19644 & 15587 & 11802 & 8865 & 6309 & 4417 & 2919 \\ 39962 & 38097 & 31060 & 24826 & 18943 & 14707 & 10993 & 7924 & 5225 & 3637 \\ 40731 & 40426 & 32716 & 26561 & 20557 & 15739 & 11634 & 8327 & 5591 & 4035 \end{pmatrix},$$

which shows the distribution of  $\pi_{\bar{v}}$  versus  $\pi_v$  (see Fig. 51); for example, '3988' at the upper left means that 3988 of the million variables had  $\pi_{\bar{v}}$  between 0.0 and 0.1 and  $\pi_v$  between 0.9 and 1.0. This distribution, which appeared after  $\delta$  had been reduced to  $\approx 0.0098$  by 110 iterations, is terrible—very few variables are biased in a meaningful way. Therefore another run was made with  $\epsilon$  reduced to .001; but that failed to converge after 1000 iterations. Finally, with  $\epsilon = .001$  and  $N = 2000$ , pseudo-satisfaction occurred at  $i = 1373$ , with the nice distribution

$$\begin{pmatrix} 406678 & 1946 & 1045 & 979 & 842 & 714 & 687 & 803 & 1298 & 167649 \\ 338 & 2 & 2 & 3 & 0 & 3 & 1 & 4 & 2 & 1289 \\ 156 & 1 & 0 & 0 & 0 & 1 & 0 & 2 & 1 & 875 \\ 118 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 743 \\ 99 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 663 \\ 62 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 810 \\ 41 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1015 \\ 55 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1139 \\ 63 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 2 & 1949 \\ 116 & 61 & 72 & 41 & 61 & 103 & 120 & 162 & 327 & 406839 \end{pmatrix}$$

(although  $\delta$  was now  $\approx 1!$ ). The biases were now pronounced, yet not entirely reliable; the  $\psi$  parameter had to be raised, in order to avoid a contradiction when propagating unit literals in the reduced problem. Finally, with  $\psi = .99$ , more than 800,000 variables could be set successfully. A solution was obtained after 210 gigamems (including 21 megamems for WalkSAT to finish the job, but not including the time spent learning how to set the parameters for so many clauses).

Success with Algorithm S isn't guaranteed. But hey, when it works, it's sometimes the only known way to solve a particularly tough problem.

Survey propagation may be viewed as an extension of the “belief propagation” messages used in the study of Bayesian networks [see J. Pearl, *Probabilistic Reasoning in Intelligent Systems* (1988), Chapter 4]; it essentially goes beyond Boolean logic on  $\{0, 1\}$  to a three-valued logic on  $\{0, 1, *\}$ . Analogous message-passing heuristics had actually been considered much earlier by H. A. Bethe and R. E. Peierls [*Proc. Royal Society of London* **A150** (1935), 552–575], and independently by R. G. Gallager [*IRE Transactions* **IT-8** (1962), 21–28]. For further information see M. Mézard and A. Montanari, *Information, Physics, and Computation* (2009), Chapters 14–22.

**\*Preprocessing of clauses.** A SAT-solving algorithm will often run considerably faster if its input has been transformed into an equivalent but simpler set of clauses. Such transformations and simplifications typically require data structures that would be inappropriate for the main work of a solver, so they are best considered separately.

Of course we can combine a preprocessor and a solver into a single program; and “preprocessing” techniques can be applied again after new clauses have been learned, if we reach a stage where we want to clean up and start afresh. In the latter case the simplifications are called *inprocessing*. But the basic ideas are most easily explained by assuming that we just want to preprocess a given family of clauses  $F$ . Our goal is to produce nicer clauses  $F'$ , which are satisfiable if and only if  $F$  is satisfiable.

We shall view preprocessing as a sequence of elementary transformations

$$F = F_0 \rightarrow F_1 \rightarrow \dots \rightarrow F_r = F', \quad (160)$$

where each step  $F_j \rightarrow F_{j+1}$  “flows downhill” in the sense that it either (i) eliminates a variable without increasing the number of clauses, or (ii) retains all the variables but decreases the number of literals in clauses. Many different downhill transformations are known; and we can try to apply each of the gimmicks in our repertoire, in some order, until none of them lead to any further progress.

Sometimes we'll actually *solve* the given problem, by reaching an  $F'$  that is either trivially satisfiable ( $\emptyset$ ) or trivially unsatisfiable (contains  $\epsilon$ ). But we probably won't be so lucky unless  $F$  was pretty easy to start with, because we're going to consider only downhill transformations that are quite simple.

Before discussing particular transformations, however, let's think about the endgame: Suppose  $F$  has  $n$  variables but  $F'$  has  $n' < n$ . After we've fed the clauses  $F'$  into a SAT solver and received back a solution,  $x'_1 \dots x'_{n'}$ , how can we convert it to a full solution  $x_1 \dots x_n$  of the original problem  $F$ ? Here's how: For every transformation  $F_j \rightarrow F_{j+1}$  that eliminates a variable  $x_k$ , we shall specify an *erp rule* (so-called because it reverses the effect of *preprocessing*). An erp rule for elimination is simply an assignment ' $l \leftarrow E$ ', where  $l$  is  $x_k$  or  $\bar{x}_k$ , and  $E$  is a Boolean expression that involves only variables that have not been eliminated. We undo the effect of elimination by assigning to  $x_k$  the value that makes  $l$  true if and only if  $E$  is true.

belief propagation  
Bayesian networks  
Pearl  
Bethe  
Peierls  
Gallager  
Mézard  
Montanari  
Preprocessing-  
simplifications  
data structures  
restarting  
inprocessing  
downhill transformations  
erp rule

For example, suppose two transformations remove  $x$  and  $y$  with the erp rules

$$\bar{x} \leftarrow \bar{y} \vee z, \quad y \leftarrow 1.$$

To reverse these eliminations, right to left, we would set  $y$  true, then  $x \leftarrow \bar{z}$ .

As the preprocessor discovers how to eliminate variables, it can immediately write the corresponding erp rules to a file, so that those rules don't consume memory space. Afterwards, given a reduced solution  $x'_1 \dots x'_{n'}$ , a postprocessor can read that file *in reverse order* and provide the unreduced solution  $x_1 \dots x_n$ .

**Transformation 1. Unit conditioning.** If a unit clause ' $l$ ' is present, we can replace  $F$  by  $F|l$  and use the erp rule  $l \leftarrow 1$ . This elementary simplification will be carried out naturally by most solvers; but it is perhaps even more important in a preprocessor, since it often enables further transformations that the solver would not readily see. Conversely, other transformations in the preprocessor might enable unit conditionings that will continue to ripple down.

One consequence of unit conditioning is that all clauses of  $F'$  will have length two or more, unless  $F'$  is trivially unsatisfiable.

**Transformation 2. Subsumption.** If every literal in clause  $C$  appears also in another clause  $C'$ , we can remove  $C$ . In particular, duplicate clauses will be discarded. No erp rule is needed, because no variable goes away.

**Transformation 3. Self-subsumption.** If every literal in  $C$  except  $\bar{x}$  appears also in another clause  $C'$ , where  $C'$  contains  $x$ , we can delete  $x$  from  $C$  because  $C' \setminus x = C \diamond C'$ . In other words, the fact that  $C$  *almost* subsumes  $C'$  allows us at least to *strengthen*  $C'$ , without actually removing it. Again there's no erp rule. [Self-subsumption was called "the replacement principle" by J. A. Robinson in *JACM* **12** (1965), 39.]

Exercise 374 discusses data structures and algorithms by which subsumptions and self-subsumptions can be discovered with reasonable efficiency.

**Transformation 4. Downhill resolution.** Suppose  $x$  appears only in clauses  $C_1, \dots, C_p$  and  $\bar{x}$  appears only in  $C'_1, \dots, C'_q$ . We've observed (see (112)) that variable  $x$  can be eliminated if we replace those  $p + q$  clauses by the  $pq$  clauses  $\{C_i \diamond C'_j \mid 1 \leq i \leq p, 1 \leq j \leq q\}$ . The corresponding erp rule (see exercise 367) is

$$\text{either } \bar{x} \leftarrow \bigwedge_{i=1}^p (C_i \setminus x) \quad \text{or } x \leftarrow \bigwedge_{j=1}^q (C'_j \setminus \bar{x}). \quad (161)$$

Every variable can be eliminated in this way, but we might be flooded with too many clauses. We can prevent this by limiting ourselves to "downhill" cases, in which the new clauses don't outnumber the old ones. The condition  $pq \leq p + q$  is equivalent to  $(p - 1)(q - 1) \leq 1$ , as noted above following (112); the variable is always removed in such cases. But the number of new clauses might be small even when  $pq$  is large, because of tautologies or subsumption. Furthermore, N. Eén and A. Biere wrote a fundamental paper on preprocessing [*LNCS* **3569** (2005), 61–75] that introduced important special cases in which many of the  $pq$  potential clauses can be omitted; see exercise 369. Therefore a preprocessor typically tries

postprocessor  
 Unit conditioning  
 BCP (Boolean constraint propagation), see  
 $F|l$   
 Subsumption  
 Self-subsumption  
 strengthening a clause  
 replacement principle  
 Robinson  
 data structures  
 Downhill resolution  
 variable elimination  
 elimination of variables  
 Eén  
 Biere

to eliminate via resolution whenever  $\min(p, q) \leq 10$ , say, and abandons the attempt only when more than  $p + q$  resolvents have been generated.

Many other transformations are possible, although the four listed above have proved to be the most effective in practice. We could, for instance, look for *failed literals*: If unit propagation leads to a contradiction when we assume that some literal  $l$  is true (namely when  $F \wedge (l) \vdash_1 \epsilon$ ), then we're allowed to assume that  $l$  is false (because the unit clause  $(\bar{l})$  is certifiable). This observation and several others related to it were exploited in the lookahead mechanisms of Algorithm Y above. But Algorithm C generally has no trouble finding failed literals all by itself, as a natural byproduct of its mechanism for resolving conflicts. Exercises 378–384 discuss other techniques that have been proposed for preprocessing.

Sometimes preprocessing turns out to be dramatically successful. For example, the anti-maximal-element clauses of exercise 228 can be proved unsatisfiable via transformations 1–4 after only about 400 megamems of work when  $m = 50$ . Yet Algorithm C spends 3 gigamems on that untransformed problem when  $m$  is only 14; and it needs 11  $G\mu$  when  $m = 15$ , . . . , failing utterly before  $m = 20$ .

A more typical example arises in connection with Fig. 35 above: The problem of showing that there's no 4-step path to **LIFE** involves 8725 variables, 33769 clauses, and 84041 literals, and Algorithm C requires about 6 gigamems to demonstrate that those clauses are unsatisfiable. Preprocessing needs less than 10 megamems to reduce that problem to just 3263 variables, 19778 clauses, and 56552 literals; then Algorithm C can handle those with 5  $G\mu$  of further work.

On the other hand, preprocessing might take too long, or it might produce clauses that are more difficult to deal with than the originals. It's totally useless on the *waarden* or *langford* problems. (Further examples are discussed below.)

**Encoding constraints into clauses.** Some problems, like *waarden* ( $j, k; n$ ), are inherently Boolean, and they're essentially given to us as native-born ANDs of ORs. But in most cases we can represent a combinatorial problem via clauses in many different ways, not immediately obvious, and the particular encoding that we choose can have an enormous effect on the speed with which a SAT solver is able to crank out an answer. Thus the art of problem encoding turns out to be just as important as the art of devising algorithms for satisfiability.

Our study of SAT instances has already introduced us to dozens of interesting encodings; and new applications often lead to further ideas, because Boolean algebra is so versatile. Each problem may seem at first to need its own special tricks. But we'll see that several general principles are available for guidance.

In the first place, different solvers tend to like different encodings: An encoding that's good for one algorithm might be bad for another.

Consider, for example, the *at-most-one* constraint,  $y_1 + \cdots + y_p \leq 1$ , which arises in a great many applications. The obvious way to enforce this condition is to assert  $\binom{p}{2}$  binary clauses  $(\bar{y}_i \vee \bar{y}_j)$ , for  $1 \leq i < j \leq p$ , so that  $y_i = y_j = 1$  is forbidden; but those clauses become unwieldy when  $p$  is large. The alternative encoding in exercise 12, due to Marijn Heule, does the same job with only  $3p - 6$  binary constraints when  $p \geq 3$ , by introducing a few auxiliary variables

failed literals  
unit propagation  
lookahead  
anti-maximal-element clauses  
*waarden*  
*langford*  
Encoding–  
at-most-one  
Heule  
auxiliary variables

$a_1, \dots, a_{\lfloor (p-3)/2 \rfloor}$ . When we formulated Langford's problem in terms of clauses, via (12), (13), and (14) above, we therefore considered two variants called *langford*( $n$ ) and *langford'*( $n$ ), where the former uses the obvious encoding of at-most-one constraints and the latter uses Heule's method. Furthermore, exercise 7.1.1–55(b) encoded at-most-one constraints in yet another way, having the same number of binary clauses but about twice as many auxiliary variables; let's give the name *langford''*( $n$ ) to the clauses that we get from that scheme.

We weren't ready to discuss which of the encodings works better in practice, when we introduced *langford*( $n$ ) and *langford'*( $n$ ) above, because we hadn't yet examined any SAT-solving algorithms. But now we're ready to reveal the answer; and the answer is: "It depends." Sometimes *langford'*( $n$ ) wins over *langford*( $n$ ); sometimes it loses. It always seems to beat *langford''*( $n$ ). Here, for example, are typical statistics, with runtimes rounded to megamems (M $\mu$ ) or kilomems (K $\mu$ ):

|                        | variables | clauses | Algorithm D    | Algorithm L   | Algorithm C    |         |
|------------------------|-----------|---------|----------------|---------------|----------------|---------|
| <i>langford</i> (9)    | 104       | 1722    | 23 M $\mu$     | 16 M $\mu$    | 15 M $\mu$     | (UNSAT) |
| <i>langford'</i> (9)   | 213       | 801     | 82 M $\mu$     | 16 M $\mu$    | 21 M $\mu$     | (UNSAT) |
| <i>langford''</i> (9)  | 335       | 801     | 139 M $\mu$    | 20 M $\mu$    | 24 M $\mu$     | (UNSAT) |
| <i>langford</i> (13)   | 228       | 5875    | 71685 M $\mu$  | 45744 M $\mu$ | 295571 M $\mu$ | (UNSAT) |
| <i>langford'</i> (13)  | 502       | 1857    | 492992 M $\mu$ | 38589 M $\mu$ | 677815 M $\mu$ | (UNSAT) |
| <i>langford''</i> (13) | 795       | 1857    | 950719 M $\mu$ | 46398 M $\mu$ | 792757 M $\mu$ | (UNSAT) |
| <i>langford</i> (16)   | 352       | 11494   | 5 M $\mu$      | 52 M $\mu$    | 301 K $\mu$    | (SAT)   |
| <i>langford'</i> (16)  | 796       | 2928    | 12 M $\mu$     | 31 M $\mu$    | 418 K $\mu$    | (SAT)   |
| <i>langford''</i> (16) | 1264      | 2928    | 20 M $\mu$     | 38 M $\mu$    | 510 K $\mu$    | (SAT)   |
| <i>langford</i> (64)   | 6016      | 869650  | (huge)         | (bigger)      | 35 M $\mu$     | (SAT)   |
| <i>langford'</i> (64)  | 14704     | 53184   | (huger)        | (big)         | 73 M $\mu$     | (SAT)   |
| <i>langford''</i> (64) | 23488     | 53184   | (hugest)       | (biggest)     | 304 M $\mu$    | (SAT)   |

Algorithm D prefers *langford* to *langford'*, because it doesn't perform unit propagations very efficiently. Algorithm L, which excels at unit propagation, likes *langford'* better. Algorithm C also excels at unit propagation, but it exhibits peculiar behavior: It prefers *langford*, and on satisfiable instances it zooms in quickly to find a solution; but for some reason it runs *very* slowly on unsatisfiable instances when  $n \geq 10$ .

Another general principle is that short encodings—encodings with few variables and/or few clauses—are *not* necessarily better than longer encodings. For example, we often need to use Boolean variables to encode the value of a variable  $x$  that actually ranges over  $d > 2$  different values, say  $0 \leq x < d$ . In such cases it's natural to use the binary representation  $x = (x_{l-1} \dots x_0)_2$ , where  $l = \lceil \lg d \rceil$ , and to construct clauses based on the independent bits  $x_j$ ; but that representation, known as the *log encoding*, surprisingly turns out to be a bad idea in many cases unless  $d$  is large. A *direct encoding* with  $d$  binary variables  $x_0, x_1, \dots, x_{d-1}$ , where  $x_j = [x = j]$ , is often much better. And the *order encoding* with  $d - 1$  binary variables  $x^1, \dots, x^{d-1}$ , where  $x^j = [x \geq j]$ , is often better yet; this encoding was introduced in 1994 by J. M. Crawford and A. B. Baker [AAAI Conf. 12 (1994), 1092–1097]. In fact, exercise 408 presents an important

Langford's problem  
*langford''*( $n$ )  
 unit propagations  
 binary representation  
 log encoding  
 direct encoding  
 sparse encoding, see direct encoding  
 order encoding  
 Crawford  
 Baker

application where the order encoding is the method of choice even when  $d$  is 1000 or more! The order encoding is exponentially larger than the log encoding, yet it wins in this application because it allows the SAT solver to deduce consequences rapidly via unit propagation.

Graph coloring problems illustrate this principle nicely. When we tried early in this section to color a graph with  $d$  colors, we encoded the color of each vertex with a direct representation, (15); but we could have used binary notation for those colors. And we could also have used the order encoding, even though the numerical ordering of colors is irrelevant in the problem itself. With a log encoding, exercise 391 exhibits three distinct ways to enforce the constraint that adjacent vertices have different colors. With the order encoding, exercise 395 explains that it's easy to handle graph coloring. And there also are four ways to work with the direct encoding, namely (a) to insist on one color per vertex by including the at-most-one exclusion clauses (17); or (b) to allow multivalued (multicolored) vertices by omitting those clauses; or (c) to actually *welcome* multicolored vertices, by omitting (17) and forcing each color class to be a kernel, as suggested in answer 14; or (d) to include (17) but to replace the “preclusion” clauses (16) by so-called “support” clauses as explained in exercise 399.

These eight options can be compared empirically by trying to arrange 64 colored queens on a chessboard so that no queens of the same color appear in the same row, column, or diagonal. That task is possible with 9 colors, but not with 8. By symmetry we can prespecify the colors of all queens in the top row.

| encoding    | colors | variables | clauses | Algorithm L   | Algorithm C   |         |
|-------------|--------|-----------|---------|---------------|---------------|---------|
| univalued   | 8      | 512       | 7688    | 3333 M $\mu$  | 9813 M $\mu$  | (UNSAT) |
| multivalued | 8      | 512       | 5896    | 1330 M $\mu$  | 11997 M $\mu$ | (UNSAT) |
| kernel      | 8      | 512       | 6408    | 4196 M $\mu$  | 12601 M $\mu$ | (UNSAT) |
| support     | 8      | 512       | 13512   | 16796 M $\mu$ | 20990 M $\mu$ | (UNSAT) |
| log(a)      | 8      | 2376      | 5120    | (immense)     | 20577 M $\mu$ | (UNSAT) |
| log(b)      | 8      | 192       | 5848    | (enormous)    | 15033 M $\mu$ | (UNSAT) |
| log(c)      | 8      | 192       | 5848    | (enormous)    | 15033 M $\mu$ | (UNSAT) |
| order       | 8      | 448       | 6215    | 43615 M $\mu$ | 5122 M $\mu$  | (UNSAT) |
| univalued   | 9      | 576       | 8928    | 2907 M $\mu$  | 464 M $\mu$   | (SAT)   |
| multivalued | 9      | 576       | 6624    | 104 M $\mu$   | 401 M $\mu$   | (SAT)   |
| kernel      | 9      | 576       | 7200    | 93 M $\mu$    | 87 M $\mu$    | (SAT)   |
| support     | 9      | 576       | 15480   | 2103 M $\mu$  | 613 M $\mu$   | (SAT)   |
| log(a)      | 9      | 3168      | 6776    | (gigantic)    | 1761 M $\mu$  | (SAT)   |
| log(b)      | 9      | 256       | 6776    | (colossal)    | 1107 M $\mu$  | (SAT)   |
| log(c)      | 9      | 256       | 6584    | (mammoth)     | 555 M $\mu$   | (SAT)   |
| order       | 9      | 512       | 7008    | (monstrous)   | 213 M $\mu$   | (SAT)   |

(Each running time shown here is the median of nine runs, made with different random seeds.) It's clear from this data that the log encodings are completely unsuitable for Algorithm L; and even the order encoding confuses that algorithm's heuristics. But Algorithm L shines over Algorithm C with respect to most of the direct encodings. On the other hand, Algorithm C loves the order encoding, especially in the difficult unsatisfiable case.

unit propagation  
 Graph coloring problems  
 at-most-one  
 exclusion clauses  
 multivalued  
 kernel  
 “preclusion” clauses  
 “support” clauses  
 colored queens  
 chessboard  
 queens  
 median



And that’s not the end of the story. H. Tajima [M.S. thesis, Kobe University (2008)] and N. Tamura noticed that order encoding has another property, which trumps all other encodings with respect to graph coloring: Every  $k$ -clique of vertices  $\{v_1, \dots, v_k\}$  in a graph allows us to append two additional “hint clauses”

$$(\bar{v}_1^{d-k+1} \vee \dots \vee \bar{v}_k^{d-k+1}) \wedge (v_1^{k-1} \vee \dots \vee v_k^{k-1}) \tag{162}$$

to the clauses for  $d$ -coloring—because some vertex of the clique must have a color  $\leq d - k$ , and some vertex must have a color  $\geq k - 1$ . With these additional clauses, the running time to prove unsatisfiability of the 8-coloring problem drops drastically to just  $60 M\mu$  with Algorithm L, and to only  $13 M\mu$  with Algorithm C. We can even reduce it to just  $2 M\mu$  (!) by using that idea *twice* (see exercise 396).

The order encoding has several other nice properties, so it deserves a closer look. When we represent a value  $x$  in the range  $0 \leq x < d$  by the binary variables  $x^j = [x \geq j]$  for  $1 \leq j < d$ , we always have

$$x = x^1 + x^2 + \dots + x^{d-1}; \tag{163}$$

hence order encoding is often known as *unary representation*. The axiom clauses

$$(\bar{x}^{j+1} \vee x^j) \quad \text{for } 1 \leq j < d - 1 \tag{164}$$

are always included, representing the fact that  $x \geq j + 1$  implies  $x \geq j$  for each  $j$ ; these clauses force all the 1s to the left and all the 0s to the right. When  $d = 2$  the unary representation reduces to a one-bit encoding equal to  $x$  itself; when  $d = 3$  it’s a two-bit encoding with 00, 10, and 11 representing 0, 1, and 2.

We might not know all of the bits  $x^j$  of  $x$ ’s unary encoding while a problem is in the course of being solved. But if we do know that, say,  $x^3 = 1$  and  $x^7 = 0$ , then we know that  $x$  belongs to the interval  $[3..7)$ .

Suppose we know the unary representation of  $x$ . Then no calculation is necessary if we want to know the unary representation of  $y = x + a$ , when  $a$  is a constant, because  $y^j = x^{j-a}$ . Similarly,  $z = a - x$  is equivalent to  $z^j = \bar{x}^{a+1-j}$ ; and  $w = \lfloor x/a \rfloor$  is equivalent to  $w^j = x^{aj}$ . Out-of-bounds superscripts are easy to handle in formulas such as this, because  $x^i = 1$  when  $i \leq 0$  and  $x^i = 0$  when  $i \geq d$ . The special case  $\bar{x} = d - 1 - x$  is obtained by left-right reflection of  $\bar{x}^1 \dots \bar{x}^{j-1}$ :

$$(d - 1 - x)^j = (\bar{x})^j = \overline{x^{d-j}}. \tag{165}$$

If we are using the order encoding for two independent variables  $x$  and  $y$ , with  $0 \leq x, y < d$ , it’s similarly easy to encode the additional relation  $x \leq y + a$ :

$$x - y \leq a \iff x \leq y + a \iff \bigwedge_{j=\max(0, a+1)}^{\min(d-1, d+a)} (\bar{x}^j \vee y^{j-a}). \tag{166}$$

And there are analogous ways to place bounds on the sum,  $x + y$ :

$$x + y \leq a \iff x \leq \bar{y} + a + 1 - d \iff \bigwedge_{j=\max(0, a+2-d)}^{\min(d-1, a+1)} (\bar{x}^j \vee \bar{y}^{a+1-j}); \tag{167}$$

Tajima  
Tamura  
clique  
hint clauses  
unary representation  
axiom clauses  
ternary numbers  
complementation of unary representation

$$x + y \geq a \iff \bar{x} \leq y - a - 1 + d \iff \bigwedge_{j=\max(1, a+1-d)}^{\min(d, a)} (x^j \vee y^{a+1-j}). \quad (168)$$

2SAT  
lexicographic order  
comparison  
carries  
CNF  
eliminated  
circuit  
Tseytin encoding  
gate

In fact, exercise 405 shows that the general condition  $ax + by \leq c$  can be enforced with at most  $d$  binary clauses, when  $a, b$ , and  $c$  are constant. Any set of such relations, involving at most two variables per constraint, is therefore a 2SAT problem.

Relations between three or more order-encoded variables can also be handled without difficulty, as long as  $d$  isn't too large. For example, conditions such as  $x + y \leq z$  and  $x + y \geq z$  can be expressed with  $O(d \log d)$  clauses of length  $\leq 3$  (see exercise 407). Arbitrary linear inequalities can also be represented, in principle. But of course we shouldn't expect SAT solvers to compete with algebraic methods on problems that are inherently numerical.

Another constraint of great importance in the encoding of combinatorial problems is the relation of *lexicographic order*: Given two bit vectors  $x_1 \dots x_n$  and  $y_1 \dots y_n$ , we want to encode the condition  $(x_1 \dots x_n)_2 \leq (y_1 \dots y_n)_2$  as a conjunction of clauses. Fortunately there's a nice way to do this with just  $3n - 2$  ternary clauses involving  $n - 1$  auxiliary variables  $a_1, \dots, a_{n-1}$ , namely

$$\bigwedge_{k=1}^{n-1} ((\bar{x}_k \vee y_k \vee \bar{a}_{k-1}) \wedge (\bar{x}_k \vee a_k \vee \bar{a}_{k-1}) \wedge (y_k \vee a_k \vee \bar{a}_{k-1})) \wedge (\bar{x}_n \vee y_n \vee \bar{a}_{n-1}), \quad (169)$$

where ' $\bar{a}_0$ ' is omitted. For example, the clauses

$$(\bar{x}_1 \vee y_1) \wedge (\bar{x}_1 \vee a_1) \wedge (y_1 \vee a_1) \wedge (\bar{x}_2 \vee y_2 \vee \bar{a}_1) \wedge (\bar{x}_2 \vee a_2 \vee \bar{a}_1) \wedge (y_2 \vee a_2 \vee \bar{a}_1) \wedge (\bar{x}_3 \vee y_3 \vee \bar{a}_2)$$

assert that  $x_1 x_2 x_3 \leq y_1 y_2 y_3$ . And the same formula, but with the final term  $(\bar{x}_n \vee y_n \vee \bar{a}_{n-1})$  replaced by  $(\bar{x}_n \vee \bar{a}_{n-1}) \wedge (y_n \vee \bar{a}_{n-1})$ , works for the *strict* comparison  $x_1 \dots x_n < y_1 \dots y_n$ . These formulas arise by considering the carries that occur when  $(\bar{x}_1 \dots \bar{x}_n)_2 + (1 \text{ or } 0)$  is added to  $(y_1 \dots y_n)_2$ . (See exercise 414.)

The *general* problem of encoding a constraint on the Boolean variables  $x_1, \dots, x_n$  is the question of finding a family of clauses  $F$  that are satisfiable if and only if  $f(x_1, \dots, x_n)$  is true, where  $f$  is a given Boolean function. We usually introduce auxiliary variables  $a_1, \dots, a_m$  into the clauses of  $F$ , unless  $f$  can be expressed directly with a short CNF formula; thus the encoding problem is to find a "good" family  $F$  such that we have

$$f(x_1, \dots, x_n) = 1 \iff \exists a_1 \dots \exists a_m \bigwedge_{C \in F} C, \quad (170)$$

where each  $C$  is a clause on the variables  $\{a_1, \dots, a_m, x_1, \dots, x_n\}$ . The variables  $a_1, \dots, a_m$  can be eliminated by resolution as in (112), at least in principle, leaving us with a CNF for  $f$ —although that CNF might be huge. (See exercise 248.)

If there's a simple circuit that computes  $f$ , we know from (24) and exercise 42 that there's an equally simple "Tseytin encoding"  $F$ , with one auxiliary variable for each gate in the circuit. For example, suppose we want to encode the condition  $x_1 \dots x_n \neq y_1 \dots y_n$ . The shortest CNF expression for this function  $f(x_1, \dots, x_n, y_1, \dots, y_n)$  has  $2^n$  clauses (see exercise 415); but there's a simple

circuit (Boolean chain) with just  $n + 1$  gates:

$$a_1 \leftarrow x_1 \oplus y_1, \quad \dots, \quad a_n \leftarrow x_n \oplus y_n, \quad f \leftarrow a_1 \vee \dots \vee a_n.$$

Using (24) we get the  $4n$  clauses

$$\bigwedge_{j=1}^n ((\bar{x}_j \vee y_j \vee a_j) \wedge (x_j \vee \bar{y}_j \vee a_j) \wedge (x_j \vee y_j \vee \bar{a}_j) \wedge (\bar{x}_j \vee \bar{y}_j \vee \bar{a}_j)), \quad (171)$$

together with  $(a_1 \vee \dots \vee a_n)$ , as a representation of ‘ $x_1 \dots x_n \neq y_1 \dots y_n$ ’.

But this is overkill; D. A. Plaisted and S. Greenbaum have pointed out [*Journal of Symbolic Computation* **2** (1986), 293–304] that we can often avoid about half of the clauses in such situations. Indeed, only  $2n$  of the clauses (171) are necessary (and sufficient), namely the ones involving  $\bar{a}_j$ :

$$\bigwedge_{j=1}^n ((x_j \vee y_j \vee \bar{a}_j) \wedge (\bar{x}_j \vee \bar{y}_j \vee \bar{a}_j)). \quad (172)$$

The other clauses are “blocked” (see exercise 378) and unhelpful. Thus it’s a good idea to examine whether all of the clauses in a Tseytin encoding are really needed. Exercise 416 illustrates another interesting case.

An efficient encoding is possible also when  $f$  has a small BDD, and in general whenever  $f$  can be computed by a short branching program. Recall the example “Pi function” introduced in 7.1.1–(22); we observed in 7.1.2–(6) that it can be written  $((x_2 \wedge \bar{x}_4) \oplus \bar{x}_3) \wedge \bar{x}_1 \oplus x_2$ . Thus it has a 12-clause Tseytin encoding

$$(x_2 \vee \bar{a}_1) \wedge (\bar{x}_4 \vee \bar{a}_1) \wedge (\bar{x}_2 \vee x_4 \vee a_1) \wedge (x_3 \vee a_1 \vee a_2) \wedge (\bar{x}_3 \vee \bar{a}_1 \vee a_2) \wedge (\bar{x}_3 \vee a_1 \vee \bar{a}_2) \\ \wedge (x_3 \vee \bar{a}_1 \vee \bar{a}_2) \wedge (\bar{x}_1 \vee \bar{a}_3) \wedge (a_2 \vee \bar{a}_3) \wedge (x_1 \vee \bar{a}_2 \vee a_3) \wedge (x_2 \vee a_3) \wedge (\bar{x}_2 \vee \bar{a}_3).$$

The Pi function also has a short branching program, 7.1.4–(8), namely

$$I_8 = (\bar{1}? 7: 6), I_7 = (\bar{2}? 5: 4), I_6 = (\bar{2}? 0: 1), I_5 = (\bar{3}? 1: 0), \\ I_4 = (\bar{3}? 3: 2), I_3 = (\bar{4}? 1: 0), I_2 = (\bar{4}? 0: 1),$$

where the instruction ‘ $(\bar{v}? l: h)$ ’ means “If  $x_v = 0$ , go to  $I_l$ , otherwise go to  $I_h$ ,” except that  $I_0$  and  $I_1$  unconditionally produce the values 0 and 1. We can convert any such branching program into a sequence of clauses, by translating ‘ $I_j = (\bar{v}? l: h)$ ’ into

$$(\bar{a}_j \vee x_v \vee a_l) \wedge (\bar{a}_j \vee \bar{x}_v \vee a_h), \quad (173)$$

where  $a_0$  is omitted, and where any clauses containing  $a_1$  are dropped. We also omit  $\bar{a}_t$ , where  $I_t$  is the first instruction; in this example  $t = 8$ . (These simplifications correspond to asserting the unit clauses  $(\bar{a}_0) \wedge (a_1) \wedge (a_t)$ .) The branching program above therefore yields ten clauses,

$$(x_1 \vee a_7) \wedge (\bar{x}_1 \vee a_6) \wedge (\bar{a}_7 \vee x_2 \vee a_5) \wedge (\bar{a}_7 \vee \bar{x}_2 \vee a_4) \wedge (\bar{a}_6 \vee x_2) \\ \wedge (\bar{a}_5 \vee \bar{x}_3) \wedge (\bar{a}_4 \vee x_3 \vee a_3) \wedge (\bar{a}_4 \vee \bar{x}_3 \vee a_2) \wedge (\bar{a}_3 \vee \bar{x}_4) \wedge (\bar{a}_2 \vee x_4).$$

We can readily eliminate  $a_6, a_5, a_3, a_2$ , thereby getting a six-clause equivalent

$$(x_1 \vee a_7) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{a}_7 \vee x_2 \vee \bar{x}_3) \wedge (\bar{a}_7 \vee \bar{x}_2 \vee a_4) \wedge (\bar{a}_4 \vee x_3 \vee \bar{x}_4) \wedge (\bar{a}_4 \vee \bar{x}_3 \vee x_4);$$

and a preprocessor will simplify this to the four-clause CNF

$$(\bar{x}_1 \vee x_2) \wedge (x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4), \quad (174)$$

which appeared in exercise 7.1.1–19.

Exercise 417 explains why this translation scheme is valid. The method applies to *any* branching program whatsoever: The  $x$  variables can be tested in any order—that is, the  $v$ 's need not be decreasing as in a BDD; moreover, a variable may be tested more than once.

**Unit propagation and forcing.** The effectiveness of an encoding depends largely on how well that encoding avoids bad partial assignments to the variables. If we're trying to encode a Boolean condition  $f(x_1, x_2, \dots, x_n)$ , and if the tentative assignments  $x_1 \leftarrow 1$  and  $x_2 \leftarrow 0$  cause  $f$  to be false regardless of the values of  $x_3$  through  $x_n$ , we'd like the solver to deduce this fact without further ado, ideally by unit propagation once  $x_1$  and  $\bar{x}_2$  have been asserted. With a CDCL solver like Algorithm C, a quickly recognized conflict means a relatively short learned clause—and that's a hallmark of progress. Even better would be a situation in which unit propagation, after asserting  $x_1$ , would already force  $x_2$  to be true; and furthermore if unit propagation after  $\bar{x}_2$  would also force  $\bar{x}_1$ .

Such scenarios aren't equivalent to each other. For example, consider the clauses  $F = (\bar{x}_1 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$ . Then, using the notation ' $F \vdash_1 l$ ' to signify that  $F$  leads to  $l$  via unit propagation, we have  $F \mid x_1 \vdash_1 x_2$ , but  $F \mid \bar{x}_2 \not\vdash_1 \bar{x}_1$ . And with the clauses  $G = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$  we have  $G \mid x_1 \mid \bar{x}_2 \vdash_1 \epsilon$  (see Eq. (119)), but  $G \mid x_1 \not\vdash_1 x_2$  and  $G \mid \bar{x}_2 \not\vdash_1 \bar{x}_1$ .

Consider now the simple at-most-one constraint on just three variables,  $f(x_1, x_2, x_3) = [x_1 + x_2 + x_3 \leq 1]$ . We can try to represent  $f$  by proceeding methodically using the methods suggested above, either by constructing a circuit for  $f$  or by constructing  $f$ 's BDD. The first alternative (see exercise 420) yields

$$F = (x_1 \vee \bar{x}_2 \vee a_1) \wedge (\bar{x}_1 \vee x_2 \vee a_1) \wedge (x_1 \vee x_2 \vee \bar{a}_1) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_3 \vee \bar{a}_1); \quad (175)$$

the second approach (see exercise 421) leads to a somewhat different solution,

$$G = (x_1 \vee a_4) \wedge (\bar{x}_1 \vee a_3) \wedge (\bar{a}_4 \vee \bar{x}_2 \vee a_2) \wedge (\bar{a}_3 \vee x_2 \vee a_2) \wedge (\bar{a}_3 \vee \bar{x}_2) \wedge (\bar{a}_2 \vee \bar{x}_3). \quad (176)$$

But neither of these encodings is actually very good, because  $F \mid x_3 \not\vdash_1 \bar{x}_1$  and  $G \mid x_3 \not\vdash_1 \bar{x}_1$ . Much better is the encoding that we get from the general scheme of (18) and (19) in the case  $n = 3$ ,  $r = 1$ , namely

$$S = (\bar{a}_1 \vee a_2) \wedge (\bar{x}_1 \vee a_1) \wedge (\bar{x}_2 \vee a_2) \wedge (\bar{x}_2 \vee \bar{a}_1) \wedge (\bar{x}_3 \vee \bar{a}_2), \quad (177)$$

where  $a_1$  and  $a_2$  stand for  $s_1^1$  and  $s_2^1$ ; or the one obtained from (20) and (21),

$$B = (\bar{x}_3 \vee a_1) \wedge (\bar{x}_2 \vee a_1) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (\bar{a}_1 \vee \bar{x}_1), \quad (178)$$

where  $a_1$  stands for  $b_1^2$ . With either (177) or (178) we have  $S \mid x_i \vdash_1 \bar{x}_j$  and  $B \mid x_i \vdash_1 \bar{x}_j$  by unit propagation whenever  $i \neq j$ . And of course the obvious encoding for this particular  $f$  is best of all, because  $n$  is so small:

$$O = (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee \bar{x}_3). \quad (179)$$

preprocessor  
Unit propagation  
forcing-  
CDCL solver  
notation ' $F \vdash_1 l$ '  
at-most-one constraint  
circuit  
BDD

Suppose  $f(x_1, \dots, x_n)$  is a Boolean function that's represented by a family of clauses  $F$ , possibly involving auxiliary variables  $\{a_1, \dots, a_m\}$ , as in (170). We say that  $F$  is a *forcing* representation if we have

$$F|L \vdash l \quad \text{implies} \quad F|L \vdash_1 l \quad (180)$$

whenever  $L \cup l$  is a set of strictly distinct literals contained in  $\{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$ . In other words, if the partial assignment represented by  $L$  logically implies the truth of some other literal  $l$ , we insist that unit propagation alone should be able to deduce  $l$  from  $F|L$ . The auxiliary variables  $\{a_1, \dots, a_m\}$  are exempt from this requirement; only the potential forcings between *primary* variables  $\{x_1, \dots, x_n\}$  are supposed to be recognized easily when they occur.

(*Technical point:* If  $F|L \vdash \epsilon$ , meaning that  $F|L$  is unsatisfiable, we implicitly have  $F|L \vdash l$  for *all* literals  $l$ . In such a case (180) tells us that  $F|L \vdash_1 l$  and  $F|L \vdash_1 \bar{l}$  both hold; hence  $F|L$  can then be proved unsatisfiable by unit propagation alone.)

We've seen that the clauses  $S$  and  $B$  in (177) and (178) are forcing for the constraint  $[x_1 + x_2 + x_3 \leq 1]$ , but the clauses  $F$  and  $G$  in (175) and (176) are not. In fact, the clauses of (18) and (19) that led to (177) are *always* forcing, for the general cardinality constraint  $[x_1 + \dots + x_n \leq r]$ ; and so are the clauses of (20) and (21) that led to (178). (See exercises 429 and 430.) Moreover, the general at-most-one constraint  $[x_1 + \dots + x_n \leq 1]$  can be represented more efficiently by Heule's  $3(n-2)$  binary clauses and  $\lfloor (n-3)/2 \rfloor$  auxiliary variables (exercise 12), or with about  $n \lg n$  binary clauses and only  $\lceil \lg n \rceil$  auxiliary variables (exercise 394); both of these representations are forcing.

In general, we're glad to know as soon as possible when a variable's value has been forced by other values, because the variables of a large problem typically participate in many constraints simultaneously. If we know that  $x$  can't be 0 in constraint  $f$ , then we can often conclude that some other variable  $y$  can't be 1 in some other constraint  $g$ , if  $x$  appears in both  $f$  and  $g$ . There's lots of feedback.

On the other hand it might be worse to use a large representation  $F$  that is forcing than to use a small representation  $G$  that isn't, because additional clauses can make a SAT solver work harder. The tradeoffs are delicate, and they're difficult to predict in advance.

Every Boolean constraint  $f(x_1, \dots, x_n)$  has at least one forcing representation that involves no auxiliary variables. Indeed, it's easy to see that the *conjunctive prime form*  $F$  of  $f$ —the AND of all  $f$ 's prime clauses—is forcing.

Smaller representations are also often forcing, even without auxiliaries. For example, the simple constraint  $[x_1 \geq x_2 \geq \dots \geq x_n]$  has  $\binom{n}{2}$  prime clauses, namely  $(x_j \vee \bar{x}_k)$  for  $1 \leq j < k \leq n$ ; but only  $n-1$  of those clauses, the cases when  $k = j+1$  as in (164), are necessary and sufficient for forcing. Exercise 424 presents another, more-or-less random example.

In the worst case, all forcing representations of certain constraints are known to be huge, even when auxiliary variables are introduced (see exercise 428). But exercises 431–441 discuss many examples of useful and instructive forcing representations that require relatively few clauses.

auxiliary variables  
*forcing* representation  
*primary* variables  
 cardinality constraint  
 at-most-one constraint  
 Heule  
 feedback  
 representation  $F$   
 conjunctive prime form  
 prime clauses

We've glossed over an interesting technicality in definition (180), however: A sneaky person might actually construct a representation  $F$  that is absolutely useless in practice, even though it meets all of those criteria for forcing. For example, let  $G(a_1, \dots, a_m)$  be a family of clauses that are satisfiable—but only when the auxiliary variables  $a_j$  are set to extremely hard-to-find values. Then we might have  $f(x_1) = x_1$  and  $F = (x_1) \wedge G(a_1, \dots, a_m)$  (!). This defect in definition (180) was first pointed out by M. Gwynne and O. Kullmann [arXiv:1406.7398 [cs.CC] (2014), 67 pages], who have also traced the history of the subject.

To avoid such a glitch, we implicitly assume that  $F$  is an *honest* representation of  $f$ , in the following sense: Whenever  $L$  is a set of  $n$  literals that fully characterizes a solution  $x_1 \dots x_n$  to the constraint  $f(x_1, \dots, x_n) = 1$ , *the clauses  $F \mid L$  must be easy to satisfy*, using the SLUR algorithm of exercise 444. That algorithm is efficient because it does not backtrack. All of the examples in exercises 439–444 meet this test of honesty; indeed, the test is automatically passed whenever every clause of  $F$  contains at most one negated auxiliary variable.

Some authors have suggested that a SAT solver should branch only on primary variables  $x_i$ , rather than on auxiliary variables  $a_j$ , whenever possible. But an extensive study by M. Järvisalo and I. Niemelä [LNCS 4741 (2007), 348–363; *J. Algorithms* 63 (2008), 90–113] has shown that such a restriction is not advisable with Algorithm C, and it might lead to a severe slowdown.

**Symmetry breaking.** Sometimes we can achieve enormous speedup by exploiting symmetries. Consider, for example, the clauses for placing  $m+1$  pigeons into  $m$  holes, (106)–(107). We've seen in Lemma B and Theorem B that Algorithm C and other resolution-related methods cannot demonstrate the unsatisfiability of those clauses without performing exponentially many steps as  $m$  grows. However, the clauses are symmetrical with respect to pigeons; independently, they're also symmetrical with respect to holes: If  $\pi$  is any permutation of  $\{0, 1, \dots, m\}$  and if  $\rho$  is any permutation of  $\{1, 2, \dots, m\}$ , the transformation  $x_{jk} \mapsto x_{(j\pi)(k\rho)}$  for  $0 \leq j \leq m$  and  $1 \leq k \leq m$  leaves the set of clauses (106)–(107) unchanged. Thus the pigeonhole problem has  $(m+1)! m!$  symmetries.

We'll prove below that the symmetries on the holes allow us to assume safely that the hole-occupancy vectors are lexicographically ordered, namely that

$$x_{0k} x_{1k} \dots x_{mk} \leq x_{0(k+1)} x_{1(k+1)} \dots x_{m(k+1)}, \quad \text{for } 1 \leq k < m. \quad (181)$$

These constraints preserve satisfiability; and we know from (169) that they are readily expressed as clauses. Without the help of such additional clauses the running time of Algorithm C rises from 19 megamems for  $m = 7$  to 177 M $\mu$  for  $m = 8$ , and then to 3.5 gigamems and 86 G $\mu$  for  $m = 9$  and 10. But with (181), the same algorithm shows unsatisfiability for  $m = 10$  after only 1 megamem; and for  $m = 20$  and  $m = 30$  after only 284 M $\mu$  and 3.6 G $\mu$ , respectively.

Even better results occur when we order the *pigeon*-occupancy vectors:

$$x_{j1} x_{j2} \dots x_{jm} \leq x_{(j+1)1} x_{(j+1)2} \dots x_{(j+1)m}, \quad \text{for } 0 \leq j < m. \quad (182)$$

With these constraints added to (106) and (107), Algorithm C polishes off the case  $m = 10$  in just 69 *kilomems*. It can even handle  $m = 100$  in 133 M $\mu$ . This

Gwynne  
Kullmann  
*honest* representation  
SLUR algorithm  
backtrack  
negated auxiliary variable  
branch  
primary variables  
auxiliary variables  
Järvisalo  
Niemelä  
Symmetry breaking-  
symmetries  
pigeons  
permutation  
lexicographically ordered

remarkable improvement was achieved by adding only  $m^2 - m$  new variables and  $3m^2 - 2m$  new clauses to the original  $m^2 + m$  variables and  $(m+1) + (m^3 + m^2)/2$  clauses of (106) and (107). (Moreover, the reasoning that justifies (182) doesn't "cheat" by invoking the mathematical pigeonhole principle behind the scenes.)

Actually that's not all. The theory of columnwise symmetry (see exercise 498) also tells us that we're allowed to add the  $\binom{m}{2}$  simple binary clauses

$$(x_{(j-1)j} \vee \bar{x}_{(j-1)k}) \quad \text{for } 1 \leq j < k \leq m \quad (183)$$

to (106) and (107), instead of (182). This principle is rather weak in general; but it turns out to be ideally suited to pigeons: It reduces the running time for  $m = 100$  to just 21 megamems, although it needs no auxiliary variables whatsoever!

Of course the status of (106)–(107) has never been in doubt. Those clauses serve merely as training wheels because of their simplicity; they illustrate the fact that many symmetry-breaking strategies exist. Let's turn now to a more interesting problem, which has essentially the same symmetries, but with the roles of pigeons and holes played by "points" and "lines" instead. Consider a set of  $m$  points and  $n$  lines, where each line is a subset of points; we will require that no two points appear together in more than one line. (Equivalently, no two lines may intersect in more than one point.) Such a configuration may be called *quad-free*, because it is equivalent to an  $m \times n$  binary matrix  $(x_{ij})$  that contains no "quad," namely no  $2 \times 2$  submatrix of 1s; element  $x_{ij}$  means that point  $i$  belongs to line  $j$ . Quad-free matrices are obviously characterized by  $\binom{m}{2} \binom{n}{2}$  clauses,

$$(\bar{x}_{ij} \vee \bar{x}_{i'j'} \vee \bar{x}_{ij'} \vee \bar{x}_{i'j}), \quad \text{for } 1 \leq i < i' \leq m \text{ and } 1 \leq j < j' \leq n. \quad (184)$$

What is the maximum number of 1s in an  $m \times n$  quad-free matrix? [This question, when  $m = n$ , was posed by K. Zarankiewicz, *Colloquium Mathematicæ* **2** (1951), 301, who also considered how to avoid more general submatrices of 1s.] Let's call that value  $Z(m, n) - 1$ ; then  $Z(m, n)$  is the smallest  $r$  such that every  $m \times n$  matrix with  $r$  nonzero entries contains a quad.

We've actually encountered examples of this problem before, but in a disguised form. For example (see exercise 448), a Steiner triple system on  $v$  objects exists if and only if  $v$  is odd and there is a quad-free matrix with  $m = v$ ,  $n = v(v-1)/6$ , and  $r = v(v-1)/2$ . Other combinatorial block designs have similar characterizations.

Table 5 shows the values of  $Z(m, n)$  for small cases. These values were discovered by delicate combinatorial reasoning, without computer assistance; so it's instructive to see how well a SAT solver can compete against real intelligence.

The first interesting case occurs when  $m = n = 8$ : One can place 24 markers on a chessboard without forming a quad, but  $Z(8, 8) = 25$  markers is too many. If we simply add the cardinality constraints  $\sum_{i=1}^m \sum_{j=1}^n x_{ij} \geq r$  to (184), Algorithm C will quickly find a solution when  $m = n = 8$  and  $r = 24$ . But it bogs down when  $r = 25$ , requiring about 10 teramems to show unsatisfiability.

Fortunately we can take advantage of  $m! n!$  symmetries, which permute rows and columns without affecting quads. Exercise 495 shows that those symmetries

points  
lines  
quad-free  
binary matrix  
0-1 matrix  
submatrix  
Zarankiewicz  
avoiding submatrices  
 $Z(m, n)$   
Steiner triple system  
block designs  
chessboard  
cardinality constraints

**Table 5** $Z(m, n)$ , THE MINIMUM NUMBER OF 1S WITH (184) UNSATISFIABLE

| $n =$     | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| $m = 2:$  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| $m = 3:$  | 5  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| $m = 4:$  | 6  | 8  | 10 | 11 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
| $m = 5:$  | 7  | 9  | 11 | 13 | 15 | 16 | 18 | 19 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 |
| $m = 6:$  | 8  | 10 | 13 | 15 | 17 | 19 | 20 | 22 | 23 | 25 | 26 | 28 | 29 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 |
| $m = 7:$  | 9  | 11 | 14 | 16 | 19 | 22 | 23 | 25 | 26 | 28 | 29 | 31 | 32 | 34 | 35 | 37 | 38 | 40 | 41 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
| $m = 8:$  | 10 | 12 | 15 | 18 | 20 | 23 | 25 | 27 | 29 | 31 | 33 | 34 | 36 | 37 | 39 | 40 | 42 | 43 | 45 | 46 | 48 | 49 | 51 | 52 | 54 | 55 |
| $m = 9:$  | 11 | 13 | 16 | 19 | 22 | 25 | 27 | 30 | 32 | 34 | 37 | 38 | 40 | 41 | 43 | 44 | 46 | 47 | 49 | 50 | 52 | 53 | 55 | 56 | 58 | 59 |
| $m = 10:$ | 12 | 14 | 17 | 21 | 23 | 26 | 29 | 32 | 35 | 37 | 40 | 41 | 43 | 45 | 47 | 48 | 50 | 52 | 53 | 55 | 56 | 58 | 59 | 61 | 62 | 64 |
| $m = 11:$ | 13 | 15 | 18 | 22 | 25 | 28 | 31 | 34 | 37 | 40 | 43 | 45 | 46 | 48 | 51 | 52 | 54 | 56 | 58 | 60 | 61 | 63 | 64 | 66 | 67 | 69 |
| $m = 12:$ | 14 | 16 | 19 | 23 | 26 | 29 | 33 | 37 | 40 | 43 | 46 | 49 | 50 | 52 | 54 | 56 | 58 | 61 | 62 | 64 | 66 | 67 | 69 | 71 | 73 | 74 |
| $m = 13:$ | 15 | 17 | 20 | 24 | 28 | 31 | 34 | 38 | 41 | 45 | 49 | 53 | 54 | 56 | 58 | 60 | 62 | 65 | 67 | 68 | 80 | 72 | 74 | 76 | 79 | 80 |
| $m = 14:$ | 16 | 18 | 21 | 25 | 29 | 32 | 36 | 40 | 43 | 46 | 50 | 54 | 57 | 59 | 61 | 64 | 66 | 69 | 71 | 73 | 74 | 76 | 79 | 81 | 83 | 85 |
| $m = 15:$ | 17 | 19 | 22 | 26 | 31 | 34 | 37 | 41 | 45 | 48 | 52 | 56 | 59 | 62 | 65 | 68 | 70 | 73 | 76 | 78 | 79 | 81 | 83 | 86 | 87 | 89 |
| $m = 16:$ | 18 | 20 | 23 | 27 | 32 | 35 | 39 | 43 | 47 | 51 | 54 | 58 | 61 | 65 | 68 | 71 | 74 | 77 | 81 | 82 | 84 | 86 | 88 | 91 | 92 | 94 |

[References: R. K. Guy, in *Theory of Graphs*, Tihany 1966, edited by Erdős and Katona (Academic Press, 1968), 119–150; R. J. Nowakowski, Ph.D. thesis (Univ. of Calgary, 1978), 202.]

Guy  
Erdos  
Katona  
Nowakowski  
Satisfiability-preserving transformations–  
Symmetry  
conditional symmetry, see endomorphism  
endomorphism

allow us to add the lexicographic constraints

$$x_{i1}x_{i2}\dots x_{in} \geq x_{(i+1)1}x_{(i+1)2}\dots x_{(i+1)n}, \quad \text{for } 1 \leq i < m; \quad (185)$$

$$x_{1j}x_{2j}\dots x_{mj} \geq x_{1(j+1)}x_{2(j+1)}\dots x_{m(j+1)}, \quad \text{for } 1 \leq j < n. \quad (186)$$

(Increasing order, with ‘ $\leq$ ’ in place of ‘ $\geq$ ’, could also have been used, but decreasing order turns out to be better; see exercise 497.) The running time to prove unsatisfiability when  $r = 25$  now decreases dramatically, to only about 50 megamems. And it falls to 48 M $\mu$  if the lexicographic constraints are shortened to consider only the leading 4 elements of a row or column, instead of testing all 8.

The constraints of (185) and (186) are useful in satisfiable problems too—not in the easy case  $m = n = 8$ , when they aren’t necessary, but for example in the case  $m = n = 13$  when  $r = 52$ : Then they lead Algorithm C to a solution after about 200 gigamems, while it needs more than 18 teramems to find a solution without such help. (See exercise 449.)

**Satisfiability-preserving maps.** Let’s proceed now to the promised theory of symmetry breaking. In fact, we will do more: Symmetry is about *permutations* that preserve structural properties, but we will consider arbitrary *mappings* instead. Mappings are more general than permutations, because they needn’t be invertible. If  $x = x_1 \dots x_n$  is any potential solution to a satisfiability problem, our theory is based on transformations  $\tau$  that map  $x \mapsto x\tau = x'_1 \dots x'_n$ , where  $x\tau$  is required to be a solution whenever  $x$  is a solution.

In other words, if  $F$  is a family of clauses on  $n$  variables and if  $f(x) = [x \text{ satisfies } F]$ , then we are interested in all mappings  $\tau$  for which  $f(x) \leq f(x\tau)$ . Such a mapping is conventionally called an *endomorphism* of the solutions.\* If an

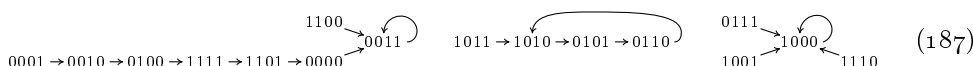
\* This word is a bit of a mouthful. But it’s easier to say “endomorphism” than to say “satisfiability-preserving transformation,” and you can use it to impress your friends. The term “conditional symmetry” has also been used by several authors in special cases.



endomorphism  $\tau$  is actually a permutation, it's called an *automorphism*. Thus, if there are  $K$  solutions to the problem, out of  $N = 2^n$  possibilities, the total number of mappings is  $N^N$ ; the total number of endomorphisms is  $K^K N^{N-K}$ ; and the total number of automorphisms is  $K!(N-K)!$ .

Notice that we don't require  $f(x)$  to be exactly equal to  $f(x\tau)$ . An endomorphism is allowed to map a nonsolution into a solution, and only  $K^K(N-K)^{N-K}$  mappings satisfy that stronger property. On the other hand, automorphisms always *do* satisfy  $f(x) = f(x\tau)$ ; see exercise 454.

Here, for instance, is a more-or-less random mapping when  $n = 4$ :



Exercises 455 and 456 discuss potential endomorphisms of this mapping.

In general there will be one or more *cycles*, and every element of a cycle is the root of an oriented tree that leads to it. For example, the cycles of (187) are (0011), (1010 0101 0110), and (1000).

Several different endomorphisms  $\tau_1, \tau_2, \dots, \tau_p$  are often known. In such cases it's helpful to imagine the digraph with  $2^n$  vertices that has arcs from each vertex  $x$  to its successors  $x\tau_1, x\tau_2, \dots, x\tau_p$ . This digraph will have one or more *sink components*, which are strongly connected components  $Y$  from which there is no escape: If  $x \in Y$  then  $x\tau_k \in Y$  for  $1 \leq k \leq p$ . (In the special case where each  $\tau_k$  is an automorphism, the sink components are traditionally called *orbits* of the automorphism group.) When  $p = 1$ , a sink component is the same as a cycle.

The clauses  $F$  are satisfiable if and only if  $f(x) = 1$  for at least one  $x$ . Such an  $x$  will lead to at least one sink component  $Y$ , all of whose elements will satisfy  $f(y) = 1$ . Thus it suffices to test satisfiability by checking just one element  $y$  in every sink component  $Y$ , to see if  $f(y) = 1$ .

Let's consider a simple problem based on the "sweep" of an  $m \times n$  matrix  $X = (x_{ij})$ , which is the largest diagonal sum of any  $t \times t$  submatrix:

$$\text{sweep}(X) = \max_{\substack{1 \leq i_1 < i_2 < \dots < i_t \leq m \\ 1 \leq j_1 < j_2 < \dots < j_t \leq n}} (x_{i_1 j_1} + x_{i_2 j_2} + \dots + x_{i_t j_t}). \quad (188)$$

When  $X$  is binary,  $\text{sweep}(X)$  is the length of the longest downward-and-rightward path that passes through its 1s. We can use satisfiability to decide whether such a matrix exists having  $\text{sweep}(X) \leq k$  and  $\sum_{i=1}^m \sum_{j=1}^n x_{ij} \geq r$ , given  $m, n, k$ , and  $r$ ; suitable clauses are exhibited in exercise 460. A solution with  $m = n = 10, k = 3$ , and  $r = 51$  appears at the right: It has 51 1s, but no four of them lie in a monotonic southeasterly path.

This problem has  $2^{mn}$  candidate matrices  $X$ , and experiments with small  $m$  and  $n$  suggest several endomorphisms that can be applied to such candidates without increasing the sweep.

- $\tau_1$ : If  $x_{ij} = 1$  and  $x_{i(j+1)} = 0$ , and if  $x_{i'j} = 0$  for  $1 \leq i' < i$ , we can set  $x_{ij} \leftarrow 0$  and  $x_{i(j+1)} \leftarrow 1$ .
- $\tau_2$ : If  $x_{ij} = 1$  and  $x_{(i+1)j} = 0$ , and if  $x_{ij'} = 0$  for  $1 \leq j' < j$ , we can set  $x_{ij} \leftarrow 0$  and  $x_{(i+1)j} \leftarrow 1$ .

automorphism  
 pi as source of "random"  
 cycles  
 oriented tree  
 digraph  
 sink components  
 strongly connected components  
 orbits  
 sweep  
 trace  
 submatrix

0000111111  
 0000100011  
 0000100111  
 0001101101  
 0111111001  
 1111100001  
 1010000011  
 1010000010  
 1110111110  
 1111100000

- $\tau_3$ : If the  $2 \times 2$  submatrix in rows  $\{i, i + 1\}$  and columns  $\{j, j + 1\}$  is  $\begin{smallmatrix} 11 \\ 10 \end{smallmatrix}$ , we can change it to  $\begin{smallmatrix} 01 \\ 11 \end{smallmatrix}$ .

These transformations are justified in exercise 462. They're sometimes applicable for several different  $i$  and  $j$ ; for instance,  $\tau_3$  could be used to change any of eight different  $2 \times 2$  submatrices in the example solution. In such cases we make an arbitrary decision, by choosing (say) the lexicographically smallest possible  $i$  and  $j$ .

The clauses that encode this problem have auxiliary variables besides  $x_{ij}$ ; but we can ignore the auxiliary variables when reasoning about endomorphisms.

Each of these endomorphisms either leaves  $X$  unchanged or replaces it by a lexicographically *smaller* matrix. *Therefore the sink components of  $\{\tau_1, \tau_2, \tau_3\}$  consist of the matrices  $X$  that are fixed points of all three transformations.* Hence we're allowed to append additional clauses, stating that neither  $\tau_1$  nor  $\tau_2$  nor  $\tau_3$  is applicable. For instance, transformation  $\tau_3$  is ruled out by the clauses

$$\bigwedge_{i=1}^{m-1} \bigwedge_{j=1}^{n-1} (\bar{x}_{ij} \vee \bar{x}_{i(j+1)} \vee \bar{x}_{(i+1)j} \vee x_{(i+1)(j+1)}), \quad (189)$$

which state that the submatrix  $\begin{smallmatrix} 11 \\ 10 \end{smallmatrix}$  doesn't appear. The clauses for  $\tau_1$  and  $\tau_2$  are only a bit more complicated (see exercise 461).

These additional clauses give interesting answers in satisfiable instances, although they aren't really helpful running-time-wise. On the other hand, they're spectacularly successful when the problem is *unsatisfiable*.

For example, we can show, without endomorphisms, that the case  $m = n = 10$ ,  $k = 3$ ,  $r = 52$  is impossible, and hence that any solution for  $r = 51$  is optimum; Algorithm C proves this after about 16 gigamems of work. Adding the clauses for  $\tau_1$  and  $\tau_2$ , but not  $\tau_3$ , increases the running time to  $23 \text{ G}\mu$ ; on the other hand the clauses for  $\tau_3$  without  $\tau_1$  or  $\tau_2$  reduce it to  $6 \text{ G}\mu$ . When we use all three endomorphisms simultaneously, however, the running time to prove unsatisfiability goes down to just  $3.5 \text{ megamems}$ , a speedup of more than 4500.

Even better is the fact that the fixed points of  $\{\tau_1, \tau_2, \tau_3\}$  actually have an extremely simple form — see exercise 463 — from which we can readily determine the answer by hand, without running the machine at all! Computer experiments have helped us to guess this result; but once we've proved it, we've solved infinitely many cases in one fell swoop. Theory and practice are synergistic.

Another interesting example arises when we want to test whether or not a given graph has a *perfect matching*, which is a set of nonoverlapping edges that exactly touch each vertex. We'll discuss beautiful, efficient algorithms for this problem in Sections 7.5.1 and 7.5.5; but it's interesting to see how well a simple-minded SAT solver can compete with those methods.

Perfect matching is readily expressible as a SAT problem whose variables are called ' $uv$ ', one for each edge  $u - v$ . Variables ' $uv$ ' and ' $vu$ ' are identical. Whenever the graph contains a 4-cycle  $v_0 - v_1 - v_2 - v_3 - v_0$ , we might include two of its edges  $\{v_0v_1, v_2v_3\}$  in the matching; but we could equally well have included  $\{v_1v_2, v_3v_0\}$  instead. Thus there's an endomorphism that says, "If  $v_0v_1 = v_2v_3 = 1$  (hence  $v_1v_2 = v_3v_0 = 0$ ), set  $v_0v_1 \leftarrow v_2v_3 \leftarrow 0$  and  $v_1v_2 \leftarrow v_3v_0 \leftarrow 1$ ."

auxiliary variables  
lexicographically  
swoop  
Theory and practice  
perfect matching  
exact cover by pairs+  
4-cycle

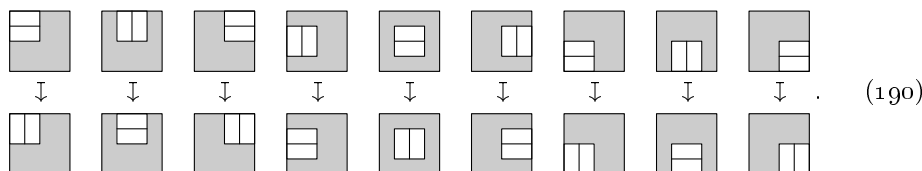
And we can carry this idea further: Let the edges be totally ordered in some arbitrary fashion, and for each edge  $uv$  consider all 4-cycles in which  $uv$  is the largest edge. In other words, we consider all cycles of the form  $u \text{---} v \text{---} u' \text{---} v' \text{---} u$  in which  $vu', u'v', v'u$  all precede  $uv$  in the ordering. If any such cycles exist, choose one of them arbitrarily, and let  $\tau_{uv}$  be one of two endomorphisms:

$$\begin{aligned} \tau_{uv}^-: & \text{ "If } uv = u'v' = 1, \text{ set } uv \leftarrow u'v' \leftarrow 0 \text{ and } vu' \leftarrow v'u \leftarrow 1." \\ \tau_{uv}^+: & \text{ "If } vu' = v'u = 1, \text{ set } uv \leftarrow u'v' \leftarrow 1 \text{ and } vu' \leftarrow v'u \leftarrow 0." \end{aligned}$$

Either  $\tau_{uv}^-$  or  $\tau_{uv}^+$  is stipulated, for each  $uv$ . Exercise 465 proves that a perfect matching is in the sink component of any such family of endomorphisms if and only if it is fixed by all of them. Therefore we need only search for fixed points.

For example, consider the problem of covering an  $m \times n$  board with dominoes. This is the problem of finding a perfect matching on the grid graph  $P_m \square P_n$ . The graph has  $mn$  vertices  $(i, j)$ , with  $m(n - 1)$  "horizontal" edges  $h_{ij}$  from  $(i, j)$  to  $(i, j + 1)$  and  $(m - 1)n$  "vertical" edges  $v_{ij}$  from  $(i, j)$  to  $(i + 1, j)$ . It has exactly  $(m - 1)(n - 1)$  4-cycles; and if we number the edges from left to right, no two 4-cycles have the same largest edge. Therefore we can construct  $(m - 1)(n - 1)$  endomorphisms, in each of which we're free to decide whether to allow a particular cycle to be filled by two horizontal dominoes or by two vertical ones.

Let's stipulate that  $h_{ij}$  and  $h_{(i+1)j}$  are allowed together only when  $i + j$  is odd;  $v_{ij}$  and  $v_{i(j+1)}$  are allowed together only when  $i + j$  is even. The nine endomorphisms when  $m = n = 4$  are then



And it's not difficult to see that only *one*  $4 \times 4$  domino covering is fixed by all nine. Indeed (exercise 466), the solution turns out to be unique for *all*  $m$  and  $n$ .

The famous problem of the "mutilated chessboard" asks for a domino covering when two opposite corner cells have been removed. This problem is unsatisfiable when  $m$  and  $n$  are both even, by exercise 7.1.4–213. But a SAT solver can't discover this fact quickly from the clauses alone, because there are many ways to get quite close to a solution; see the discussion following 7.1.4–(130). [S. Dantchev and S. Riis, in *FOCS* 42 (2001), 220–229, have proved in fact that every resolution refutation of these clauses requires  $2^{\Omega(n)}$  steps.]

When Algorithm C is presented with mutilated boards of sizes  $6 \times 6$ ,  $8 \times 8$ ,  $10 \times 10$ ,  $\dots$ ,  $16 \times 16$ , it needs respectively about  $55 \text{ K}\mu$ ,  $1.4 \text{ M}\mu$ ,  $31 \text{ M}\mu$ ,  $668 \text{ M}\mu$ ,  $16.5 \text{ G}\mu$ , and  $.91 \text{ T}\mu$  (that's *teramems*) to prove unsatisfiability. The even-odd endomorphisms typified by (190) come to our rescue, however: They narrow the search space spectacularly, reducing the respective running times to only  $15 \text{ K}\mu$ ,  $60 \text{ K}\mu$ ,  $135 \text{ K}\mu$ ,  $250 \text{ K}\mu$ ,  $470 \text{ K}\mu$ ,  $690 \text{ K}\mu$  (that's *kilomems*). They even can verify the unsatisfiability of a mutilated  $256 \times 256$  domino cover after fewer than  $4.2 \text{ G}\mu$  of calculation, exhibiting a growth rate of roughly  $O(n^3)$ .

dominoes  
grid graph  
mutilated chessboard  
Dantchev  
Riis  
resolution refutation

Endomorphisms can also speed up SAT solving in another important way:

**Theorem E.** *Let  $p_1 p_2 \dots p_n$  be any permutation of  $\{1, 2, \dots, n\}$ . If the Boolean function  $f(x_1, x_2, \dots, x_n)$  is satisfiable, then it has a solution such that  $x_{p_1} x_{p_2} \dots x_{p_n}$  is lexicographically less than or equal to  $x'_{p_1} x'_{p_2} \dots x'_{p_n}$  for every endomorphism of  $f$  that takes  $x_1 x_2 \dots x_n \mapsto x'_1 x'_2 \dots x'_n$ .*

lexicographically  
lex-leader  
signed permutations  
permuting variables and/or complementing  
literals  
order  $r$

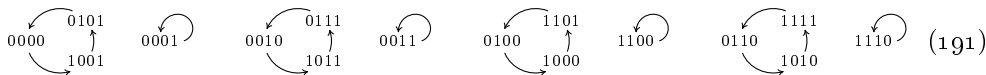
*Proof.* The lexicographically smallest solution of  $f$  has this property. ■

Maybe we shouldn't call this a "theorem"; it's an obvious consequence of the fact that endomorphisms always map solutions into solutions. But it deserves to be remembered and placed on some sort of pedestal, because we will see that it has many useful applications.

Theorem E is extremely good news, at least potentially, because every Boolean function has a *huge* number of endomorphisms. (See exercise 457.) On the other hand, there's a catch: We almost never *know* any of those endomorphisms until after we've solved the problem! Still, whenever we *do* happen to know one of the zillions of nontrivial endomorphisms that exist, we're allowed to add clauses that narrow the search. There's always a "lex-leader" solution that satisfies  $x_1 x_2 \dots x_n \leq x'_1 x'_2 \dots x'_n$ , if there's any solution at all.

A second difficulty that takes some of the shine away from Theorem E is the fact that most endomorphisms are too complicated to express neatly as clauses. What we really want is an endomorphism that's nice and simple, so that lexicographic ordering is equally simple.

Fortunately, such endomorphisms are often available; in fact, they're usually *automorphisms* — symmetries of the problem — defined by *signed permutations* of the variables. A signed permutation represents the operation of permuting variables and/or complementing them; for example, the signed permutation  $\bar{4}13\bar{2}$  stands for the mapping  $(x_1, x_2, x_3, x_4) \mapsto (x_{\bar{4}}, x_1, x_3, x_{\bar{2}}) = (\bar{x}_4, x_1, x_3, \bar{x}_2)$ . This operation transforms the states in a much more regular way than (187):



If  $\sigma$  takes the literal  $u$  into  $v$ , we write  $u\sigma = v$ ; and in such cases  $\sigma$  also takes  $\bar{u}$  into  $\bar{v}$ . Thus we always have  $\bar{u}\sigma = \overline{u\sigma}$ . We also write  $x\sigma$  for the result of applying  $\sigma$  to a sequence  $x$  of literals; for example,  $(x_1, x_2, x_3, x_4)\sigma = (\bar{x}_4, x_1, x_3, \bar{x}_2)$ . This mapping is a symmetry or automorphism of  $f(x)$  if and only if  $f(x) = f(x\sigma)$  for all  $x$ . Exercises 474 and 475 discuss basic properties of such symmetries; see also exercise 7.2.1.2–20.

Notice that a signed permutation can be regarded as an unsigned permutation of the  $2n$  literals  $\{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$ , and as such it can be written as a product of cycles. For instance, the symmetry  $\bar{4}13\bar{2}$  corresponds to the cycles  $(1\bar{4}2)(\bar{1}4\bar{2})(3)(\bar{3})$ . We can multiply signed permutations by multiplying these cycles in the normal way, just as in Section 1.3.3.

The product  $\sigma\tau$  of two symmetries  $\sigma$  and  $\tau$  is always a symmetry. Thus in particular, if  $\sigma$  is any symmetry, so are its powers  $\sigma^2, \sigma^3$ , etc. We say that  $\sigma$  has order  $r$  if  $\sigma, \sigma^2, \dots, \sigma^r$  are distinct and  $\sigma^r$  is the identity. A signed permutation

of order 1 or 2 is called a *signed involution*; this important special case arises if and only if  $\sigma$  is its own inverse ( $\sigma^2 = 1$ ).

It's clearly easier to work with permutations of  $2n$  literals than to work with permutations of  $2^n$  states  $x_1 \dots x_n$ . The main advantage of a signed permutation  $\sigma$  is that we can test whether or not  $\sigma$  preserves the family  $F$  of clauses in a satisfiability problem. If it does, we can be sure that  $\sigma$  also is an automorphism when it acts on all  $2^n$  states. (See exercise 492.)

Let's go back to the example *waerden*(3, 10; 97) that we've often discussed above. These clauses have an obvious symmetry, which takes  $x_1 x_2 \dots x_{97} \mapsto x_{97} x_{96} \dots x_1$ . If we don't break this symmetry, Algorithm C typically verifies unsatisfiability after about 530 M $\mu$  of computation. Now Theorem E tells us that we can also assert that  $x_1 x_2 x_3 \leq x_{97} x_{96} x_{95}$ , say; but that symmetry-breaker doesn't really help at all, because  $x_1$  has very little influence on  $x_{97}$ . Fortunately, however, Theorem E allows us to choose *any* permutation  $p_1 p_2 \dots p_n$  on which to base lexicographic comparisons. For example, we can assert that  $x_{48} x_{47} x_{46} \dots \leq x_{50} x_{51} x_{52} \dots$  — provided that we don't *also* require  $x_1 x_2 x_3 \dots \leq x_{97} x_{96} x_{95} \dots$ . (One fixed global ordering must be used, but the endomorphs can be arbitrary.)

Even the simple assertion that  $x_{48} \leq x_{50}$ , which is the clause '4850', cuts the running time down to about 410 M $\mu$ , because this new clause combines nicely with the existing clauses 46 48 50, 48 49 50, 48 50 52 to yield the helpful binary clauses 46 50, 49 50, 50 52. If we go further and assert that  $x_{48} x_{47} \leq x_{50} x_{51}$ , the running time improves to 345 M $\mu$ . And the next steps  $x_{48} x_{47} x_{46} \leq x_{50} x_{51} x_{52}$ ,  $\dots$ ,  $x_{48} x_{47} x_{46} x_{45} x_{44} x_{43} \leq x_{50} x_{51} x_{52} x_{53} x_{54} x_{55}$  take us down to 290 M $\mu$ , then 260 M $\mu$ , 235 M $\mu$ , 220 M $\mu$ ; we've saved more than half of the running time by exploiting a single reflection symmetry! Only 16 simple additional clauses, namely

$$\overline{48}50, \overline{48}a_1, 50a_1, \overline{47}51\overline{a}_1, \overline{47}a_2\overline{a}_1, 51a_2\overline{a}_1, \overline{46}52\overline{a}_2, \dots, \overline{43}55\overline{a}_5$$

are needed to get this speedup, using the efficient encoding of lex order in (169).

Of course all good things come to an end, and we've now reached the point of diminishing returns: Further clauses to assert that  $x_{48} x_{47} \dots x_{42} \leq x_{50} x_{51} \dots x_{56}$  in the *waerden*(3, 10; 97) problem turn out to be counterproductive.

A wonderful simplification occurs when a symmetry  $\sigma$  is a signed involution that has comparatively few 2-cycles. Suppose, for example, that  $\sigma = 5\overline{3}\overline{2}41\overline{6}\overline{9}\overline{8}\overline{7}$ ; in cycle form this is  $(1\overline{5})(\overline{1}\overline{5})(\overline{2}\overline{3})(\overline{2}\overline{3})(4)(\overline{4})(\overline{6}\overline{6})(\overline{7}\overline{9})(\overline{7}\overline{9})(\overline{8}\overline{8})$ . Then the lexicographic relation  $x = x_1 \dots x_9 \leq x'_1 \dots x'_9 = x\sigma$  holds if and only if  $x_1 x_2 x_6 \leq x_5 \overline{x}_3 \overline{x}_6$ . The reason is clear, once we look closer (see F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, *IEEE Trans. CAD-22* (2003), 1117–1137, §III.C): The relation  $x_1 \dots x_9 \leq x'_1 \dots x'_9$  means, in this case, “ $x_1 \leq x_5$ ; if  $x_1 = x_5$  then  $x_2 \leq \overline{x}_3$ ; if  $x_1 = x_5$  and  $x_2 = \overline{x}_3$  then  $x_3 \leq \overline{x}_2$ ; if  $x_1 = x_5$ ,  $x_2 = \overline{x}_3$ , and  $x_3 = \overline{x}_2$  then  $x_4 \leq x_4$ ; if  $x_1 = x_5$ ,  $x_2 = \overline{x}_3$ ,  $x_3 = \overline{x}_2$ , and  $x_4 = x_4$  then  $x_5 \leq x_1$ ; if  $x_1 = x_5$ ,  $x_2 = \overline{x}_3$ ,  $x_3 = \overline{x}_2$ ,  $x_4 = x_4$ , and  $x_5 = x_1$  then  $x_6 \leq \overline{x}_6$ ; if  $x_1 = x_5$ ,  $x_2 = \overline{x}_3$ ,  $x_3 = \overline{x}_2$ ,  $x_4 = x_4$ ,  $x_5 = x_1$ , and  $x_6 = \overline{x}_6$  then we're done for.” With this expanded description the simplifications are obvious.

In general this reasoning allows us to improve Theorem E as follows:

**Corollary E.** Let  $p_1 p_2 \dots p_n$  be any permutation of  $\{1, 2, \dots, n\}$ . For every signed involution  $\sigma$  that is a symmetry of clauses  $F$ , we can write  $\sigma$  in cycle form

$$(p_{i_1} \pm p_{j_1})(\bar{p}_{i_1} \mp p_{j_1})(p_{i_2} \pm p_{j_2})(\bar{p}_{i_2} \mp p_{j_2}) \dots (p_{i_t} \pm p_{j_t})(\bar{p}_{i_t} \mp p_{j_t}) \quad (192)$$

with  $i_1 < j_1, i_2 < j_2, \dots, i_t < j_t, i_1 < i_2 < \dots < i_t$ , and with  $(\bar{p}_{i_k} \mp p_{j_k})$  omitted when  $i_k = j_k$ ; and we're allowed to append clauses to  $F$  that assert the lexicographic relation  $x_{p_{i_1}} x_{p_{i_2}} \dots x_{p_{i_q}} \leq x_{\pm p_{j_1}} x_{\pm p_{j_2}} \dots x_{\pm p_{j_q}}$ , where  $q = t$  or  $q$  is the smallest  $k$  with  $i_k = j_k$ . ■

In the common case when  $\sigma$  is an ordinary signless involution, all of the signs can be eliminated here; we simply assert that  $x_{p_{i_1}} \dots x_{p_{i_t}} \leq x_{p_{j_1}} \dots x_{p_{j_t}}$ .

This involution principle justifies all of the symmetry-breaking techniques that we used above in the pigeonhole and quad-free matrix problems. See, for example, the details discussed in exercise 495.

The idea of breaking symmetry by appending clauses was pioneered by J.-F. Puget [LNCS 689 (1993), 350–361], then by J. Crawford, M. Ginsberg, E. Luks, and A. Roy [Int. Conf. Knowledge Representation and Reasoning 5 (1998), 148–159], who considered unsigned permutations only. They also attempted to discover symmetries algorithmically from the clauses that were given as input. Experience has shown, however, that useful symmetries can almost always be better supplied by a person who understands the structure of the underlying problem.

Indeed, symmetries are often “semantic” rather than “syntactic.” That is, they are symmetries of the underlying Boolean function, but not of the clauses themselves. In the Zarankiewicz problem about quad-free matrices, for example, we appended efficient cardinality clauses to ensure that  $\sum x_{ij} \geq r$ ; that condition is symmetric under row and column swaps, but the clauses are not.

In this connection it may also be helpful to mention the *monkey wrench principle*: All of the techniques by which we've proved quickly that the pigeonhole clauses are unsatisfiable would have been useless if there had been one more clause such as  $(x_{01} \vee x_{11} \vee \bar{x}_{22})$ ; that clause would have destroyed the symmetry!

We conclude that we're allowed to *remove* clauses from  $F$  until reaching a subset of clauses  $F_0$  for which symmetry-breakers  $S$  can be added. If  $F = F_0 \cup F_1$ , and if  $F_0$  is satisfiable  $\iff F_0 \cup S$  is satisfiable, then  $F_0 \cup S \vdash \epsilon \implies F \vdash \epsilon$ .

**One hundred test cases.** And now —ta da!— let's get to the climax of this long story, by looking at how our SAT solvers perform when presented with 100 moderately challenging instances of the satisfiability problem. The 100 sets of clauses summarized on the next two pages come from a wide variety of different applications, many of which were discussed near the beginning of this section, while others appear in the exercises below.

Every test case has a code name, consisting of a letter and a digit. Table 6 characterizes each problem and also shows exactly how many variables, clauses, and total literals are involved. For example, the description of problem A1 ends with '2043<sub>[24772]</sub><sup>[55195]</sup>U'; this means that A1 consists of 24772 clauses on 2043 variables, having 55195 literals altogether, and those clauses are unsatisfiable. Furthermore, since '24772' is underlined, all of A1's clauses have length 3 or less.

Puget  
Crawford  
Ginsberg  
Luks  
Roy  
Zarankiewicz problem  
quad-free  
cardinality clauses  
monkey wrench principle  
pigeonhole clauses  
test cases—

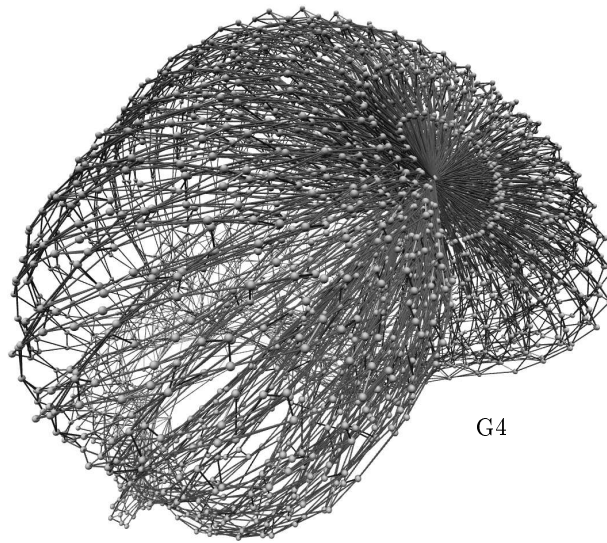
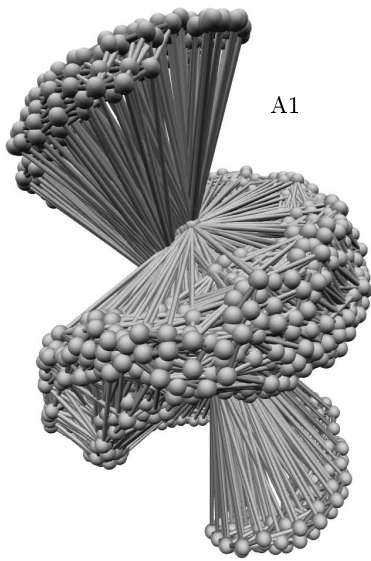
**Table 6**  
CAPSULE SUMMARIES OF THE HUNDRED TEST CASES

|  |   |
|--|---|
| <p><b>A1.</b> Find <math>x = x_1x_2 \dots x_{99}</math> with <math>\nu x = 27</math> and no three equally spaced 1s. (See exercise 31.)<br/>2043 24772 55195 U</p> <p><b>A2.</b> Like A1, but <math>x_1x_2 \dots x_{100}</math>.<br/>2071 25197 56147 S</p> <p><b>B1.</b> Cover a mutilated <math>10 \times 10</math> board with 49 dominoes, <i>without</i> using extra clauses to break symmetry.<br/>176 572 1300 U</p> <p><b>B2.</b> Like B1, but a <math>12 \times 12</math> board with 71 dominoes.<br/>260 856 1948 U</p> <p><b>C1.</b> Find an 8-step Boolean chain that computes <math>(z_2z_1z_0)_2 = x_1 + x_2 + x_3 + x_4</math>. (See exercise 479(a).)<br/>384 16944 66336 U</p> <p><b>C2.</b> Find a 7-step Boolean chain that computes the modified full adder functions <math>z_1, z_2, z_3</math> in exercise 481(b).<br/>469 26637 100063 U</p> <p><b>C3.</b> Like C2, but with 8 steps.<br/>572 33675 134868 S</p> <p><b>C4.</b> Find a 9-step Boolean chain that computes <math>z_i</math> and <math>z_r</math> in the mod-3 addition problem of exercise 480(b).<br/>678 45098 183834 S</p> <p><b>C5.</b> Connect A to A, ..., J to J in Dudeney's puzzle of exercise 392, (<i>iv</i>).<br/>1980 22518 70356 S</p> <p><b>C6.</b> Like C5, but move the J in row 8 from column 4 to column 5.<br/>1980 22518 70356 U</p> <p><b>C7.</b> Given binary strings <math>s_1, \dots, s_{50}</math> of length 200, randomly generated at distances <math>\leq r_j</math> from some string <math>x</math>, find <math>x</math> (see exercise 502).<br/>65719 577368 1659623 S</p> <p><b>C8.</b> Given binary strings <math>s_1, \dots, s_{40}</math> of length 500, inspired by biological data, find a string at distance <math>\leq 42</math> from each of them.<br/>123540 909120 2569360 U</p> <p><b>C9.</b> Like C8, but at distance <math>\leq 43</math>.<br/>124100 926200 2620160 S</p> <p><b>D1.</b> Satisfy <i>factor_fifo</i>(18, 19, 111111111111). (See exercise 41.)<br/>1940 6374 16498 U</p> <p><b>D2.</b> Like D1, but <i>factor_lifo</i>.<br/>1940 6374 16498 U</p> <p><b>D3.</b> Like D1, but (19, 19, 111111111111).<br/>2052 6745 17461 S</p> <p><b>D4.</b> Like D2, but (19, 19, 111111111111).<br/>2052 6745 17461 S</p> <p><b>D5.</b> Solve <math>(x_1 \dots x_9)_2 \times (y_1 \dots y_9)_2 \neq (x_1 \dots x_9)_2 \times' (y_1 \dots y_9)_2</math>, with two copies of the <i>same</i> Dadda multiplication circuit.<br/>864 2791 7236 U</p> <p><b>E0.</b> Find an Erdős discrepancy pattern <math>x_1 \dots x_{500}</math> (see exercise 482).<br/>1603 9157 27469 S</p> <p><b>E1.</b> Like E0, but <math>x_1 \dots x_{750}</math>.<br/>2556 14949 44845 S</p> <p><b>E2.</b> Like E0, but <math>x_1 \dots x_{1000}</math>.<br/>3546 21035 63103 S</p> <p><b>F1.</b> Satisfy <i>fsnark</i>(99). (See exercise 176.)<br/>1782 4161 8913 U</p> | <p><b>F2.</b> Like F1, but without the clauses <math>(\bar{e}_{1,3} \vee \bar{f}_{99,3}) \wedge (\bar{f}_{1,1} \vee \bar{e}_{2,1})</math>.<br/>1782 4159 8909 S</p> <p><b>G1.</b> Win Late Binding Solitaire with the "most difficult winnable deal" in answer 486.<br/>1242 22617 65593 S</p> <p><b>G2.</b> Like G1, but with the most difficult <i>unwinnable</i> deal.<br/>1242 22612 65588 U</p> <p><b>G3.</b> Find a test pattern for the fault "<math>B_{43}^{43}</math> stuck at 0" in <i>prod</i>(16, 32).<br/>3498 11337 29097 S</p> <p><b>G4.</b> Like G3, but for the fault "<math>D_{34}^{13,9}</math> stuck at 0."<br/>3502 11349 29127 S</p> <p><b>G5.</b> Find a <math>7 \times 15</math> array <math>X_0</math> leading to <math>X_3 = \mathbf{LIFE}</math> as in Fig. 35, having at most 38 live cells.<br/>7150 28508 71873 U</p> <p><b>G6.</b> Like G5, but at most 39 live cells.<br/>7152 28536 71956 S</p> <p><b>G7.</b> Like G5, but <math>X_4 = \mathbf{LIFE}</math> and <math>X_0</math> can be arbitrary.<br/>8725 33769 84041 U</p> <p><b>G8.</b> Find a configuration in the Game of Life that proves <math>f^*(7, 7) = 28</math> (see exercise 83).<br/>97909 401836 1020174 S</p> <p><b>K0.</b> Color the <math>8 \times 8</math> queen graph with 8 colors, using the direct encoding (15) and (16), also forcing the colors of all vertices in the top row.<br/>512 5896 12168 U</p> <p><b>K1.</b> Like K0, but with the exclusion clauses (17) also.<br/>512 7688 15752 U</p> <p><b>K2.</b> Like K1, but with kernel clauses instead of (17) (see answer 14).<br/>512 6408 24328 U</p> <p><b>K3.</b> Like K1, but with support clauses instead of (16) (see exercise 399).<br/>512 13512 97288 U</p> <p><b>K4.</b> Like K1, but using the order encoding for colors.<br/>448 6215 21159 U</p> <p><b>K5.</b> Like K4, but with the hint clauses (162) appended.<br/>448 6299 21663 U</p> <p><b>K6.</b> Like K5, but with double clique hints (exercise 396).<br/>896 8559 27927 U</p> <p><b>K7.</b> Like K1, but with the log encoding of exercise 391(a).<br/>2376 5120 15312 U</p> <p><b>K8.</b> Like K1, but with the log encoding of exercise 391(b).<br/>192 5848 34968 U</p> <p><b>L1.</b> Satisfy <i>langford</i>(10).<br/>130 2437 5204 U</p> <p><b>L2.</b> Satisfy <i>langford'</i>(10).<br/>273 1020 2370 U</p> <p><b>L3.</b> Satisfy <i>langford</i>(13).<br/>228 5875 12356 U</p> <p><b>L4.</b> Satisfy <i>langford'</i>(13).<br/>502 1857 4320 U</p> <p><b>L5.</b> Satisfy <i>langford</i>(32).<br/>1472 102922 210068 S</p> <p><b>L6.</b> Satisfy <i>langford'</i>(32).<br/>3512 12768 29760 S</p> <p><b>L7.</b> Satisfy <i>langford</i>(64).<br/>6016 869650 1756964 S</p> <p><b>L8.</b> Satisfy <i>langford'</i>(64).<br/>14704 53184 124032 S</p> <p><b>M1.</b> Color the McGregor graph of order 10 (Fig. 33) with 4 colors, using one color at most 6 times, via the cardinality constraints (18) and (19).<br/>1064 2752 6244 U</p> |
|--|---|

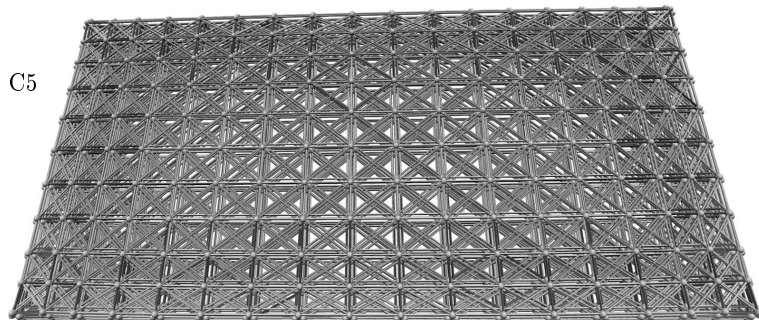
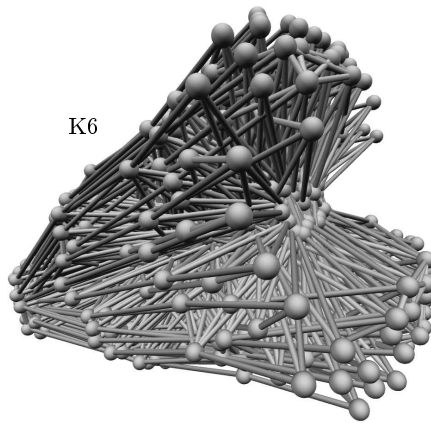
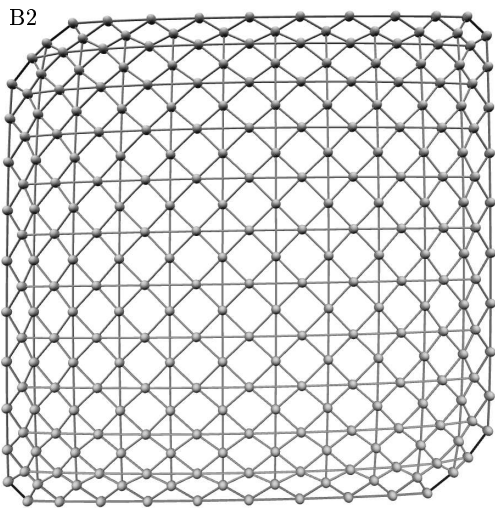
test cases, capsule summaries+ many items are indexed here but they don't show in margin!

- M2.** Like M1, but via (20) and (21).  
814|2502|5744|U
- M3.** Like M1, but at most 7 times.  
1161|2944|6726|S
- M4.** Like M2, but at most 7 times.  
864|2647|6226|S
- M5.** Like M4, but order 16 and at most 11 times.  
2256|7801|18756|U
- M6.** Like M5, but at most 12 times.  
2288|8080|19564|S
- M7.** Color the McGregor graph of order 9 with 4 colors, and with at least 18 regions doubly colored (see exercise 19).  
952|4539|13875|S
- M8.** Like M7, but with at least 19 regions.  
952|4540|13877|U
- N1.** Place 100 nonattacking queens on a  $100 \times 100$  board.  
10000|1151800|2313400|S
- O1.** Solve a random open shop scheduling problem with 8 machines and 8 jobs, in 1058 units of time.  
50846|557823|1621693|U
- O2.** Like O1, but in 1059 units.  
50901|558534|1623771|S
- P0.** Satisfy (99), (100), and (101) for  $m = 20$ , thereby exhibiting a poset of size 20 with no maximal element.  
400|7260|22080|U
- P1.** Like P0, but with  $m = 14$  and using only the clauses of exercise 228.  
196|847|2667|U
- P2.** Like P0, but with  $m = 12$  and using only the clauses of exercise 229.  
144|530|1674|U
- P3.** Like P2, but omitting the clause  $(\bar{x}_{31} \vee \bar{x}_{16} \vee x_{36})$ .  
144|529|1671|S
- P4.** Like P3, but with  $m = 20$ .  
400|2509|7827|S
- Q0.** Like K0, but with 9 colors.  
576|6624|13688|S
- Q1.** Like K1, but with 9 colors.  
576|8928|18296|S
- Q2.** Like K2, but with 9 colors.  
576|7200|27368|S
- Q3.** Like K3, but with 9 colors.  
576|15480|123128|S
- Q4.** Like K4, but with 9 colors.  
512|7008|24200|S
- Q5.** Like K5, but with 9 colors.  
512|7092|24704|S
- Q6.** Like K6, but with 9 colors.  
1024|9672|31864|S
- Q7.** Like K7, but with 9 colors.  
3168|6776|20800|S
- Q8.** Like K8, but with 9 colors.  
256|6776|52832|S
- Q9.** Like Q8, but with the log encoding of exercise 391(c).  
256|6584|42256|S
- R1.** Satisfy *rand*(3, 1061, 250, 314159).  
250|1061|3183|S
- R2.** Satisfy *rand*(3, 1062, 250, 314159).  
250|1062|3186|U
- S1.** Find a 4-term disjunctive normal form on  $\{x_1, \dots, x_{20}\}$  that differs from (27) but agrees with it at 108 random training points.  
356|4229|16596|S
- S2.** Like S1, but at 109 points.  
360|4310|16760|U
- S3.** Find a sorting network on nine elements that begins with the comparators [1:6][2:7][3:8][4:9] and finishes in five more parallel rounds. (See exercise 64.)  
5175|85768|255421|U
- S4.** Like S3, but in six more rounds.  
6444|107800|326164|S
- T1.** Find a  $24 \times 100$  tatami tiling that spells ‘TATAMI’ as in exercise 118.  
2874|10527|26112|S
- T2.** Like T1, but  $24 \times 106$  and the ‘I’ should have serifs.  
3048|11177|27724|U
- T3.** Solve the TAOCP problem of exercise 389 with only 4 knight moves.  
3752|12069|27548|U
- T4.** Like T3, but with 5 knight moves.  
3756|12086|27598|S
- T5.** Find the pixel in row 5, column 18 of Fig. 37(c), the lexicographically last solution to the Cheshire Tom problem.  
8837|39954|100314|S
- T6.** Like T5, but column 19.  
8837|39955|100315|U
- T7.** Solve the run-count extension of the Cheshire Tom problem (see exercise 117).  
25734|65670|167263|S
- T8.** Like T7, but find a solution that differs from Fig. 36.  
25734|65671|167749|U
- W1.** Satisfy *waarden*(3, 10; 97).  
97|2779|11662|U
- W2.** Satisfy *waarden*(3, 13; 159).  
159|7216|31398|S
- W3.** Satisfy *waarden*(5, 5; 177).  
177|7656|38280|S
- W4.** Satisfy *waarden*(5, 5; 178).  
178|7744|38720|U
- X1.** Prove that the “taking turns” protocol (43) gives mutual exclusion for at least 100 steps.  
1010|3612|10614|U
- X2.** Prove that assertions  $\Phi$  for the four-bit protocol of exercise 101, analogous to (50), are invariant.  
129|354|926|U
- X3.** Prove that Bob won’t starve in 36 steps, assuming the  $\Phi$  of X2.  
1652|10552|28971|U
- X4.** Prove that there’s a simple 36-step path with the four-bit protocol, assuming the  $\Phi$  of X2.  
22199|50264|130404|S
- X5.** Like X4, but 37 steps.  
23388|52822|137034|U
- X6.** Like X1, but with Peterson’s protocol (49) instead of (43).  
2218|8020|23222|U
- X7.** Prove that there’s a simple 54-step path with protocol (49).  
26450|56312|147572|S
- X8.** Like X7, but 55 steps.  
27407|58317|152807|U

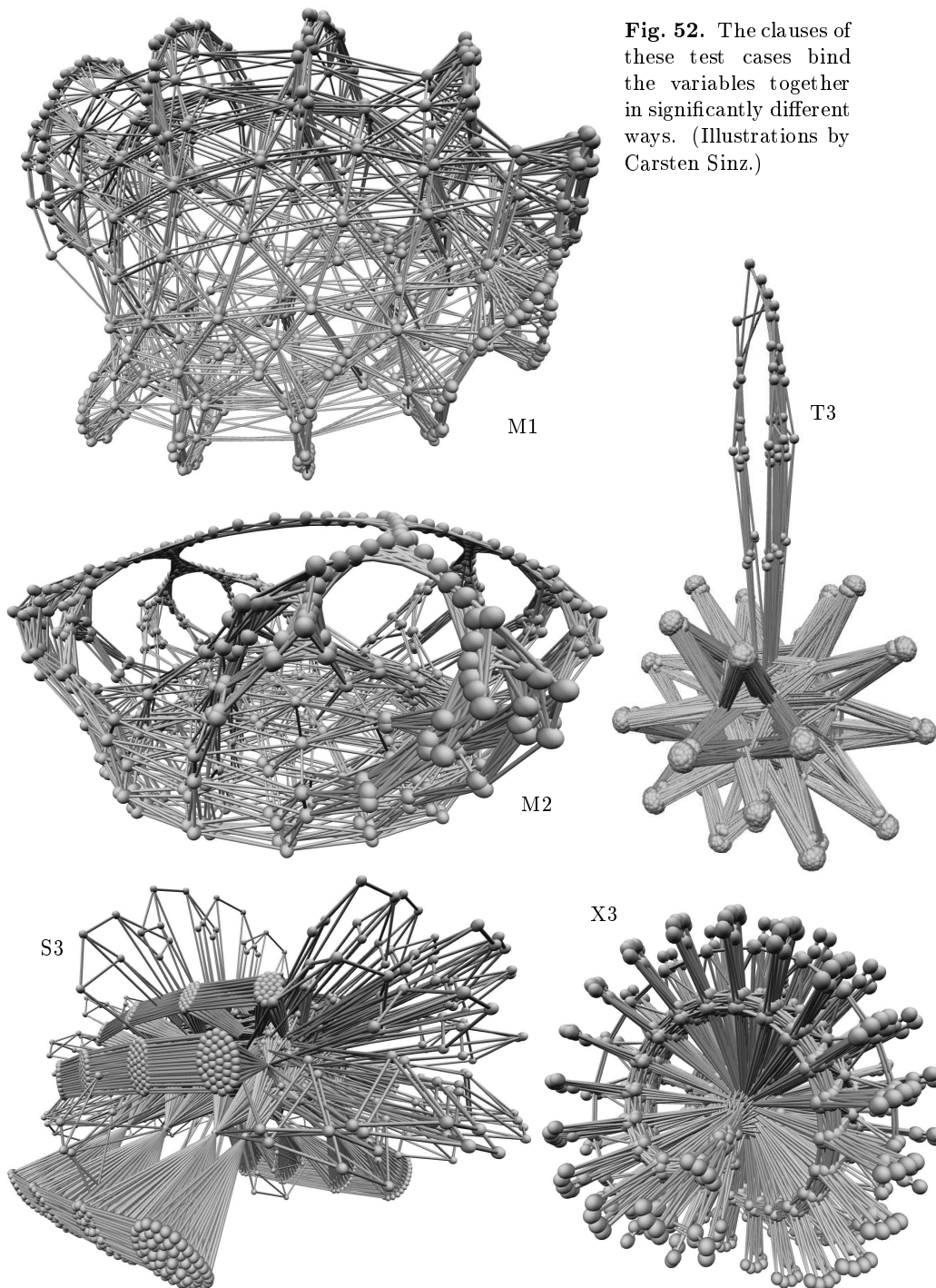




menagerie+  
graph layout++  
geek art+  
visualizations++  
3D visualizations++  
variable interaction graphs++



Sinz



**Fig. 52.** The clauses of these test cases bind the variables together in significantly different ways. (Illustrations by Carsten Sinz.)

Of course we can't distinguish hard problems from easy ones by simply counting variables, clauses, and literals. The great versatility with which clauses can capture logical relationships means that different sets of clauses can lead to wildly different phenomena. Some of this immense variety is indicated in Fig. 52, which depicts ten instructive “variable interaction graphs.” Each variable is represented by a ball, and two variables are linked when they appear together in at least one clause. (Some edges are darker than others; see exercise 506. For further examples of such 3D visualizations, presented also in color, see Carsten Sinz, *Journal of Automated Reasoning* **39** (2007), 219–243.)

A single SAT solver cannot be expected to excel on all of the many species of problems. Furthermore, nearly all of the 100 instances in Table 6 are well beyond the capabilities of the simple algorithms that we began with: Algorithms A, B, and D are unable to crack *any* of those test cases without needing more than fifty gigamems of computation, except for the simplest examples—L1, L2, L5, P3, P4, and X2. Algorithm L, the souped-up refinement of Algorithm D, also has a lot of difficulty with most of them. On the other hand, Algorithm C does remarkably well. It polishes off 79 of the given problems in fewer than *ten Gμ*.

Thus the test cases of Table 6 are tough, yet they're within reach. Almost all of them can be solved in say two minutes, at most, with methods known today.

Complete details can be found in the file `SATexamples.tgz` on the author's website, together with many related problems both large and small.

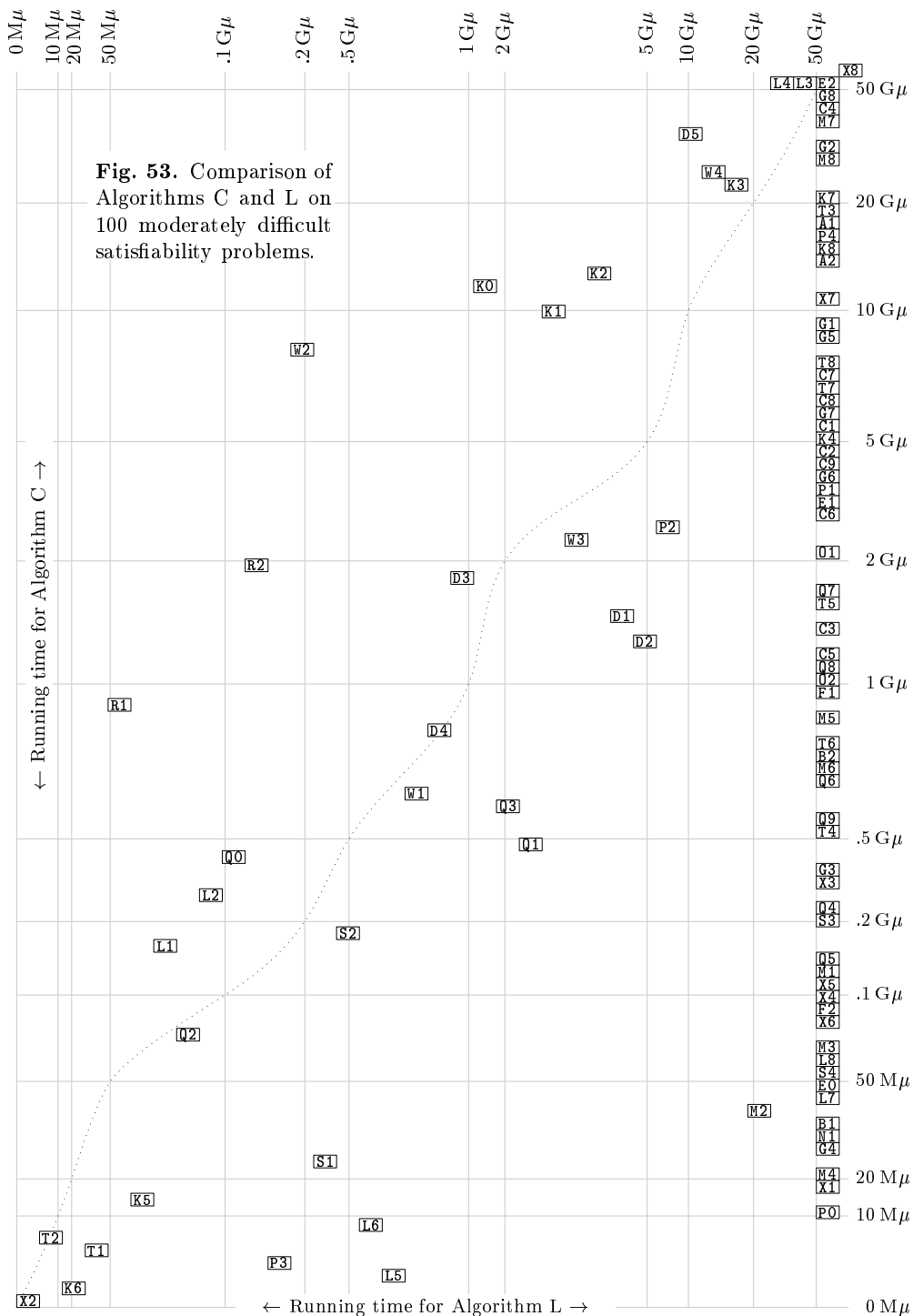
Exactly 50 of these 100 cases are satisfiable. So we're naturally led to wonder whether Algorithm W (“WalkSAT”) will handle such cases well. The answer is that Algorithm W sometimes succeeds brilliantly—especially on problems C7, C9, L5, L7, M3, M4, M6, P3, P4, Q0, Q1, R1, S1, where it typically outperforms all the other methods we've discussed. In particular it solved S1 in just  $1\text{ M}\mu$ , in the author's tests, compared to  $25\text{ M}\mu$  by the next best method, Algorithm C; it won by  $15\text{ M}\mu$  versus Algorithm C's  $83\text{ M}\mu$  on M3, by  $83\text{ M}\mu$  versus Algorithm L's  $104\text{ M}\mu$  on Q0, by  $95\text{ M}\mu$  versus Algorithm C's  $464\text{ M}\mu$  on Q1, and by a whopping  $104\text{ M}\mu$  versus Algorithm C's  $7036\text{ M}\mu$  on C7. That was a surprise. WalkSAT also was reasonably competitive on problem N1. But in all other cases it was nowhere near the method of choice. Therefore we'll consider only Algorithms L and C in the remainder of this discussion.\*

When does a lookahead algorithm like Algorithm L outperform a clause-learning algorithm like Algorithm C? Figure 53 shows how they compare to each other on our 100 test cases: Each problem is plotted with Algorithm C's running time on the vertical axis and Algorithm L's on the horizontal axis. Thus Algorithm L is the winner for problems that appear above the dotted line. (This dotted line is “wavy” because times aren't drawn to scale: The  $k$ th fastest running time is shown as  $k$  units from the left of the page or from the bottom.)

---

\* There actually are *two* variants of Algorithm L, because the alternative heuristics of exercise 143 must be used for looking ahead when clauses of length 4 or more are present. We could use exercise 143 even when given all-ternary clauses; but experience shows that we'd tend to lose a factor of 2 or more by doing so. Our references to Algorithm L therefore implicitly assume that exercise 143 is being applied only when necessary.

Sinz  
`SATexamples.tgz`  
 Knuth  
 website  
 Knuth  
 lookahead algorithm  
 clause-learning algorithm



All of these experiments were aborted after  $50G\mu$ , if necessary, since many of these problems could potentially take centuries before running to completion. Thus the test cases for which Algorithm L timed out appear at the right edge of Fig. 53, and the tough cases for Algorithm C appear at the top. Only E2 and X8 were too hard for both algorithms to handle within the specified cutoff time.

Algorithm L is deterministic: It uses no random variables. However, a slight change (see exercise 505) will randomize it, because the inputs can be shuffled as they are in Algorithm C; and we might as well assume that this change has been made. Then both Algorithms L and C have variable running times. They will find solutions or prove unsatisfiability more quickly on some runs than on others, as we've already seen for Algorithm C in Fig. 49.

To compensate for this variability, each of the runtimes reported in Fig. 53 is the *median* of nine independent trials. Figure 54 shows all  $9 \times 100$  of the empirical running times obtained with Algorithm C, sorted by their median values. We can see that many of the problems have near-constant behavior; indeed, the ratio max/min was less than 2 in 38 of the cases. But 10 cases turned out to be highly erratic in these experiments, with max/min  $> 100$ ; problem P4 was actually solved once after only 323 *kilomems*, while another run lasted 339 *gigamems*!

One might expect satisfiable problems, such as P4, to benefit more from lucky guesses than unsatisfiable problems do; and these experiments strongly support that hypothesis: Of the 21 problems with max/min  $> 30$ , all but P0 are satisfiable, and all 32 of the problems with max/min  $< 1.7$  are unsatisfiable. One might also expect the mean running time (the arithmetic average) to exceed the median running time, in problems like this — because bad luck can be significantly bad, though hopefully rare. Yet the mean is actually *smaller* than the median in 30 cases, about equally distributed between satisfiable and unsatisfiable.

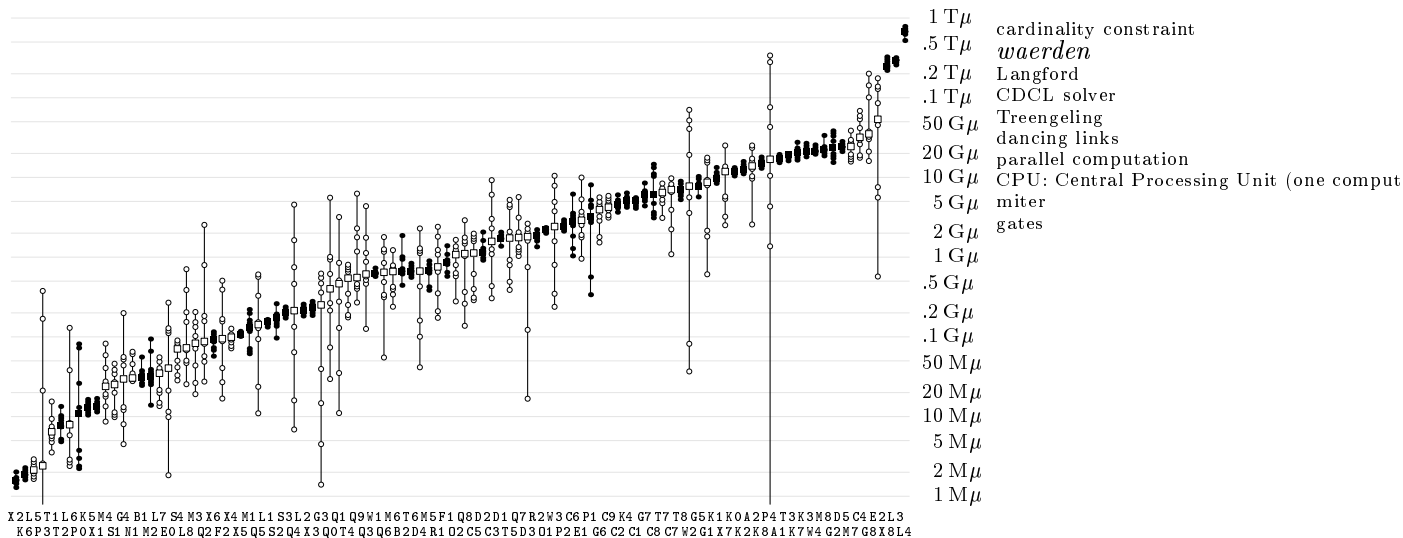
The median is a nice measure because it is meaningful even in the presence of occasional timeouts. It's also fair, because we are able to achieve the median time, or better, more often than not.

We should point out that input/output has been excluded from these time comparisons. Each satisfiability problem is supposed to appear within a computer's memory as a simple list of clauses, after which the counting of mems actually begins. We include the cost of initializing the data structures and solving the problem, but then we stop counting before actually outputting a solution.

Some of the test cases in Table 6 and Fig. 53 represent different encodings of the same problem. For example, problems K0–K8 all demonstrate that the  $8 \times 8$  queen graph can't be colored with 8 colors. Similarly, problems Q0–Q9 all show that 9 colors will suffice. We've already discussed these examples above when considering alternative encodings; and we noted that the best solutions, K6 and Q5, are obtained with an extended order encoding and with Algorithm C. Therefore the fact that Algorithm L beats Algorithm C on problems K0, K1, K2, and K3 is somewhat irrelevant; those problems won't occur in practice.

Problems L5 and L6 compare different ways to handle the at-most-one constraint. L6 is slightly better for Algorithm L, but Algorithm C prefers L5. Similarly, M1 and M2 compare different ways to deal with a more general

deterministic  
median  
mean running time  
average  
timeouts  
input/output  
encodings  
queen graph  
order encoding  
at-most-one



**Fig. 54.** Nine random running times of Algorithm C, sorted by their medians.  
(Unsatisfiable cases have solid dots or squares; satisfiable cases are hollow.)

cardinality constraint. Here M2 turns out to be better, although both are quite easy for Algorithm C and difficult for Algorithm L.

We’ve already noted that Algorithm L shines with respect to random problems such as R1 and R2, and it dominates all competitors even more when unsatisfiable random 3SAT problems get even bigger. Lookahead methods are also successful in *waerden* problems like W1–W4.

Unsatisfiable Langford problems such as L3 and L4 are definitely *bêtes noires* for Algorithm C, although not so bad for Algorithm L. Even the world’s fastest CDCL solver, “Treengeling,” was unable to refute the clauses of *langford* (17) in 2013 until it had learned 26.7 billion clauses; this process took more than a week, using a cluster of 24 computers working together. By contrast, the dancing links method of Section 7.2.2.1 was able to prove unsatisfiability after fewer than 7.2 Tμ of computation—that’s about 90 minutes on a single vintage-2013 CPU.

We’ve now discussed every case where Algorithm L trumps Algorithm C, *except* for D5; and D5 is actually somewhat scandalous! It’s an inherently simple problem that hardware designers call a “miter”: Imagine two *identical* circuits that compute some function  $f(x_1, \dots, x_n)$ , one with gates  $g_1, \dots, g_m$  and another with corresponding gates  $g'_1, \dots, g'_m$ , all represented as in (24). The problem is to find  $x_1 \dots x_n$  for which the final results  $g_m$  and  $g'_m$  aren’t equal. It’s obviously unsatisfiable. Furthermore, there’s an obvious way to refute it, by successively learning the clauses  $(\bar{g}_1 \vee g'_1)$ ,  $(\bar{g}'_1 \vee g_1)$ ,  $(\bar{g}_2 \vee g'_2)$ ,  $(\bar{g}'_2 \vee g_2)$ , etc. In theory, therefore, Algorithm C will almost surely finish in polynomial time (see exercise 386). But in practice, the algorithm won’t discover those clauses without quite a lot of flailing around, unless special-purpose techniques are introduced to help it discover isomorphic gates.

Thus Algorithm C does have an Achilles heel or two. On the other hand, it is the clear method of choice in the vast majority of our test cases, and we can expect it to be the major workhorse for most of the satisfiability problems that we encounter in daily work. Therefore it behooves us to understand its behavior in some detail, not just to look at its total cost as measured in mems.

empirical performance measurements++

**Table 7**

ALGORITHM C'S EMPIRICAL BEHAVIOR ON THE HUNDRED TEST CASES

| name | runtime        | bytes   | cells  | nodes | learned | of size                   | triv | disc | sub | flushes | sat? |
|------|----------------|---------|--------|-------|---------|---------------------------|------|------|-----|---------|------|
| X2   | 0+2 M $\mu$    | 57 K    | 9 K    | 2 K   | 1 K     | 32.0 $\rightarrow$ 12.0   | 50%  | 6%   | 1%  | 30      | U    |
| K6   | 0+2 M $\mu$    | 314 K   | 46 K   | 1 K   | 0 K     | 15.8 $\rightarrow$ 11.8   | 22%  | 4%   | 3%  | 6       | U    |
| L5   | 1+1 M $\mu$    | 1841 K  | 210 K  | 0 K   | 0 K     | 146.1 $\rightarrow$ 38.4  | 51%  | 23%  | 0%  | 0       | S    |
| P3   | 0+2 M $\mu$    | 96 K    | 19 K   | 2 K   | 1 K     | 18.4 $\rightarrow$ 12.6   | 4%   | 11%  | 1%  | 45      | S    |
| T1   | 0+6 M $\mu$    | 541 K   | 35 K   | 3 K   | 1 K     | 7.4 $\rightarrow$ 6.8     | 3%   | 2%   | 6%  | 9       | S    |
| T2   | 0+7 M $\mu$    | 574 K   | 37 K   | 4 K   | 1 K     | 7.2 $\rightarrow$ 6.8     | 1%   | 2%   | 4%  | 6       | U    |
| L6   | 0+8 M $\mu$    | 672 K   | 39 K   | 1 K   | 0 K     | 195.9 $\rightarrow$ 67.8  | 86%  | 0%   | 0%  | 0       | S    |
| P0   | 0+11 M $\mu$   | 376 K   | 81 K   | 8 K   | 4 K     | 17.8 $\rightarrow$ 14.7   | 3%   | 10%  | 10% | 28      | U    |
| K5   | 0+13 M $\mu$   | 294 K   | 55 K   | 3 K   | 2 K     | 18.6 $\rightarrow$ 12.4   | 33%  | 1%   | 1%  | 14      | U    |
| X1   | 0+13 M $\mu$   | 284 K   | 38 K   | 29 K  | 4 K     | 6.3 $\rightarrow$ 5.8     | 0%   | 3%   | 8%  | 53      | U    |
| M4   | 0+24 M $\mu$   | 308 K   | 47 K   | 6 K   | 4 K     | 20.5 $\rightarrow$ 16.3   | 14%  | 2%   | 1%  | 3       | S    |
| S1   | 0+25 M $\mu$   | 366 K   | 72 K   | 9 K   | 4 K     | 34.0 $\rightarrow$ 26.7   | 22%  | 4%   | 1%  | 14      | S    |
| G4   | 0+29 M $\mu$   | 759 K   | 76 K   | 3 K   | 2 K     | 37.1 $\rightarrow$ 24.2   | 26%  | 0%   | 0%  | 1       | S    |
| N1   | 16+14 M $\mu$  | 19644 K | 2314 K | 41 K  | 0 K     | 629.3 $\rightarrow$ 291.7 | 44%  | 6%   | 0%  | 15      | S    |
| B1   | 0+31 M $\mu$   | 251 K   | 55 K   | 10 K  | 7 K     | 13.5 $\rightarrow$ 11.3   | 3%   | 5%   | 4%  | 14      | U    |
| M2   | 0+32 M $\mu$   | 326 K   | 53 K   | 7 K   | 5 K     | 18.2 $\rightarrow$ 12.8   | 20%  | 1%   | 1%  | 6       | U    |
| L7   | 12+23 M $\mu$  | 14695 K | 1758 K | 2 K   | 1 K     | 411.2 $\rightarrow$ 107.6 | 66%  | 4%   | 0%  | 0       | S    |
| E0   | 0+40 M $\mu$   | 571 K   | 95 K   | 5 K   | 3 K     | 30.2 $\rightarrow$ 19.3   | 14%  | 11%  | 0%  | 6       | S    |
| S4   | 1+69 M $\mu$   | 3291 K  | 600 K  | 6 K   | 2 K     | 17.2 $\rightarrow$ 12.6   | 19%  | 1%   | 1%  | 8       | S    |
| L8   | 1+72 M $\mu$   | 3047 K  | 224 K  | 3 K   | 2 K     | 547.9 $\rightarrow$ 169.1 | 87%  | 0%   | 0%  | 0       | S    |
| M3   | 0+83 M $\mu$   | 493 K   | 84 K   | 13 K  | 9 K     | 28.4 $\rightarrow$ 19.2   | 31%  | 0%   | 1%  | 1       | S    |
| Q2   | 0+87 M $\mu$   | 885 K   | 190 K  | 11 K  | 8 K     | 61.7 $\rightarrow$ 45.8   | 36%  | 0%   | 0%  | 11      | S    |
| X6   | 0+93 M $\mu$   | 775 K   | 122 K  | 86 K  | 17 K    | 13.5 $\rightarrow$ 11.4   | 0%   | 3%   | 3%  | 32      | U    |
| F2   | 0+95 M $\mu$   | 714 K   | 118 K  | 42 K  | 22 K    | 14.3 $\rightarrow$ 13.1   | 0%   | 2%   | 4%  | 5       | S    |
| X4   | 1+98 M $\mu$   | 3560 K  | 158 K  | 24 K  | 3 K     | 16.2 $\rightarrow$ 11.4   | 9%   | 2%   | 3%  | 623     | S    |
| X5   | 1+106 M $\mu$  | 3747 K  | 166 K  | 23 K  | 3 K     | 16.5 $\rightarrow$ 11.0   | 11%  | 3%   | 3%  | 726     | U    |
| M1   | 0+131 M $\mu$  | 483 K   | 84 K   | 16 K  | 12 K    | 23.2 $\rightarrow$ 13.4   | 33%  | 1%   | 0%  | 1       | U    |
| Q5   | 0+143 M $\mu$  | 708 K   | 157 K  | 13 K  | 11 K    | 28.8 $\rightarrow$ 23.6   | 21%  | 2%   | 2%  | 6       | S    |
| L1   | 0+157 M $\mu$  | 597 K   | 139 K  | 21 K  | 18 K    | 36.7 $\rightarrow$ 19.0   | 60%  | 3%   | 0%  | 30      | U    |
| S2   | 0+176 M $\mu$  | 722 K   | 161 K  | 29 K  | 17 K    | 37.5 $\rightarrow$ 27.5   | 33%  | 3%   | 1%  | 8       | U    |
| S3   | 1+201 M $\mu$  | 2624 K  | 471 K  | 12 K  | 6 K     | 14.5 $\rightarrow$ 9.8    | 21%  | 1%   | 2%  | 1       | U    |
| Q4   | 0+213 M $\mu$  | 781 K   | 175 K  | 19 K  | 16 K    | 29.2 $\rightarrow$ 23.3   | 25%  | 3%   | 1%  | 6       | S    |
| L2   | 0+216 M $\mu$  | 588 K   | 136 K  | 23 K  | 20 K    | 36.2 $\rightarrow$ 17.4   | 75%  | 1%   | 0%  | 6       | U    |
| X3   | 0+235 M $\mu$  | 1000 K  | 191 K  | 61 K  | 25 K    | 37.7 $\rightarrow$ 19.3   | 34%  | 1%   | 2%  | 14      | U    |
| G3   | 0+251 M $\mu$  | 1035 K  | 145 K  | 12 K  | 9 K     | 57.9 $\rightarrow$ 28.1   | 42%  | 1%   | 0%  | 0       | S    |
| Q0   | 0+401 M $\mu$  | 1493 K  | 342 K  | 37 K  | 28 K    | 63.3 $\rightarrow$ 40.0   | 50%  | 0%   | 0%  | 14      | S    |
| Q1   | 0+464 M $\mu$  | 1516 K  | 343 K  | 41 K  | 33 K    | 63.0 $\rightarrow$ 41.0   | 45%  | 0%   | 0%  | 14      | S    |
| T4   | 0+546 M $\mu$  | 2716 K  | 544 K  | 202 K | 18 K    | 218.3 $\rightarrow$ 61.5  | 83%  | 1%   | 0%  | 3018    | S    |
| Q9   | 0+555 M $\mu$  | 1409 K  | 343 K  | 152 K | 71 K    | 26.7 $\rightarrow$ 20.6   | 3%   | 5%   | 2%  | 99      | S    |
| Q3   | 0+613 M $\mu$  | 1883 K  | 448 K  | 27 K  | 22 K    | 60.1 $\rightarrow$ 40.3   | 41%  | 1%   | 1%  | 7       | S    |
| W1   | 0+626 M $\mu$  | 848 K   | 208 K  | 71 K  | 63 K    | 20.8 $\rightarrow$ 13.4   | 5%   | 14%  | 1%  | 28      | U    |
| Q6   | 0+646 M $\mu$  | 1211 K  | 266 K  | 40 K  | 35 K    | 30.4 $\rightarrow$ 23.2   | 30%  | 1%   | 1%  | 2       | S    |
| M6   | 0+660 M $\mu$  | 1378 K  | 266 K  | 80 K  | 52 K    | 34.0 $\rightarrow$ 22.2   | 33%  | 1%   | 1%  | 59      | S    |
| B2   | 0+668 M $\mu$  | 906 K   | 216 K  | 96 K  | 75 K    | 17.1 $\rightarrow$ 13.2   | 4%   | 5%   | 2%  | 16      | U    |
| T6   | 1+668 M $\mu$  | 2355 K  | 291 K  | 34 K  | 25 K    | 41.4 $\rightarrow$ 19.1   | 57%  | 0%   | 1%  | 11      | U    |
| D4   | 0+669 M $\mu$  | 1009 K  | 186 K  | 35 K  | 28 K    | 55.7 $\rightarrow$ 15.9   | 70%  | 0%   | 0%  | 2       | S    |
| M5   | 0+677 M $\mu$  | 1183 K  | 219 K  | 73 K  | 48 K    | 32.6 $\rightarrow$ 20.2   | 37%  | 1%   | 1%  | 139     | U    |
| R1   | 0+756 M $\mu$  | 913 K   | 220 K  | 87 K  | 74 K    | 17.3 $\rightarrow$ 12.4   | 3%   | 8%   | 0%  | 9       | S    |
| F1   | 0+859 M $\mu$  | 1485 K  | 311 K  | 218 K | 135 K   | 17.6 $\rightarrow$ 15.1   | 1%   | 3%   | 3%  | 6       | U    |
| O2   | 7+1069 M $\mu$ | 18951 K | 3144 K | 3 K   | 2 K     | 17.0 $\rightarrow$ 9.5    | 35%  | 0%   | 0%  | 1       | S    |
| Q8   | 0+1107 M $\mu$ | 1786 K  | 437 K  | 184 K | 109 K   | 29.4 $\rightarrow$ 20.2   | 6%   | 6%   | 1%  | 109     | S    |

| C5   | 0+1127 M $\mu$   | 1987 K  | 419 K  | 159 K   | 104 K   | 24.4 $\rightarrow$ 16.5  | 12%  | 2%   | 1%  | 776     | s    |
|------|------------------|---------|--------|---------|---------|--------------------------|------|------|-----|---------|------|
| D2   | 0+1159 M $\mu$   | 962 K   | 177 K  | 54 K    | 45 K    | 51.8 $\rightarrow$ 11.5  | 73%  | 0%   | 0%  | 2       | U    |
| C3   | 0+1578 M $\mu$   | 2375 K  | 571 K  | 190 K   | 96 K    | 49.7 $\rightarrow$ 23.4  | 39%  | 3%   | 2%  | 11      | s    |
| D1   | 0+1707 M $\mu$   | 1172 K  | 230 K  | 76 K    | 62 K    | 45.1 $\rightarrow$ 11.6  | 73%  | 0%   | 0%  | 2       | U    |
| T5   | 1+1735 M $\mu$   | 3658 K  | 617 K  | 80 K    | 59 K    | 72.5 $\rightarrow$ 40.9  | 50%  | 0%   | 0%  | 43      | s    |
| Q7   | 0+1761 M $\mu$   | 2055 K  | 419 K  | 515 K   | 118 K   | 33.9 $\rightarrow$ 20.3  | 9%   | 7%   | 0%  | 12      | s    |
| D3   | 0+1807 M $\mu$   | 1283 K  | 254 K  | 77 K    | 64 K    | 57.3 $\rightarrow$ 14.0  | 80%  | 0%   | 0%  | 1       | s    |
| R2   | 0+1886 M $\mu$   | 1220 K  | 296 K  | 173 K   | 149 K   | 17.0 $\rightarrow$ 11.8  | 3%   | 9%   | 0%  | 14      | U    |
| O1   | 7+2212 M $\mu$   | 18928 K | 3140 K | 5 K     | 3 K     | 17.3 $\rightarrow$ 8.9   | 39%  | 0%   | 0%  | 4       | U    |
| W3   | 0+2422 M $\mu$   | 1819 K  | 448 K  | 191 K   | 174 K   | 19.3 $\rightarrow$ 15.5  | 2%   | 12%  | 1%  | 18      | s    |
| P2   | 0+2435 M $\mu$   | 2039 K  | 504 K  | 378 K   | 301 K   | 20.9 $\rightarrow$ 13.7  | 3%   | 11%  | 1%  | 45      | U    |
| C6   | 0+2792 M $\mu$   | 2551 K  | 560 K  | 305 K   | 217 K   | 27.0 $\rightarrow$ 17.0  | 20%  | 2%   | 1%  | 492     | U    |
| E1   | 0+2902 M $\mu$   | 2116 K  | 453 K  | 180 K   | 144 K   | 38.0 $\rightarrow$ 20.5  | 21%  | 18%  | 0%  | 2       | s    |
| P1   | 0+3280 M $\mu$   | 2726 K  | 674 K  | 819 K   | 549 K   | 18.2 $\rightarrow$ 14.4  | 0%   | 9%   | 3%  | 45      | U    |
| G6   | 1+3941 M $\mu$   | 3523 K  | 647 K  | 380 K   | 253 K   | 31.0 $\rightarrow$ 17.8  | 31%  | 0%   | 0%  | 0       | s    |
| C9   | 13+4220 M $\mu$  | 35486 K | 4923 K | 116 K   | 32 K    | 11.8 $\rightarrow$ 9.9   | 5%   | 1%   | 1%  | 4986    | s    |
| C2   | 0+4625 M $\mu$   | 2942 K  | 712 K  | 442 K   | 255 K   | 46.1 $\rightarrow$ 18.8  | 42%  | 4%   | 1%  | 15      | U    |
| K4   | 0+5122 M $\mu$   | 1858 K  | 446 K  | 267 K   | 241 K   | 19.6 $\rightarrow$ 13.7  | 19%  | 2%   | 1%  | 5       | U    |
| C1   | 0+5178 M $\mu$   | 2532 K  | 613 K  | 510 K   | 311 K   | 48.9 $\rightarrow$ 17.0  | 48%  | 6%   | 1%  | 20      | U    |
| G7   | 1+6070 M $\mu$   | 4227 K  | 771 K  | 546 K   | 369 K   | 32.5 $\rightarrow$ 17.6  | 35%  | 0%   | 0%  | 0       | U    |
| C8   | 13+6081 M $\mu$  | 35014 K | 4823 K | 151 K   | 58 K    | 15.3 $\rightarrow$ 10.7  | 15%  | 1%   | 1%  | 8067    | U    |
| T7   | 1+6467 M $\mu$   | 5428 K  | 544 K  | 333 K   | 108 K   | 26.8 $\rightarrow$ 15.3  | 32%  | 1%   | 1%  | 14565   | s    |
| C7   | 8+7029 M $\mu$   | 20971 K | 3174 K | 908 K   | 32 K    | 9.5 $\rightarrow$ 8.4    | 0%   | 3%   | 0%  | 4965    | s    |
| T8   | 1+7046 M $\mu$   | 5322 K  | 517 K  | 356 K   | 117 K   | 26.9 $\rightarrow$ 15.0  | 33%  | 0%   | 1%  | 15026   | U    |
| W2   | 0+7785 M $\mu$   | 3561 K  | 884 K  | 501 K   | 432 K   | 34.7 $\rightarrow$ 21.3  | 13%  | 17%  | 1%  | 28      | s    |
| G5   | 1+7799 M $\mu$   | 4312 K  | 844 K  | 642 K   | 446 K   | 33.4 $\rightarrow$ 17.4  | 39%  | 0%   | 0%  | 0       | U    |
| G1   | 0+8681 M $\mu$   | 5052 K  | 1221 K | 631 K   | 350 K   | 61.1 $\rightarrow$ 34.1  | 38%  | 1%   | 2%  | 55      | s    |
| K1   | 0+9813 M $\mu$   | 2864 K  | 685 K  | 405 K   | 360 K   | 36.2 $\rightarrow$ 18.4  | 53%  | 2%   | 0%  | 13      | U    |
| X7   | 1+11857 M $\mu$  | 6235 K  | 697 K  | 1955 K  | 224 K   | 40.6 $\rightarrow$ 23.7  | 35%  | 0%   | 1%  | 31174   | s    |
| K0   | 0+11997 M $\mu$  | 3034 K  | 731 K  | 493 K   | 421 K   | 35.6 $\rightarrow$ 19.4  | 45%  | 2%   | 0%  | 14      | U    |
| K2   | 0+12601 M $\mu$  | 3028 K  | 729 K  | 500 K   | 427 K   | 34.8 $\rightarrow$ 18.0  | 46%  | 2%   | 0%  | 12      | U    |
| A2   | 0+13947 M $\mu$  | 3766 K  | 843 K  | 645 K   | 585 K   | 34.4 $\rightarrow$ 15.9  | 32%  | 1%   | 0%  | 0       | s    |
| K8   | 0+15033 M $\mu$  | 2748 K  | 680 K  | 821 K   | 699 K   | 21.2 $\rightarrow$ 13.1  | 8%   | 15%  | 1%  | 93      | U    |
| P4   | 0+16907 M $\mu$  | 6936 K  | 1721 K | 1676 K  | 1314 K  | 36.5 $\rightarrow$ 24.0  | 5%   | 11%  | 1%  | 33      | s    |
| A1   | 0+17073 M $\mu$  | 3647 K  | 815 K  | 763 K   | 701 K   | 30.7 $\rightarrow$ 14.7  | 29%  | 2%   | 0%  | 0       | U    |
| T3   | 0+19266 M $\mu$  | 10034 K | 2373 K | 2663 K  | 323 K   | 291.8 $\rightarrow$ 72.9 | 86%  | 1%   | 0%  | 34265   | U    |
| K7   | 0+20577 M $\mu$  | 3168 K  | 721 K  | 1286 K  | 828 K   | 23.3 $\rightarrow$ 13.5  | 9%   | 15%  | 0%  | 9       | U    |
| K3   | 0+20990 M $\mu$  | 3593 K  | 878 K  | 453 K   | 407 K   | 36.7 $\rightarrow$ 19.0  | 55%  | 2%   | 0%  | 6       | U    |
| W4   | 0+21295 M $\mu$  | 3362 K  | 834 K  | 977 K   | 899 K   | 19.0 $\rightarrow$ 14.1  | 4%   | 15%  | 0%  | 21      | U    |
| M8   | 0+22281 M $\mu$  | 4105 K  | 994 K  | 992 K   | 785 K   | 37.3 $\rightarrow$ 20.5  | 43%  | 1%   | 1%  | 6       | U    |
| G2   | 0+23424 M $\mu$  | 6910 K  | 1685 K | 1198 K  | 701 K   | 68.8 $\rightarrow$ 34.3  | 47%  | 1%   | 1%  | 120     | U    |
| D5   | 0+24141 M $\mu$  | 3232 K  | 779 K  | 787 K   | 654 K   | 63.5 $\rightarrow$ 13.4  | 78%  | 0%   | 0%  | 2       | U    |
| M7   | 0+24435 M $\mu$  | 4438 K  | 1077 K | 1047 K  | 819 K   | 40.6 $\rightarrow$ 23.3  | 42%  | 1%   | 1%  | 6       | s    |
| C4   | 1+31898 M $\mu$  | 8541 K  | 2108 K | 1883 K  | 1148 K  | 60.6 $\rightarrow$ 25.7  | 42%  | 4%   | 1%  | 12      | s    |
| G8   | 7+35174 M $\mu$  | 24854 K | 2992 K | 4350 K  | 1101 K  | 48.0 $\rightarrow$ 34.7  | 9%   | 0%   | 0%  | 1523    | s    |
| E2   | 0+53739 M $\mu$  | 5454 K  | 1258 K | 2020 K  | 1658 K  | 41.5 $\rightarrow$ 20.8  | 25%  | 21%  | 0%  | 3       | s    |
| X8   | 2+248789 M $\mu$ | 12814 K | 2311 K | 17005 K | 3145 K  | 56.4 $\rightarrow$ 22.5  | 63%  | 0%   | 0%  | 330557  | U    |
| L3   | 0+295571 M $\mu$ | 19653 K | 4894 K | 7402 K  | 6886 K  | 70.7 $\rightarrow$ 31.0  | 63%  | 8%   | 0%  | 30      | U    |
| L4   | 0+677815 M $\mu$ | 22733 K | 5664 K | 8545 K  | 7931 K  | 78.6 $\rightarrow$ 35.4  | 86%  | 0%   | 0%  | 5       | U    |
| name | runtime          | bytes   | cells  | nodes   | learned | of size                  | triv | disc | sub | flushes | sat? |

Table 7 summarizes the salient statistics, again listing all cases in order of their median running time (exclusive of input and output). Each running time is actually broken into two parts, ‘ $x+y$ ’, where  $x$  is the time to initialize the data structures in step C1 and  $y$  is the time for the other steps, both rounded to megamems. For example, the exact median processing time for case L5 was 1,484,489 $\mu$  to initialize, then 655,728 $\mu$  to find a solution; this is shown as ‘1+1 M $\mu$ ’ in the third line of the table. The time for initialization is usually negligible except when there are many clauses, as in problem N1.



The median run of problem L5 also allocated 1,841,372 bytes of memory for data; this total includes the space needed for 210,361 cells in the MEM array, at 4 bytes per cell, together with other arrays such as VAL, OVAL, HEAP, etc. The implementation considered here keeps unlearned binary clauses in a separate BIMP table, as explained in the answer to exercise 267.

This run of L5 found a solution after implicitly traversing a search tree with 138 “nodes.” The number of nodes, or “decisions,” is the number of times step C6 of the algorithm goes to step C3. It is shown as ‘OK’ in Table 7, because the node counts, byte counts, and cell counts are rounded to the nearest thousand.

The number of nodes always exceeds or equals the number of learned clauses, which is the number of conflicts detected at levels  $d > 0$ . (See step C7.) In the case of problem L5, only 84 clauses were learned; so again the table reports ‘OK’. These 84 clauses had average length  $r + 1 = 146.1$ ; then the simplification process of exercise 257 reduced this average to just 38.4. Nevertheless, the resulting simplified clauses were still sufficiently long that the “trivial” clauses discussed in exercise 269 were sometimes used instead; this substitution happened 43 times (51%). Furthermore 19 of the learned clauses (23%) were immediately discarded, using the method of exercise 271. These percentages show up in the ‘triv’ and ‘disc’ columns of the table.

Sometimes, as in problems D1–D5, a large majority of the learned clauses were replaced by trivial ones; on the other hand, 27 of the 100 cases turned out to be less than 10% trivial in this sense. Table 7 also shows that the discard rate was 5% or more in 26 cases. The ‘sub’ column refers to learned clauses that were “subsumed on the fly” by the technique of exercise 270; this optimization is less common, yet it occurs often enough to be worthwhile.

The great variety in our examples is reflected in the variety of behaviors exhibited in Table 7, although several interesting trends can also be perceived. For example, the number of nodes is naturally correlated with the number of learned clauses, and both statistics tend to grow as the total running time increases. But there are significant exceptions: Two outliers, O1 and O2, have a remarkably high ratio of mems per learned clause, because of their voluminous data.

The penultimate column of Table 7 counts how often Algorithm C decided to restart itself after flushing unproductive literals from its current trail. This quantity does not simply represent the number of times step C5 discovers that  $M \geq M_f$ ; it depends also on the current agility level (see (127)) and on the parameter  $\psi$  in Table 4. Some problems, like A1 and A2, had such high agility that they were solved satisfactorily with no restarts whatsoever; but another one, T4, finished in about 500 megamems after restarting more than 3000 times.

The number of “purges” (recycling phases) is not shown, but it can be estimated from the number of learned clauses (see exercise 508). An aggressive purging policy has kept the total number of memory cells comfortably small.

**Tuning up the parameters.** Table 7 shows that the hardest problem of all for Algorithm C in these experiments, L4, found itself substituting trivial clauses 86% of the time but making only 5 restarts. That test case would probably have

MEM  
 binary clauses  
 BIMP  
 search tree  
 decision tree, see search tree  
 nodes  
 decisions  
 learned clauses  
 conflicts  
 levels  
 trivial  
 subsumed on the fly  
 on-the-fly subsumption  
 restart  
 flushing  
 trail  
 agility level  
 purges  
 recycling phases  
 tuning of parameters–  
 parameters, tuning of–

been solved much more quickly if the algorithm’s parameters had been specially adjusted for instances of the Langford problem.

Algorithm C, as implemented in the experiments above, has ten major parameters that can be modified by the user on each run:

- $\alpha$ , tradeoff between  $p$  and  $q$  in clause RANGE scores (see Eq. (123));
- $\rho$ , damping factor in variable ACT scores (see after (118));
- $\varrho$ , damping factor in clause ACT scores (see Eq. (125));
- $\Delta_p$ , initial value of the purging threshold  $M_p$  (see after (125));
- $\delta_p$ , amount of gradual increase in  $M_p$  (see after (125));
- $\tau$ , threshold used to prefer trivial clauses (see answer to exercise 269);
- $w$ , full “warmup” runs done after a restart (see answer to exercise 287);
- $p$ , probability of choosing a decision variable at random (see exercise 266);
- $P$ , probability that OVAL( $k$ ) is initially even;
- $\psi$ , agility threshold for flushing (see Table 4).

The values for these parameters initially came from seat-of-the-pants guesses

$$\alpha = 0.2, \quad \rho = 0.95, \quad \varrho = 0.999, \quad \Delta_p = 20000, \quad \delta_p = 500, \\ \tau = 1, \quad w = 0, \quad p = 0.02, \quad P = 0, \quad \psi = 0.166667; \quad (193)$$

and these defaults gave reasonably good results, so they were used happily for many months (although there was no good reason to believe that they couldn’t be improved). Then finally, after the author had assembled the set of 100 test cases in Table 6, it was time to decide whether to recommend the default values (193) or to come up with a better set of numbers.

Parameter optimization for general broad-spectrum use is a daunting task, not only because of significant differences between species of SAT instances but also because of the variability due to random choices when solving any specific instance. It’s hard to know whether a change of parameter will be beneficial or harmful, when running times are so highly erratic. Ouch—Fig. 54 illustrates dramatic variations even when all ten parameters are held fixed, and only the seed for random numbers is changed! Furthermore the ten parameters are not at all independent: An increase in  $\rho$ , say, might be a good thing, but only if the other nine parameters are also modified appropriately. How then could *any* set of defaults be recommended, without an enormous expense of time and money?

Fortunately there’s a way out of this dilemma, thanks to advances in the theory of learning. F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle have developed a tool called ParamILS intended specifically for making such tuneups [*J. Artificial Intelligence Research* **36** (2009), 267–306]; the ‘ILS’ in this name stands for “iterated local search.” The basic idea is to start with a representative training set of not-too-hard problems, and to carry out random walks in the 10-dimensional parameter space using sophisticated refinements of WalkSAT-like principles. The best parameters discovered during this training session are then evaluated on more difficult problems outside the training set.

Langford problem  
 RANGE scores  
 tradeoffs  
 damping factors  
 ACT scores  
 activity scores  
 purging threshold  
 trivial clauses  
 warmup runs  
 restarts  
 random decision variables  
 initial guess for literals  
 OVAL  
 agility threshold  
 defaults  
 author  
 Hutter  
 Hoos  
 Leyton-Brown  
 Stützle  
 ParamILS  
 ILS  
 iterated local search  
 training set  
 random walks  
 WalkSAT

In March 2015, Holger Hoos helped the author to tune Algorithm C using ParamILS. The resulting parameters then yielded Fig. 54, and Table 7, and many other runtime values discussed above and below. Our training set consisted of 17 problems that usually cost less than  $200\text{M}\mu$  with the original parameters (193), namely  $\{\text{K5, K6, M2, M4, N1, S1, S4, X4, X6}\}$  together with stripped-down versions of  $\{\text{A1, C2, C3, D1, D2, D3, D4, K0}\}$ . For example, instead of the vector  $x_1 \dots x_{100}$  required by problem A1, we looked only for a shorter vector  $x = x_1 \dots x_{62}$ , now with  $\nu x = 20$ ; instead of D1 and D2 we sought 13-bit factors of 31415926; instead of K0 we tried to 9-color the SGB graph *jean*.

Ten independent training runs with ParamILS gave ten potential parameter settings  $(\alpha_i, \rho_i, \dots, \psi_i)$ . We evaluated them on our original 17 benchmarks, together with 25 others that were a bit more difficult:  $\{\text{F1, F2, S2, S3, T4, X5}\}$ , plus less-stripped-down variants of  $\{\text{A1, A2, A2, C7, C7, D3, D4, F1, F2, G1, G1, G2, G2, G8, K0, O1, O2, Q0, Q2}\}$ . For each of the ten shortlisted parameter settings, we ran each of these 17 + 25 problems with each of the random seeds  $\{1, 2, \dots, 25\}$ . Finally, hurray, we had a winner: The parameters  $(\alpha_i, \rho_i, \dots, \psi_i)$  with minimum total running time in this experiment were

$$\alpha = 0.4, \quad \rho = 0.9, \quad \varrho = 0.9995, \quad \Delta_p = 1000, \quad \delta_p = 500, \\ \tau = 10, \quad w = 0, \quad p = 0.02, \quad P = 0.5, \quad \psi = 0.05. \quad (194)$$

And these are now the recommended defaults for general-purpose use.

How much have we thereby gained? Figure 55 compares the running times of our 100 examples, before and after tuning. It shows that the vast majority—77 of them—now run faster; these are the cases to the right of the dotted line from  $(1\text{M}\mu, 1\text{M}\mu)$  to  $(1\text{T}\mu, 1\text{T}\mu)$ . Half of the cases experience a speedup exceeding 1.455; 27 of them now run more than twice as fast as they previously did.

Of course every rule has exceptions. The behavior of case P4 has gotten spectacularly worse, almost three orders of magnitude slower! Indeed, we saw earlier in Fig. 54 that this case has an amazingly unstable running time; further peculiarities of P4 are discussed in exercise 511.

Our other major SAT solver, Algorithm L, also has parameters, notably

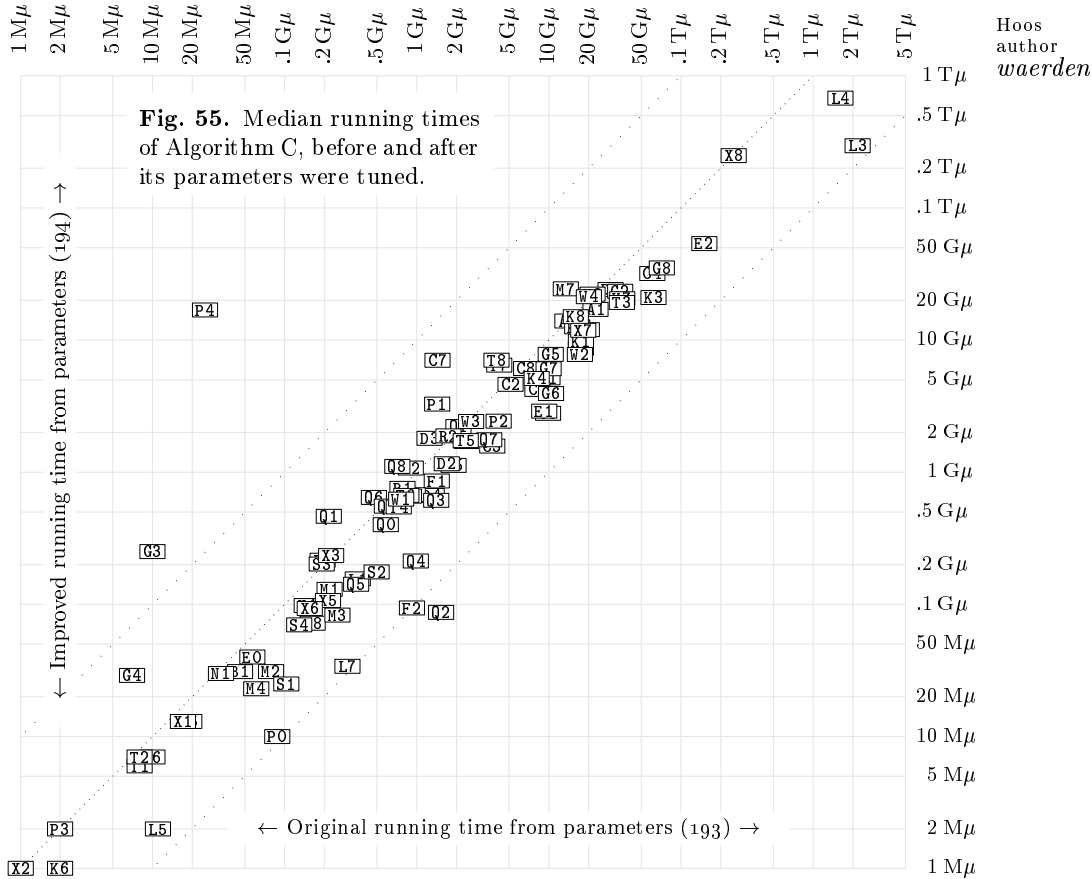
- $\alpha$ , magic tradeoff coefficient in heuristic scores (see Eq. (64));
- $\beta$ , damping factor for double-look triggering (see step Y1);
- $\gamma$ , clause weight per literal in heuristic scores (see exercise 175);
- $\varepsilon$ , offset in heuristic scores (see answer to exercise 146);
- $\Theta$ , maximum heuristic score threshold (see answer to exercise 145);
- $Y$ , maximum depth of double-lookahead (see step Y1).

ParamILS suggests the following default values, which have been used in Fig. 53:

$$\alpha = 3.5, \quad \beta = 0.9998, \quad \gamma = 0.2, \quad \varepsilon = 0.001, \quad \Theta = 20.0, \quad Y = 1. \quad (195)$$

Returning to Fig. 55, notice that the change from (193) to (194) has substantially hindered cases G3 and G4, which are examples of test pattern generation. Evidently such clauses have special characteristics that make them prefer special

Hoos  
author  
training set  
SGB  
*book* graphs  
forty-two...  
defaults  
tradeoffs  
damping factors  
heuristic scores  
adaptive control  
trigger  
double-looking ahead  
ATPG  
test pattern generation

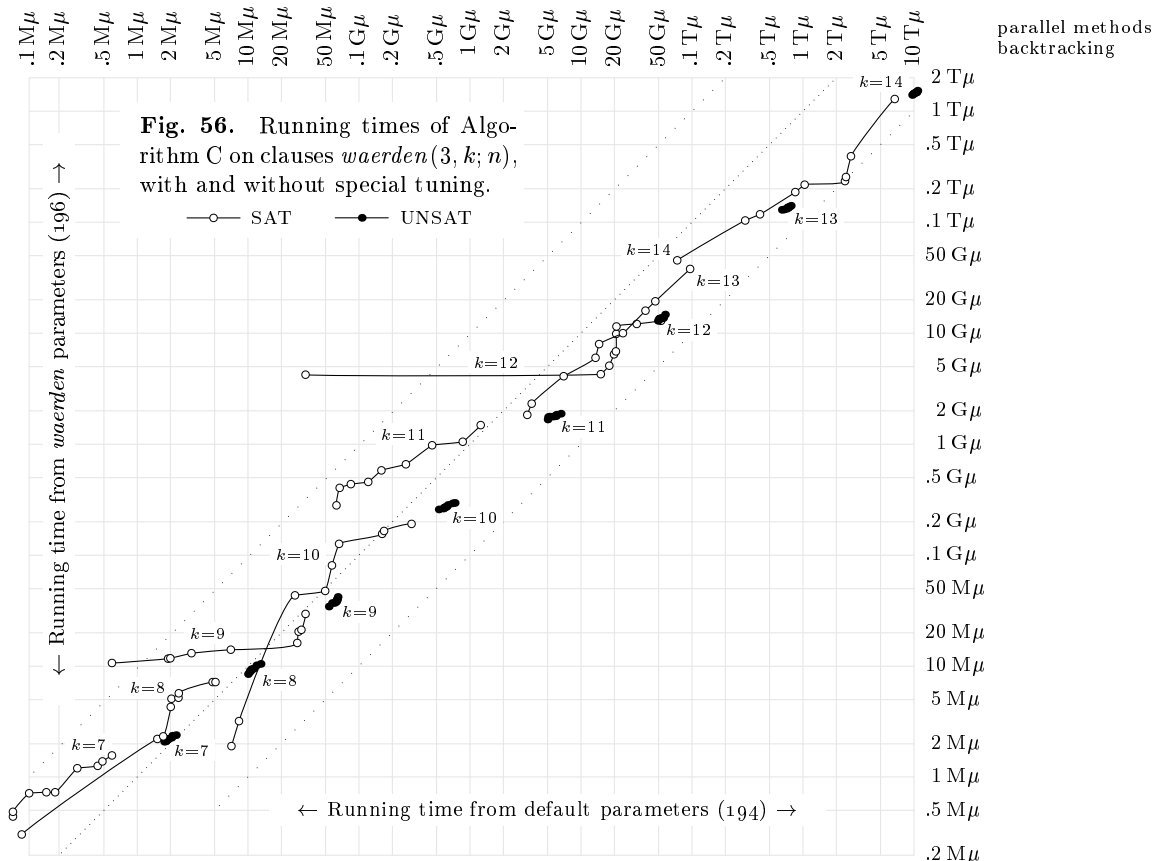


settings of the parameters. Our main reason for introducing parameters in the first place was, of course, to allow tweaking for different families of clauses.

Instead of finding values of  $(\alpha, \rho, \dots, \psi)$  that give good results in a broad spectrum of applications, we can clearly use a system like ParamILS to find values that are specifically tailored to a particular class of problems. In fact, this task is easier. For example, Hoos and the author asked for settings of the ten parameters that will tend to make Algorithm C do its best on problems of the form *waarden*(3,  $k$ ;  $n$ ). A pair of ParamILS runs, based solely on the easy training cases *waarden*(3, 9; 77) and *waarden*(3, 10; 95), suggested the parameters

$$\alpha = 0.5, \quad \rho = 0.9995, \quad \varrho = 0.99, \quad \Delta_p = 100, \quad \delta_p = 10, \\ \tau = 10, \quad w = 8, \quad p = 0.01, \quad P = 0.5, \quad \psi = 0.15, \quad (196)$$

and this set indeed works very well. Figure 56 shows typical details, with  $7 \leq k \leq 14$  and with nine independent sample runs for every choice of  $k$  and  $n$ . Each unsatisfiable instance has  $n = W(3, k)$ , as given in the table following (10) above; each satisfiable instance has  $n = W(3, k) - 1$ . The fastest run using default



parameters (194) has been paired in Fig. 56 with the fastest run using *waerden*-tuned parameters (196); similarly, the second-fastest, . . . , second-slowest, and slowest runs have also been paired. Notice that satisfiable instances tend to take an unpredictable amount of time, as in Fig. 54. In spite of the fact that the new parameters (196) were found by a careful study of just two simple instances, they clearly yield substantial savings when applied to much, much harder problems of a similar nature. (See exercise 512 for another instructive example.)

**Exploiting parallelism.** Our focus in the present book is almost entirely on sequential algorithms, but we should be aware that the really tough instances of SAT are best solved by parallel methods.

Problems that are amenable to backtracking can readily be decomposed into subproblems that partition the space of solutions. For example, if we have 16 processors available, we can start them off on independent SAT instances in which variables  $x_1x_2x_3x_4$  have been forced to equal 0000, 0001, . . . , 1111.

A naïve decomposition of that kind is rarely the best strategy, however. Perhaps only one of those sixteen cases is really challenging. Perhaps some of

the processors are slower than others. Perhaps several processors will learn new clauses that the other processors ought to know. Furthermore, the splitting into subproblems need not occur only at the root of the search tree. Careful load-balancing and sharing of information will do much better. These challenges were addressed by a pioneering system called PSATO [H. Zhang, M. P. Bonacina, and J. Hsiang, *Journal of Symbolic Computation* **21** (1996), 543–560].

A much simpler approach should also be mentioned: We can start up many different solvers, or many copies of the same solver, with different sources of random numbers. As soon as one has finished, we can then terminate the others.

The best parallelized SAT solvers currently available are based on the “cube and conquer” paradigm, which combines conflict-driven clause learning with lookahead techniques that choose branch variables for partitioning; see M. J. H. Heule, O. Kullmann, S. Wieringa, and A. Biere, *LNCS* **7261** (2012), 50–65. In particular, this approach is excellent for the *waerden* problems.

*Today has proved to be an epoch in my Logical work.  
... I think of calling it the ‘Genealogical Method.’*  
— CHARLES L. DODGSON, *Diary* (16 July 1894)

*The method of showing a statement to be tautologous  
consists merely of constructing a table under it in the usual way  
and observing that the column under the main connective  
is composed entirely of ‘T’s.*  
— W. V. O. QUINE, *Mathematical Logic* (1940)

**A brief history.** The classic syllogism “All men are mortal; Socrates is a man; hence Socrates is mortal” shows that the notion of *resolution* is quite ancient:

$$\neg \text{Man} \vee \text{Mortal}; \quad \neg \text{Socrates} \vee \text{Man}; \quad \therefore \neg \text{Socrates} \vee \text{Mortal}.$$

Of course, algebraic demonstrations that  $(\neg x \vee y) \wedge (\neg z \vee x)$  implies  $(\neg z \vee y)$ , when  $x$ ,  $y$ , and  $z$  are arbitrary Boolean expressions, had to wait until Boole and his 19th-century followers brought mathematics to bear on the subject. The most notable contributor, resolutionwise, was perhaps C. L. Dodgson, who spent the last years of his life working out theories of inference by which complex chains of reasoning could be analyzed by hand. He published *Symbolic Logic*, Part I, in 1896, addressing it to children and to the young-in-heart by using his famous pen name Lewis Carroll. Section VII.II.§3 of that book explains and illustrates how to eliminate variables by resolution, which he called the Method of Underscoring.

When Dodgson died unexpectedly at the beginning of 1898, his nearly complete manuscript for *Symbolic Logic*, Part II, vanished until W. W. Bartley III was able to resurrect it in 1977. Part II was found to contain surprisingly novel ideas—especially its Method of Trees, which would have completely changed the history of mechanical theorem proving if it had come to light earlier. In this method, which Carroll documented at length in a remarkably clear and entertaining way, he constructed search trees essentially like Fig. 39, then converted them into proofs by resolution. Instead of backtracking as in Algorithm D,

PSATO  
Zhang  
Bonacina  
Hsiang  
random numbers  
cube and conquer  
conflict-driven clause learning  
lookahead  
Heule  
Kullmann  
Wieringa  
Biere  
*waerden*  
DODGSON  
tautologous  
truth table  
QUINE  
syllogism  
Socrates  
resolution  
Boole  
Dodgson  
Carroll  
eliminate variables  
Dodgson  
Bartley  
Method of Trees  
Carroll  
backtracking

which is a recursive depth-first method, he worked breadth-first: Starting at the root, he exploited unit clauses when possible, and branched on binary (or even ternary) clauses when necessary, successively filling out all unfinished branches level-by-level in hopes of being able to reuse computations.

Logicians of the 20th century took a different tack. They basically dealt with the satisfiability problem in its equivalent dual form as the tautology problem, namely to decide when a Boolean formula is always true. But they dismissed tautology-checking as a triviality, because it could always be solved in a finite number of steps by just looking at the truth table. Logicians were far more interested in problems that were provably *unsolvable* in finite time, such as the halting problem—the question of whether or not an algorithm terminates. Nobody was bothered by the fact that an  $n$ -variable function has a truth table of length  $2^n$ , which exceeds the size of the universe even when  $n$  is rather small.

Practical computations with disjunctive normal forms were pioneered by Archie Blake in 1937, who introduced the “consensus” of two implicants, which is dual to the resolvent of two clauses. Blake’s work was, however, soon forgotten; E. W. Samson, B. E. Mills, and (independently) W. V. O. Quine rediscovered the consensus operation in the 1950s, as discussed in exercise 7.1.1–31.

The next important step was taken by E. W. Samson and R. K. Mueller [Report AFCRC-TR-55-118 (Cambridge, Mass.: Air Force Cambridge Research Center, 1955), 16 pages], who presented an algorithm for the tautology problem that uses consensus to eliminate variables one by one. Their algorithm therefore was equivalent to SAT solving by successively eliminating variables via resolution. Samson and Mueller demonstrated their algorithm by applying it to the unsatisfiable clauses that we considered in (112) above.

Independently, Martin Davis and Hilary Putnam had begun to work on the satisfiability problem, motivated by the search for algorithms to deduce formulas in first order logic — unlike Samson, Mills, and Mueller, who were chiefly interested in synthesizing efficient circuits. Davis and Putnam wrote an unpublished 62-page report “Feasible computational methods in the propositional calculus” (Rensselaer Polytechnic Institute, October 1958) in which a variety of different approaches were considered, such as the removal of unit clauses and pure literals, as well as “case analysis,” that is, backtracking with respect to the subproblems  $F|x$  and  $F|\bar{x}$ . As an alternative to case analysis, they also discussed eliminating the variable  $x$  by resolution. The account of this work that was eventually published [*JACM* 7 (1960), 201–215] concentrated on hand calculation, and omitted case analysis in favor of resolution; but when the process was later implemented on a computer, jointly with George Logemann and Donald Loveland [*CACM* 5 (1962), 394–397], the method of backtracking through different cases was found to work better with respect to memory requirements. (See Davis’s account of these developments in *Handbook of Automated Reasoning* (2001), 3–15.)

This early work didn’t actually cause the satisfiability problem to appear on many people’s mental radar screens, however. Far from it; ten years went by before SAT became an important buzzword. The picture changed in 1971, when Stephen A. Cook showed that satisfiability is the key to solving NP-

recursive  
depth-first  
breadth-first  
dual form  
tautology problem  
unsolvable  
halting problem  
disjunctive normal forms  
Blake  
consensus  
resolvent  
Samson  
Mills  
Quine  
Mueller  
Davis  
Putnam  
first order logic  
Samson  
Mills  
Mueller  
unit clauses  
pure literals  
resolution  
Logemann  
Loveland  
Davis  
Cook  
NP-complete problems

complete problems: He proved that any algorithm to solve a decision problem in nondeterministic polynomial time can be represented efficiently as a conjunction of ternary clauses to be satisfied. (See *STOC* **3** (1971), 151–158. We’ll study NP-completeness in Section 7.9.) Thus, a great multitude of hugely important problems could all be solved rather quickly, if we could only devise a decent algorithm for a *single* problem, 3SAT; and 3SAT seemed almost absurdly simple to solve.

A year of heady optimism following the publication of Cook’s paper soon gave way to the realization that, alas, 3SAT might not be so easy after all. Ideas that looked promising in small cases didn’t scale well, as the problem size was increased. Hence the central focus of work on satisfiability largely retreated into theoretical realms, unrelated to programming practice, except for occasional studies that used SAT as a simple model for the behavior of backtracking algorithms in general. Examples of such investigations, pioneered by A. T. Goldberg, P. W. Purdom, Jr., C. A. Brown, J. V. Franco, and others, appear in exercises 213–216. See P. W. Purdom, Jr., and G. N. Haven, *SICOMP* **26** (1997), 456–483, for a survey of subsequent progress on questions of that kind.

The state of SAT art in the early 90s was well represented by an international programming competition held in 1992 [see M. Buro and H. Kleine Büning, *Bulletin EATCS* **49** (February 1993), 143–151]. The winning programs in that contest can be regarded as the first successful lookahead solvers on the path from Algorithm A to Algorithm L. Max Böhm “took the gold” by choosing the next branch variable based on lexicographically maximal  $(H_1(x), \dots, H_n(x))$ , where

$$H_k(x) = h_k(x) + h_k(\bar{x}) + \min(h_k(x), h_k(\bar{x})), \quad h_k(x) = |\{C \in F \mid x \in C, |C| = k\}|.$$

[See M. Böhm and E. Speckenmeyer, *Ann. Math. Artif. Intelligence* **17** (1996), 381–400. A. Rauzy had independently proposed a somewhat similar branching criterion in 1988; see *Revue d’intelligence artificielle* **2** (1988), 41–60.] The silver medal went to Hermann Stamm, who used strong components of the dependency digraph to narrow the search at each branch node.

Advances in practical algorithms for satisfiability now began to take off. The benchmark programs of 1992 had been chosen at random, but the DIMACS Implementation Challenge of 1993 featured also a large number of structured instances of SAT. The main purpose of this “challenge” was not to crown a winner, but to bring more than 100 researchers together for a three-day workshop, at which they could compare and share results. In retrospect, the best overall performance at that time was arguably achieved by an elaborate lookahead solver called C-SAT, which introduced techniques for detailed exploration of the first-order effects of candidate literals [see O. Dubois, P. Andre, Y. Boufkhad, and J. Carlier, *DIMACS* **26** (1996), 415–436]. Further refinements leading towards the ideas in Algorithm L appeared in a Ph.D. thesis by Jon W. Freeman (Univ. of Pennsylvania, 1995), and in the work of Chu Min Li, who introduced double lookahead [see *Information Processing Letters* **71** (1999), 75–80]. The weighted binary heuristic (67) was proposed by O. Dubois and G. Dequen, *Proc. International Joint Conference on Artificial Intelligence* **17** (2001), 248–253.

nondeterministic polynomial time  
3SAT  
Goldberg  
Purdom  
Brown  
Franco  
Haven  
competition  
Buro  
Kleine Büning  
contest  
lookahead solvers  
Böhm  
Speckenmeyer  
Rauzy  
Stamm  
strong components  
dependency digraph  
C-SAT  
candidate literals  
Dubois  
Andre  
Boufkhad  
Carlier  
Freeman  
Li  
double lookahead  
Dubois  
Dequen



Meanwhile the ideas underlying Algorithm C began to emerge. João P. Marques-Silva, in his 1995 thesis directed by Karem A. Sakallah, discovered how to turn unit-propagation conflicts into one or more clauses learned at “unique implication points,” after which it was often possible to backjump past decisions that didn’t affect the conflict. [See *IEEE Trans.* **C48** (1999), 506–521.] Similar methods were developed independently by R. J. Bayardo, Jr., and R. C. Schrag [*AAAI Conf.* **14** (1997), 203–208], who considered only the special case of clauses that include the current decision literal, but introduced techniques for purging a learned clause when one of its literals was forced to flip its value. Both groups limited the size of learned clauses, and noticed that their new methods gave significant speedups on benchmark problems related to industrial applications.

The existence of fast SAT solvers, coupled with Gunnar Stålmarck’s new ideas about applying logic to computer design [see Swedish patent 467076 (1992)], led to the introduction of bounded model checking techniques by Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu [*LNCS* **1579** (1999), 193–207]. Satisfiability techniques had also been introduced to solve classical planning problems in artificial intelligence [Henry Kautz and Bart Selman, *Proc. European Conf. Artificial Intelligence* **10** (1992), 359–363]. Designers could now verify much larger models than had been possible with BDD methods.

The major breakthroughs appeared in a solver called Chaff [M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, *ACM/IEEE Design Automation Conf.* **38** (2001), 530–535], which had two especially noteworthy innovations: (i) “VSIDS” (the Variable State Independent Decreasing Sum heuristic), a surprisingly effective way to select decision literals, which also worked well with restarts, and which suggested the even better ACT heuristic of Algorithm C that soon replaced it; also (ii) lazy data structures with two watched literals per clause, which made unit propagation much faster with respect to large learned clauses. (A somewhat similar watching scheme, introduced earlier by H. Zhang and M. Stickel [*J. Automated Reasoning* **24** (2000), 277–296], had the disadvantage that it needed to be downdated while backtracking.)

These exciting developments sparked a revival of international SAT competitions, which have been held annually since 2002. The winner in 2002, BerkMin by E. Goldberg and Y. Novikov, has been described well in *Discrete Applied Mathematics* **155** (2007), 1549–1561. And year after year, these challenging contests have continued to spawn further progress. By 2010, more than twice as many benchmarks could be solved in a given period of time as in 2002, using the programs of 2002 and 2010 on the computers of 2010 [see M. Järvisalo, D. Le Berre, O. Roussel, and L. Simon, *AI Magazine* **33**,1 (Spring 2012), 89–94].

The overall champion in 2007 was SATzilla, which was actually not a separate SAT solver but rather a program that knew how to choose intelligently between *other* solvers on any given instance. SATzilla would first take a few seconds to compute basic features of a problem: the distribution of literals per clause and clauses per literal, the balance between positive and negative occurrences of variables, the proximity to Horn clauses, etc. Samples could quickly be taken to estimate how many unit propagations occur at levels 1, 4, 16, 64, 256, and how

Marques-Silva  
Sakallah  
unit-propagation  
conflicts  
unique implication points  
UIP  
backjump  
Bayardo  
Schrag  
decision literal  
purging  
learned clause  
Stålmarck  
bounded model checking  
Biere  
Cimatti  
Clarke  
Zhu  
planning  
Kautz  
Selman  
BDD  
Chaff  
Moskewicz  
Madigan  
Zhao  
Zhang  
Malik  
VSIDS  
restarts  
flushing  
ACT  
activity scores  
watched literals  
unit propagation  
Zhang  
Stickel  
backtracking  
competitions  
BerkMin  
Goldberg  
Novikov  
Järvisalo  
Le Berre  
Roussel  
Simon  
SATzilla  
Horn clauses  
unit propagations

many decisions are needed before reaching a conflict. Based on these numbers, and experience with the performance of the other solvers on the previous year's benchmarks, SATzilla was trained to select the algorithm that appeared most likely to succeed. This “portfolio” approach, which tunes itself nicely to the characteristics of vastly different sets of clauses, has continued to dominate the international competitions ever since. Of course portfolio solvers rely on the existence of “real” solvers, invented independently and bug-free, which shine with respect to particular classes of problems. And of course the winner of competitions may not be the best actual system for practical use. [See L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, *J. Artificial Intelligence Research* **32** (2008), 565–606; *LNCS* **7317** (2012), 228–241; *CACM* **57**, 5 (May 2014), 98–107.]

Historical notes about details of the algorithms, and about important related techniques such as preprocessing and encoding, have already been discussed above as the algorithms and techniques were described.

One recurring theme appears to be that the behavior of SAT solvers is full of surprises: Some of the most important improvements have been introduced for what has turned out to be the wrong reasons, and a theoretical understanding is still far from adequate.

[In future, the next breakthrough might come from “variable learning,” as suggested by Tseytin’s idea of extended resolution: Just as clause learning increases the number of clauses,  $m$ , we might find good ways to increase the number of variables,  $n$ . The subject seems to be far from fully explored.]

portfolio  
tunes itself  
Xu  
Hutter  
Hoos  
Leyton-Brown  
Tseytin  
extended resolution  
Pincusians  
*waerden*  
Local Lemma  
 $W(k_0, k_1, \dots, k_{b-1})$   
monotonic  
binary

## EXERCISES

1. [10] What are the shortest (a) satisfiable (b) unsatisfiable sets of clauses?
2. [20] Travelers to the remote planet Pincus have reported that all the healthy natives like to dance, unless they're lazy. The lazy nondancers are happy, and so are the healthy dancers. The happy nondancers are healthy; but natives who are lazy and healthy aren't happy. Although the unhappy, unhealthy ones are always lazy, the lazy dancers are healthy. What can we conclude about Pincusians, based on these reports?
3. [M21] Exactly how many clauses are in *waerden*( $j, k; n$ )?
4. [22] Show that the 32 constraints of *waerden*(3, 3; 9) in (g) remain unsatisfiable even if up to four of them are removed.
5. [M46] Is  $W(3, k) = \Theta(k^2)$ ?
- ▶ 6. [HM37] Use the Local Lemma to show that  $W(3, k) = \Omega(k^2/(\log k)^3)$ .
  7. [21] Can one satisfy the clauses  $\{(x_i \vee x_{i+2^d} \vee x_{i+2^{d+1}}) \mid 1 \leq i \leq n - 2^{d+1}, d \geq 0\} \cup \{(\bar{x}_i \vee \bar{x}_{i+2^d} \vee \bar{x}_{i+2^{d+1}}) \mid 1 \leq i \leq n - 2^{d+1}, d \geq 0\}$ ?
- ▶ 8. [20] Define clauses *waerden*( $k_0, k_1, \dots, k_{b-1}; n$ ) that are satisfiable if and only if  $n < W(k_0, k_1, \dots, k_{b-1})$ .
  9. [24] Determine the value of  $W(2, 2, k)$  for all  $k \geq 0$ . *Hint*: Consider  $k \bmod 6$ .
- ▶ 10. [21] Show that every satisfiability problem with  $m$  clauses and  $n$  variables can be transformed into an equivalent monotonic problem with  $m+n$  clauses and  $2n$  variables, in which the first  $m$  clauses have only negative literals, and the last  $n$  clauses are binary with two positive literals.

11. [27] (M. Tsimelzon, 1994.) Show that a general 3SAT problem with clauses  $\{C_1, \dots, C_m\}$  and variables  $\{1, \dots, n\}$  can be reduced to a 3D MATCHING problem of size  $10m$  that involves the following cleverly designed triples:

Each clause  $C_j$  corresponds to  $3 \times 10$  vertices, namely  $lj, \bar{l}j, |l|j'$ , and  $|l|j''$  for each  $l \in C_j$ , together with  $wj, xj, yj$ , and  $zj$ , and also  $j'k$  and  $j''k$  for  $1 \leq k \leq 7$ . If  $i$  or  $\bar{i}$  occurs in  $t$  clauses  $C_{j_1}, \dots, C_{j_t}$ , there are  $t$  “true” triples  $\{ij_k, ij'_k, ij''_k\}$  and  $t$  “false” triples  $\{\bar{i}j_k, \bar{i}j'_k, \bar{i}j''_k\}$ , for  $1 \leq k \leq t$ . Each clause  $C_j = (l_1 \vee l_2 \vee l_3)$  also spawns three “satisfiability” triples  $\{\bar{l}_1j, j'1, j''1\}, \{\bar{l}_2j, j'1, j''2\}, \{\bar{l}_3j, j'1, j''3\}$ ; six “filler” triples  $\{l_1j, j'2, j''1\}, \{\bar{l}_1j, j'3, j''1\}, \{l_2j, j'4, j''2\}, \{\bar{l}_2j, j'5, j''2\}, \{l_3j, j'6, j''3\}, \{\bar{l}_3j, j'7, j''3\}$ ; and twelve “gadget” triples  $\{wj, j'2, j''4\}, \{wj, j'4, j''4\}, \{wj, j'6, j''4\}, \{xj, j'2, j''5\}, \{xj, j'5, j''5\}, \{xj, j'7, j''5\}, \{yj, j'3, j''6\}, \{yj, j'4, j''6\}, \{yj, j'7, j''6\}, \{zj, j'3, j''7\}, \{zj, j'5, j''7\}, \{zj, j'6, j''7\}$ . Thus there are  $27m$  triples altogether.

For example, Rivest’s satisfiability problem (6) leads to a 3D matching problem with 216 triples on 240 vertices; the triples that involve vertices 18 and  $\bar{1}8$  are  $\{18, 18', 18''\}, \{\bar{1}8, 18', 11''\}, \{\bar{1}8, 8'1, 8''2\}, \{18, 8'4, 8''2\}, \{\bar{1}8, 8'5, 8''2\}$ .

12. [21] (M. J. H. Heule.) Simplify (13) by exploiting the identity

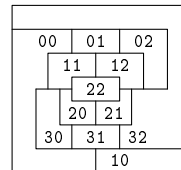
$$S_{\leq 1}(y_1, \dots, y_p) = \exists t (S_{\leq 1}(y_1, \dots, y_j, t) \wedge S_{\leq 1}(\bar{t}, y_{j+1}, \dots, y_p)).$$

13. [24] Exercise 7.2.2.1–00 defines an exact cover problem that corresponds to Langford pairs of order  $n$ . (See page vii.)

- a) What are the constraints analogous to (12) when  $n = 4$ ?
- b) Show that there’s a simple way to avoid duplicate binary clauses such as those in (14), whenever an exact cover problem is converted to clauses using (13).
- c) Describe the corresponding clauses  $langford(4)$  and  $langford'(4)$ .

14. [22] Explain why the clauses (17) might help a SAT solver to color a graph.

15. [24] By comparing the McGregor graph of order 10 in Fig. 33 with the McGregor graph of order 3 shown here, give a precise definition of the vertices and edges of the McGregor graph that has an arbitrary order  $n \geq 3$ . Exactly how many vertices and edges are present in this graph, as a function of  $n$ ?



16. [21] Do McGregor graphs have cliques of size 4?

17. [26] Let  $f(n)$  and  $g(n)$  be the smallest and largest values of  $r$  such that McGregor’s graph of order  $n$  can be 4-colored, and such that some color appears exactly  $r$  times. Use a SAT solver to find as many values of  $f(n)$  and  $g(n)$  as you can.

- ▶ 18. [28] By examining the colorings found in exercise 17, define an explicit way to 4-color a McGregor graph of arbitrary order  $n$ , in such a way that one of the colors is used at most  $\frac{5}{6}n$  times. *Hint:* The construction depends on the value of  $n \bmod 6$ .
- ▶ 19. [29] Continuing exercise 17, let  $h(n)$  be the largest number of regions that can be given two colors simultaneously (without using the clauses (17)). Investigate  $h(n)$ .
- 20. [40] In exactly how many ways can McGregor’s map (Fig. 33) be four-colored?
- 21. [22] Use a SAT solver to find a minimum-size *kernel* in the graph of Fig. 33.
- 22. [20] Color the graph  $\overline{C_5 \boxtimes C_5}$  with the fewest colors. (Two vertices of this graph can receive the same color if and only if they are a king move apart in a  $5 \times 5$  torus.)
- 23. [20] Compare the clauses (18) and (19) to (20) and (21) in the case  $n = 7, r = 4$ .

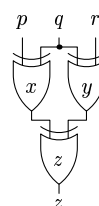
Tsimelzon  
 3D MATCHING problem  
 gadget  
 Rivest  
 Heule  
 symmetric threshold functions  
 encoding  
 exclusion clauses  
 at-most-one  
 exact cover problem  
 Langford pairs  
 $langford(n)$  and  $langford'(n)$   
 exclusion clauses  
 at-most-one  
 McGregor graph  
 kernel  
 strong product  
 complement of a graph  
 king move  
 torus

- **24.** [M32] The clauses obtained from (20) and (21) in the previous exercise can be simplified, because we can remove the two that contain the pure literal  $b_1^2$ .
- Prove that the literal  $b_1^2$  is *always* pure in (20) and (21), when  $r > n/2$ .
  - Show that  $b_1^2$  might also be pure in some cases when  $r < n/2$ .
  - The clauses obtained from (20) and (21) have *many* pure literals  $b_j^k$  when  $r$  has its maximum value  $n - 1$ . Furthermore, their removal makes other literals pure. How many clauses will remain in this case after all pure literals have been eliminated?
  - Show that the complete binary tree with  $n \geq 2$  leaves is obtained from complete binary trees with  $n'$  and  $n'' = n - n'$  leaves, where either  $n'$  or  $n''$  is a power of 2.
  - Let  $a(n, r)$  and  $c(n, r)$  be respectively the number of auxiliary variables  $b_j^k$  and the total number of clauses that remain after all of the pure auxiliary literals have been removed from (20) and (21). What are  $a(2^k, 2^{k-1})$  and  $c(2^k, 2^{k-1})$ ?
  - Prove that  $a(n, r) = a(n, n'') = a(n, n')$  for  $n'' \leq r \leq n'$ , and this common value is  $\max_{1 \leq r < n} a(n, r)$ . Also  $a(n, r) = a(n, n - r)$ ; and  $c(n, r) \geq c(n, n - r)$  if  $r \leq n/2$ .
- 25.** [21] Show that (18)–(19) and (20)–(21) are equally effective when  $r = 2$ .
- 26.** [22] Prove that Sinz's clauses (18) and (19) enforce the cardinality constraint  $x_1 + \dots + x_n \leq r$ . *Hint:* Show that they imply  $s_j^k = 1$  whenever  $x_1 + \dots + x_{j+k-1} \geq k$ .
- 27.** [20] Similarly, prove the correctness of Bailleux and Boufkhad's (20) and (21). *Hint:* They imply  $b_j^k = 1$  whenever the leaves below node  $k$  contain  $j$  or more 1s.
- **28.** [20] What clauses result from (18) and (19) when we want to ensure that  $x_1 + \dots + x_n \geq 1$ ? (This special case converts arbitrary clauses into 3SAT clauses.)
- **29.** [20] Instead of the single constraint  $x_1 + \dots + x_n \leq r$ , suppose we wish to impose a sequence of constraints  $x_1 + \dots + x_i \leq r_i$  for  $1 \leq i \leq n$ . Can this be done nicely with additional clauses and auxiliary variables?
- **30.** [22] If auxiliary variables  $s_j^k$  are used as in (18) and (19) to make  $x_1 + \dots + x_n \leq r$ , while  $s_j^{k'}$  are used to make  $\bar{x}_1 + \dots + \bar{x}_n \leq n - r$ , show that we may unify them by taking  $s_k^{j'} = \overline{s_j^k}$ , for  $1 \leq j \leq n - r$ ,  $1 \leq k \leq r$ . Can (20) and (21) be similarly unified?
- **31.** [28] Let  $F_t(r)$  be the smallest  $n$  for which there is a bit vector  $x_1 \dots x_n$  with  $x_1 + \dots + x_n = r$  and with no  $t$  equally spaced 1s. For example,  $F_3(12) = 30$  because of the unique solution 10110001101000000010110001101. Discuss how  $F_t(n)$  might be computed efficiently with the help of a SAT solver.
- 32.** [15] A *list coloring* is a graph coloring in which  $v$ 's color belongs to a given set  $L(v)$ , for each vertex  $v$ . Represent list coloring as a SAT problem.
- 33.** [21] A *double coloring* of a graph is an assignment of two distinct colors to every vertex in such a way that neighboring vertices share no common colors. Similarly, a  $q$ -tuple coloring assigns  $q$  distinct colors to each vertex. Find double and triple colorings of the cycle graphs  $C_5, C_7, C_9, \dots$ , using as few colors as possible.
- 34.** [HM26] The *fractional coloring number*  $\chi^*(G)$  of a graph  $G$  is defined to be the minimum ratio  $p/q$  for which  $G$  has a  $q$ -tuple coloring that uses  $p$  colors.
- Prove that  $\chi^*(G) \leq \chi(G)$ , and show that equality holds in McGregor's graphs.
  - Let  $S_1, \dots, S_N$  be all the independent subsets of  $G$ 's vertices. Show that
 
$$\chi^*(G) = \min_{\lambda_1, \dots, \lambda_N \geq 0} \{ \lambda_1 + \dots + \lambda_N \mid \sum_{j=1}^N \lambda_j [v \in S_j] = 1 \text{ for all vertices } v \}.$$
 (This is a fractional exact cover problem.)
  - What is the fractional coloring number  $\chi^*(C_n)$  of the cycle graph  $C_n$ ?

pure literal  
complete binary tree  
Sinz  
cardinality constraint  
Bailleux  
Boufkhad  
3SAT  
auxiliary variables  
arithmetic progressions, avoiding  
list coloring  
double coloring  
coloring, multiple  
cycle graphs  
fractional coloring number  
coloring, fractional  
McGregor's graphs  
fractional exact cover  
exact cover, fractional


- d) Consider the following greedy algorithm for coloring  $G$ : Set  $k \leftarrow 0$  and  $G_0 \leftarrow G$ ; while  $G_k$  is nonempty, set  $k \leftarrow k+1$  and  $G_k \leftarrow G_{k-1} \setminus C_k$ , where  $C_k$  is a maximum independent set of  $G_{k-1}$ . Prove that  $k \leq H_{\alpha(G)} \chi^*(G)$ , where  $\alpha(G)$  is the size of  $G$ 's largest independent set; hence  $\chi(G)/\chi^*(G) \leq H_{\alpha(G)} = O(\log n)$ . *Hint*: Let  $t_v = 1/|C_i|$  if  $v \in C_i$ , and show that  $\sum_{v \in S} t_v \leq H_{|S|}$  whenever  $S$  is an independent set.
- 35.** [22] Determine  $\chi^*(G)$  when  $G$  is (a) the graph of the contiguous United States (see 7-(17) and exercise 7-45); (b) the graph of exercise 22.
- **36.** [22] A *radio coloring* of a graph, also known as an  $L(2, 1)$  labeling, is an assignment of integer colors to vertices so that the colors of  $u$  and  $v$  differ by at least 2 when  $u - v$ , and by at least 1 when  $u$  and  $v$  have a common neighbor. (This notion, introduced by Fred Roberts in 1988, was motivated by the problem of assigning channels to radio transmitters, without interference from “close” transmitters and without strong interference from “very close” transmitters.) Find a radio coloring of Fig. 33 that uses only 16 consecutive colors.
- 37.** [20] Find an optimum radio coloring of the contiguous USA graph (see 7-(17)).
- 38.** [M25] How many consecutive colors are needed for a radio coloring of (a) the  $n \times n$  square grid  $P_n \square P_n$ ? (b) the vertices  $\{(x, y, z) \mid x, y, z \geq 0, x + y + z = n\}$ , which form a triangular grid with  $n + 1$  vertices on each side.
- 39.** [M46] Find an optimum radio coloring of the  $n$ -cube, for some value of  $n > 6$ .
- 40.** [01] Is the factorization problem (22) unsatisfiable whenever  $z$  is a prime number?
- 41.** [M21] Determine the number of Boolean operations  $\wedge, \vee, \oplus$  needed to multiply  $m$ -bit numbers by  $n$ -bit numbers with Dadda's scheme, when  $2 \leq m \leq n$ .
- 42.** [21] Tseytin encoding analogous to (24) can be devised also for ternary operations, without introducing any additional variables besides those of the function being encoded. Illustrate this principle by encoding the basic operations  $x \leftarrow t \oplus u \oplus v$  and  $y \leftarrow \langle tuv \rangle$  of a full adder directly, instead of composing them from  $\oplus, \wedge$ , and  $\vee$ .
- **43.** [21] For which integers  $n \geq 2$  do there exist odd palindromic binary numbers  $x = (x_n \dots x_1)_2 = (x_1 \dots x_n)_2$  and  $y = (y_n \dots y_1)_2 = (y_1 \dots y_n)_2$  such that their product  $xy = (z_{m+n} \dots z_1)_2 = (z_1 \dots z_{m+n})_2$  is also palindromic?
- **44.** [30] (Maximum ones.) Find the largest possible value of  $\nu x + \nu y + \nu(xy)$ , namely the greatest total number of 1 bits, over all multiplications of 32-bit binary  $x$  and  $y$ .
- 45.** [20] Specify clauses that constrain  $(z_t \dots z_1)_2$  to be a perfect square.
- 46.** [30] Find the largest perfect square less than  $2^{100}$  that is a binary palindrome.
- **47.** [20] Suppose a circuit such as Fig. 34 has  $m$  outputs and  $n$  inputs, with  $g$  gates that transform two signals into one and  $h$  gates that transform one signal into two. Find a relation between  $g$  and  $h$ , by expressing the total number of wires in two ways.
- 48.** [20] The small circuit shown here has three inputs, three XOR gates,  $p$   $q$   $r$  one fanout gate, eight wires, and one output. Which single-stuck-at faults are detected by each of the eight test patterns  $pqr$ ?
- 49.** [24] Write a program that determines exactly which of the 100 single-stuck-at faults of the circuit in Fig. 34 are detected by each of the 32 possible input patterns. Also find all the minimum sets of test patterns that will discover every such fault (unless it's not detectable).

greedy algorithm  
 maximum independent set  
 contiguous United States  
 radio coloring  
 $L(2, 1)$  labeling  
 Roberts  
 channel assignment  
 contiguous USA  
 square grid  
 grid graphs  
 triangular grid  
*simplex* graph  
 $n$ -cube  
 multiply  
 Dadda  
 Tseytin encoding  
 ternary operations  
 full adder  
 median operation  
 exclusive or, ternary  
 palindromic  
 Maximum ones  
 multiplications  
 palindrome  
 gates  
 wires  
 fanout gates  
 fanout gate  
 single-stuck-at faults









50. [24] Demonstrate Larrabee's method of representing stuck-at faults by describing the clauses that characterize test patterns for the fault " $x_2^1$  stuck at 1" in Fig. 34. (This is the wire that splits off of  $x_2$  and feeds into  $x_2^3$  and  $x_2^4$ , then to  $b_2$  and  $b_3$ ; see Table 1.)
51. [40] Study the behavior of SAT solvers on the problem of finding a small number of test patterns for all of the detectable single-stuck-at faults of the circuit *prod* (32, 32). Can a complete set of patterns for this large circuit be discovered "automatically" (without relying on number theory)?
52. [15] What clauses correspond to (29) and (30) when the *second* case on the left of Table 2,  $f(1, 0, 1, 0, \dots, 1) = 1$ , is taken into account?
- 53. [M20] The numbers in Table 2 are definitely nonrandom. Can you see why?
- 54. [23] Extend Table 2 using the rule in the previous exercise. How many rows are needed before  $f(x)$  has no  $M$ -term representation in DNF, when  $M = 3, 4$ , and 5?
55. [21] Find an equation analogous to (27) that is consistent with Table 2 and has every variable complemented. (Thus the resulting function is monotone decreasing.)
- 56. [22] Equation (27) exhibits a function matching Table 2 that depends on only 8 of the 20 variables. Use a SAT solver to show that we can actually find a suitable  $f$  that depends on only *five* of the  $x_j$ .
- 57. [29] Combining the previous exercise with the methods of Section 7.1.2, exhibit a function  $f$  for Table 2 that can be evaluated with only six Boolean operations(!).
- 58. [20] Discuss adding the clauses  $\bar{p}_{i,j} \vee \bar{q}_{i,j}$  to (29), (30), and (31).
59. [M20] Compute the exact probability that  $\hat{f}(x)$  in (32) differs from  $f(x)$  in (27).
60. [24] Experiment with the problem of learning  $f(x)$  in (27) from training sets of sizes 32 and 64. Use a SAT solver to find a conjectured function,  $\hat{f}(x)$ ; then use BDD methods to determine the probability that this  $\hat{f}(x)$  differs from  $f(x)$  for random  $x$ .
61. [20] Explain how to test when a set of clauses generated from a training set via (29)–(31) is satisfiable only by the function  $f(x)$  in (27).
62. [23] Try to learn a secret small-DNF function with  $N$ -bit training sets  $x^{(0)}, x^{(1)}, x^{(2)}, \dots$ , where  $x^{(0)}$  is random but each bit of  $x^{(k)} \oplus x^{(k-1)}$  for  $k > 0$  is 1 with probability  $p$ . (Thus, if  $p$  is small, successive data points will tend to be near each other.) Do such sets turn out to be more efficient in practice than the purely random ones that arise for  $p = 1/2$ ?
- 63. [20] Given an  $n$ -network  $\alpha = [i_1 : j_1][i_2 : j_2] \dots [i_r : j_r]$ , as defined in the exercises for Section 5.3.4, explain how to use a SAT solver to test whether or not  $\alpha$  is a sorting network. *Hint*: Use Theorem 5.3.4Z.
64. [26] The exact minimum time  $\hat{T}(n)$  of a sorting network for  $n$  elements is a famous unsolved problem, and the fact that  $\hat{T}(9) = 7$  was first established in 1987 by running a highly optimized program for many hours on a Cray 2 supercomputer.  
Show that this result can now be proved with a SAT solver in less than a second(!).
- 65. [28] Describe encodings of the Life transition function (35) into clauses.
- Use only the variables  $x'_{ij}$  and  $x_{ij}$ .
  - Use auxiliary variables as in the Bailleux and Boufkhad encoding (20)–(21), sharing intermediate results between neighboring cells as discussed in the text.
66. [24] Use a SAT solver to find short counterparts to Fig. 35 in which (a)  $X_1 = \mathbf{LIFE}$ ; (b)  $X_2 = \mathbf{LIFE}$ . In each case  $X_0$  should have the smallest possible number of live cells.

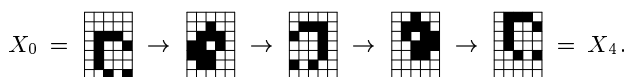
Larrabee  
 automatic test pattern gen  
 single-stuck-at faults  
 prod  
 number theory  
 monotone decreasing  
 depends on  
 BDD  
 training sets  
 comparator modules  
 sorting network  
 Cray 2  
 Life  
 Bailleux  
 Boufkhad

- 67.** [24] Find a mobile chessboard path  $X_0 \rightarrow X_1 \rightarrow \dots \rightarrow X_{21}$  with no more than five cells alive in each  $X_t$ . (The glider in (37) leaves the board after  $X_{20}$ .) How about  $X_{22}$ ?
- 68.** [39] Find a maximum-length mobile path in which 6 to 10 cells are always alive.
- 69.** [23] Find all (a) still lifes and (b) oscillators of period  $> 1$  that live in a  $4 \times 4$  board.
- 70.** [21] The live cells of an oscillator are divided into a *rotor* (those that change) and a *stator* (those that stay alive).
- Show that the rotor cannot be just a single cell.
  - Find the smallest example of an oscillator whose rotor is  $\square \leftrightarrow \blacksquare$ .
  - Similarly, find the smallest oscillators of period 3 whose rotors have the following forms:  $\square \rightarrow \blacksquare \rightarrow \square \rightarrow \square$ ;  $\square \rightarrow \blacksquare \rightarrow \blacksquare \rightarrow \square$ ;  $\blacksquare \rightarrow \square \rightarrow \square \rightarrow \blacksquare$ .
- **71.** [22] When looking for sequences of Life transition on a square grid, an asymmetrical solution will appear in eight different forms, because the grid has eight different symmetries. Furthermore, an asymmetrical periodic solution will appear in  $8r$  different forms, if  $r$  is the length of the period.
- Explain how to add further clauses so that essentially equivalent solutions will occur only once: Only “canonical forms” will satisfy the conditions.
- 72.** [28] Oscillators of period 3 are particularly intriguing, because Life seems so inherently binary.
- What are the smallest such oscillators (in terms of bounding box)?
  - Find period-3 oscillators of sizes  $9 \times n$  and  $10 \times n$ , with  $n$  odd, that have “fourfold symmetry”: The patterns are unchanged after left-right and/or up-down reflection. (Such patterns are not only pleasant to look at, they also are much easier to find, because we need only consider about one-fourth as many variables.)
  - What period-3 oscillators with fourfold symmetry have the most possible live cells, on grids of sizes  $15 \times 15$ ,  $15 \times 16$ , and  $16 \times 16$ ?
  - The period-3 oscillator shown here has another kind of four-way symmetry, because it’s unchanged after  $90^\circ$  rotation. (It was discovered in 1972 by Robert Wainwright, who called it “snake dance” because its stator involves four snakes.) What period-3 oscillators with  $90^\circ$  symmetry have the most possible live cells, on grids of sizes  $15 \times 15$  and  $16 \times 16$ ?
- 
- **73.** [21] (*Mobile flipflops.*) An oscillator of period 2 is called a *flipflop*, and the Life patterns of mobile flipflops are particularly appealing: Each cell is either blank (dead at every time  $t$ ) or type A (alive when  $t$  is even) or type B (alive when  $t$  is odd). Every nonblank cell (i) has exactly three neighbors of the other type, and (ii) doesn’t have exactly two or three neighbors of the same type.
- The blank cells of a mobile flipflop also satisfy a special condition. What is it?
  - Find a mobile flipflop on an  $8 \times 8$  grid, with top row  $\square \square \square \square \square \square \square \square$ .
  - Find patterns that are mobile flipflops on  $m \times n$  toruses for various  $m$  and  $n$ . (Thus, if replicated indefinitely, each one will tile the plane with an infinite mobile flipflop.) *Hint:* One solution has no blank cells whatsoever; another has blank cells like a checkerboard.
- 74.** [M28] Continuing the previous exercise, prove that no nonblank cell of a finite mobile flipflop has more than one neighbor of its own type. (This fact greatly speeds up the search for finite mobile flipflops.) Can two type A cells be diagonally adjacent?
- 75.** [M22] (Stephen Silver, 2000.) Show that a finite, mobile oscillator of period  $p \geq 3$  must have some cell that is alive more than once during the cycle.

mobile  
chessboard  
glider  
oscillator  
rotor  
stator  
grid  
symmetries  
canonical forms  
fourfold symmetry  
reflection  
rotational symmetry  
chiral symmetry [rotation but not reflection]  
Wainwright  
snake dance  
Mobile flipflops  
oscillator  
flipflop  
toruses  
Silver

- 76. [41] Construct a mobile Life oscillator of period 3.
- 77. [20] “Step  $X_{-1}$ ,” which precedes  $X_0$  in (38), has the glider configuration  instead of . What conditions on the still life  $X_5$  will ensure that state  $X_0$  is indeed reached? (We don’t want digestion to begin prematurely.)
- 78. [21] Find a solution to the four-step eater problem in (38) that works on a  $7 \times n$  grid, for some  $n$ , instead of  $8 \times 8$ .
- 79. [23] What happens if the glider meets the eater of (39) in its opposite phase (namely  instead of )?
- 80. [21] To counteract the problem in the previous exercise, find an eater that is symmetrical when reflected about a diagonal, so that it eats both  and . (You’ll have to go larger than  $8 \times 8$ , and you’ll have to wait longer for digestion.)
- 81. [21] Conway discovered a remarkable “spaceship,” where  $X_4$  is  $X_0$  shifted up 2:

Conway  
spaceship  
Light speed  
benchmark tests  
Garden of Eden  
bitmap  
orphan  
Alice and Bob  
mutual exclusion-



Is there a left-right symmetrical still life that will eat such spaceships?

- 82. [22] (*Light speed.*) Imagine Life on an infinite plane, with all cells dead at time 0 except in the lower left quadrant. More precisely, suppose  $X_t = (x_{tij})$  is defined for all  $t \geq 0$  and all integers  $-\infty < i, j < +\infty$ , and that  $x_{0ij} = 0$  whenever  $i > 0$  or  $j > 0$ .
  - a) Prove that  $x_{tij} = 0$  whenever  $0 \leq t < \max(i, j)$ .
  - b) Furthermore  $x_{tij} = 0$  when  $0 \leq -i \leq j$  and  $0 \leq t < i + 2j$ .
  - c) And  $x_{tij} = 0$  for  $0 \leq t < 2i + 2j$ , if  $i \geq 0$  and  $j \geq 0$ . *Hint:* If  $x_{tij} = 0$  whenever  $i \geq -j$ , prove that  $x_{tij} = 0$  whenever  $i > -j$ .

83. [21] According to the previous exercise, the earliest possible time that cell  $(i, j)$  can become alive, if all initial life is confined to the lower left quadrant of the plane, is at least

$$f(i, j) = i[i \geq 0] + j[j \geq 0] + (i + j)[i + j \geq 0].$$

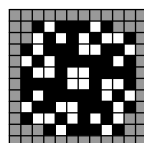
For example, when  $|i| \leq 5$  and  $|j| \leq 5$  the values of  $f(i, j)$  are shown at the right.

Let  $f^*(i, j)$  be the actual minimum time at which cell  $(i, j)$  can be alive, for some such initial state. Devise a set of clauses by which a SAT solver can test whether or not  $f^*(i_0, j_0) = f(i_0, j_0)$ , given  $i_0$  and  $j_0$ . (Such clauses make interesting benchmark tests.)

|   |   |   |   |   |    |    |    |    |    |    |
|---|---|---|---|---|----|----|----|----|----|----|
| 5 | 6 | 7 | 8 | 9 | 10 | 12 | 14 | 16 | 18 | 20 |
| 4 | 4 | 5 | 6 | 7 | 8  | 10 | 12 | 14 | 16 | 18 |
| 3 | 3 | 3 | 4 | 5 | 6  | 8  | 10 | 12 | 14 | 16 |
| 2 | 2 | 2 | 2 | 3 | 4  | 6  | 8  | 10 | 12 | 14 |
| 1 | 1 | 1 | 1 | 1 | 2  | 4  | 6  | 8  | 10 | 12 |
| 0 | 0 | 0 | 0 | 0 | 0  | 2  | 4  | 6  | 8  | 10 |
| 0 | 0 | 0 | 0 | 0 | 0  | 1  | 3  | 5  | 7  | 9  |
| 0 | 0 | 0 | 0 | 0 | 0  | 1  | 2  | 4  | 6  | 8  |
| 0 | 0 | 0 | 0 | 0 | 0  | 1  | 2  | 3  | 5  | 7  |
| 0 | 0 | 0 | 0 | 0 | 0  | 1  | 2  | 3  | 4  | 6  |
| 0 | 0 | 0 | 0 | 0 | 0  | 1  | 2  | 3  | 4  | 5  |

- 84. [33] Prove that  $f^*(i, j) = f(i, j)$  in the following cases when  $j > 0$ : (a)  $i = j$ ,  $i = j + 1$ , and  $i = j - 1$ . (b)  $i = 0$  and  $i = -1$ . (c)  $i = 1 - j$ . (d)  $i = j - 2$ . (e)  $i = -2$ .

- 85. [39] A *Garden of Eden* is a state of Life that has no predecessor.
  - a) If the pattern of 92 cells illustrated here occurs anywhere within a bitmap  $X$ , verify that  $X$  is a Garden of Eden. (The gray cells can be either dead or alive.)
  - b) This “orphan” pattern, found with a SAT solver’s help, is the smallest that is currently known. Can you imagine how it was discovered?



- 86. [M23] How many Life predecessors does a random  $10 \times 10$  bitmap have, on average?
- 87. [21] Explain why the clauses (42) represent Alice and Bob’s programs (40), and give a general recipe for converting such programs into equivalent sets of clauses.



**88.** [18] Satisfy (41) and (42) for  $0 \leq t < 6$ , and the  $20 \times 6$  additional binary clauses that exclude multiple states, along with the “embarrassing” unit clauses  $(A3_6) \wedge (B3_6)$ .

**89.** [21] Here’s a mutual-exclusion protocol once recommended in 1966. Does it work?

- |                                      |                                      |
|--------------------------------------|--------------------------------------|
| A0. Maybe go to A1.                  | B0. Maybe go to B1.                  |
| A1. Set $a \leftarrow 1$ , go to A2. | B1. Set $b \leftarrow 1$ , go to B2. |
| A2. If $l$ go to A3, else to A5.     | B2. If $l$ go to B5, else to B3.     |
| A3. If $b$ go to A3, else to A4.     | B3. If $a$ go to B3, else to B4.     |
| A4. Set $l \leftarrow 0$ , go to A2. | B4. Set $l \leftarrow 1$ , go to B2. |
| A5. Critical, go to A6.              | B5. Critical, go to B6.              |
| A6. Set $a \leftarrow 0$ , go to A0. | B6. Set $b \leftarrow 0$ , go to B0. |

mutual-exclusion protocol  
starvation  
initial state  
starvation cycle  
simple path  
invariant  
mutual exclusion  
Peterson  
starvation cycles  
pure cycle  
simple cycle  
Dekker

**90.** [20] Show that (43), (45), and (46) permit starvation, by satisfying (47) and (48).

**91.** [M21] Formally speaking, Alice is said to “starve” if there is (i) an infinite sequence of transitions  $X_0 \rightarrow X_1 \rightarrow \dots$  starting from the initial state  $X_0$ , and (ii) an infinite sequence  $@_0, @_1, \dots$  of Boolean “bumps” that changes infinitely often, such that (iii) Alice is in a “maybe” or “critical” state only a finite number of times. Prove that this can happen if and only if there is a starvation cycle (47) as discussed in the text.

**92.** [20] Suggest  $O(r^2)$  clauses with which we can determine whether or not a mutual exclusion protocol permits a path  $X_0 \rightarrow X_1 \rightarrow \dots \rightarrow X_r$  of *distinct* states.

**93.** [20] What clauses correspond to the term  $\neg\Phi(X')$  in (51)?

► **94.** [21] Suppose we know that  $(X_0 \rightarrow X_1 \rightarrow \dots \rightarrow X_r) \wedge \neg\Phi(X_r)$  is unsatisfiable for  $0 \leq r \leq k$ . What clauses will guarantee that  $\Phi$  is invariant? (The case  $k = 1$  is (51).)

**95.** [20] Using invariants like (50), prove that (45) and (46) provide mutual exclusion.

**96.** [22] Find all solutions to (52) when  $r = 2$ . Also illustrate the fact that invariants are extremely helpful, by finding a solution with distinct states  $X_0, X_1, \dots, X_r$  and with  $r$  substantially greater than 2, if the clauses involving  $\Phi$  are removed.

**97.** [20] Can states A6 and B6 occur simultaneously in Peterson’s protocol (49)?

► **98.** [M23] This exercise is about proving the nonexistence of starvation cycles (47).

- A cycle of states is called “pure” if one of the players is never bumped, and “simple” if no state is repeated. Prove that the shortest impure cycle, if any, is either simple or consists of two simple pure cycles that share a common state.
- If Alice is starved by some cycle with protocol (49), we know that she is never in states A0 or A5 within the cycle. Show that she can’t be in A1, A2, or A6 either.
- Construct clauses to test whether there exist states  $X_0 \rightarrow X_1 \rightarrow \dots \rightarrow X_r$ , with  $X_0$  arbitrary, such that  $(X_0 X_1 \dots X_{k-1})$  is a starvation cycle for some  $k \leq r$ .
- Therefore we can conclude that (49) is starvation-free without much extra work.

**99.** [25] Th. Dekker devised the first correct mutual-exclusion protocol in 1965:

- |                                      |                                      |
|--------------------------------------|--------------------------------------|
| A0. Maybe go to A1.                  | B0. Maybe go to B1.                  |
| A1. Set $a \leftarrow 1$ , go to A2. | B1. Set $b \leftarrow 1$ , go to B2. |
| A2. If $b$ go to A3, else to A6.     | B2. If $a$ go to B3, else to B6.     |
| A3. If $l$ go to A4, else to A2.     | B3. If $l$ go to B2, else to B4.     |
| A4. Set $a \leftarrow 0$ , go to A5. | B4. Set $b \leftarrow 0$ , go to B5. |
| A5. If $l$ go to A5, else to A1.     | B5. If $l$ go to B1, else to B5.     |
| A6. Critical, go to A7.              | B6. Critical, go to B7.              |
| A7. Set $l \leftarrow 1$ , go to A8. | B7. Set $l \leftarrow 0$ , go to B8. |
| A8. Set $a \leftarrow 0$ , go to A0. | B8. Set $b \leftarrow 0$ , go to B0. |

Use bounded model checking to verify its correctness.

100. [22] Show that the following protocol can starve one player but not the other:

- |                                      |                                      |
|--------------------------------------|--------------------------------------|
| A0. Maybe go to A1.                  | B0. Maybe go to B1.                  |
| A1. Set $a \leftarrow 1$ , go to A2. | B1. Set $b \leftarrow 1$ , go to B2. |
| A2. If $b$ go to A2, else to A3.     | B2. If $a$ go to B3, else to B5.     |
| A3. Critical, go to A4.              | B3. Set $b \leftarrow 0$ , go to B4. |
| A4. Set $a \leftarrow 0$ , go to A0. | B4. If $a$ go to B4, else to B1.     |
|                                      | B5. Critical, go to B6.              |
|                                      | B6. Set $b \leftarrow 0$ , go to B0. |

simultaneous write/write  
simultaneous read/write  
nondeterministically  
flickering  
bishops  
tomographically balanced  
Basket weavers  
gipatsi patterns  
pixels, rotated  
grid, rotated

► 101. [31] Protocol (49) has the potential defect that Alice and Bob might both be trying to set the value of  $l$  at the same time. Design a mutual-exclusion protocol in which each of them controls two binary signals, visible to the other. *Hint:* The method of the previous exercise can be enclosed in another protocol.

102. [22] If Alice is setting a variable at the same time that Bob is trying to read it, we might want to consider a more stringent model under which he sees either 0 or 1, nondeterministically. (And if he looks  $k$  times before she moves to the next step, he might see  $2^k$  possible sequences of bits.) Explain how to handle this model of “flickering” variables by modifying the clauses of exercise 87.

103. [18] (Do this exercise *by hand*, it's fun!) Find the  $7 \times 21$  image whose tomographic sums are  $(r_1, \dots, r_7) = (1, 0, 13, 6, 12, 7, 19)$ ;  $(c_1, \dots, c_{21}) = (4, 3, 3, 4, 1, 6, 1, 3, 3, 3, 5, 1, 1, 5, 1, 5, 1, 5, 1, 1, 1)$ ;  $(a_1, \dots, a_{27}) = (0, 0, 1, 2, 2, 3, 2, 3, 3, 2, 3, 3, 4, 3, 2, 3, 3, 4, 3, 2, 2, 1, 1, 1, 1, 1)$ ;  $(b_1, \dots, b_{27}) = (0, 0, 0, 0, 0, 1, 3, 3, 4, 3, 2, 2, 2, 3, 3, 4, 2, 3, 3, 3, 3, 3, 4, 3, 2, 1, 1)$ .

104. [M21] For which  $m$  and  $n$  is it possible to satisfy the digital tomography problem with  $a_d = b_d = 1$  for  $0 < d < m + n$ ? (Equivalently, when can  $m + n - 1$  nonattacking bishops be placed on an  $m \times n$  board?)

► 105. [M28] A matrix whose entries are  $\{-1, 0, +1\}$  is *tomographically balanced* if its row, column, and diagonal sums are all zero. Two binary images  $X = (x_{ij})$  and  $X' = (x'_{ij})$  clearly have the same row, column, and diagonal sums if and only if  $X - X'$  is tomographically balanced.

- Suppose  $Y$  is tomographically balanced and has  $m$  rows,  $n$  columns, and  $t$  occurrences of  $+1$ . How many  $m \times n$  binary matrices  $X$  and  $X'$  satisfy  $X - X' = Y$ ?
- Express the condition “ $Y$  is tomographically balanced” in terms of clauses, with the values  $\{-1, 0, +1\}$  represented respectively by the 2-bit codes  $\{10, 00, 01\}$ .
- Count the number  $T(m, n)$  of tomographically balanced matrices, for  $m, n \leq 8$ .
- How many such matrices have exactly four occurrences of  $+1$ ?
- At most how many  $+1$ s can a  $2n \times 2n$  tomographically balanced matrix have?
- True or false: The positions of the  $+1$ s determine the positions of the  $-1$ s.

106. [M20] Determine a generous upper bound on the possible number of different sets of input data  $\{r_i, c_j, a_d, b_d\}$  that might be given to a  $25 \times 30$  digital tomography problem, by assuming that each of those sums independently has any of its possible values. How does this bound compare to  $2^{750}$ ?

► 107. [22] Basket weavers from the Tonga culture of Inhambane, Mozambique, have developed appealing periodic designs called “gipatsi patterns” such as this:



(Notice that an ordinary pixel grid has been rotated by  $45^\circ$ .) Formally speaking, a gipatsi pattern of period  $p$  and width  $n$  is a  $p \times n$  binary matrix  $(x_{i,j})$  in which we have

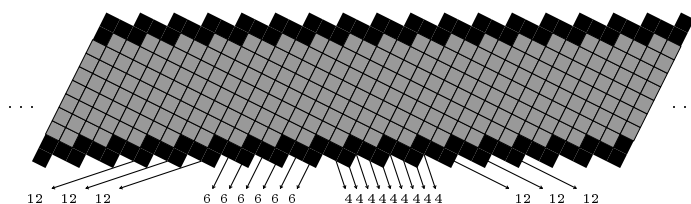
$x_{i,1} = x_{i,n} = 1$  for  $1 \leq i \leq p$ . Row  $i$  of the matrix is to be shifted right by  $i - 1$  places in the actual pattern. The example above has  $p = 6$ ,  $n = 13$ , and the first row of its matrix is 1111101111101. Such a pattern has row sums  $r_i = \sum_{j=1}^n x_{i,j}$  for  $1 \leq i \leq p$  and column sums  $c_j = \sum_{i=1}^p x_{i,j}$  for  $1 \leq j \leq n$ , as usual. By analogy with (53), it also has

$$a_d = \sum_{i+j \equiv d \pmod{p}} x_{i,j}, \quad 1 \leq d \leq p; \quad b_d = \sum_{2i+j \equiv d \pmod{2p}} x_{i,j}, \quad 1 \leq d \leq 2p.$$

- a) What are the tomographic parameters  $r_i$ ,  $c_j$ ,  $a_d$ , and  $b_d$  in the example pattern?
- b) Do any other gipatsi patterns have the same parameters?

**108.** [23] The column sums  $c_j$  in the previous exercise are somewhat artificial, because they count black pixels in only a small part of an infinite line. If we rotate the grid at a different angle, however, we can obtain infinite periodic patterns for which each of Fig. 36's four directions encounters only a finite number of pixels.

Design a pattern of period 6 in which parallel lines always have equal tomographic projections, by changing each of the gray pixels in the following diagram to either white or black:



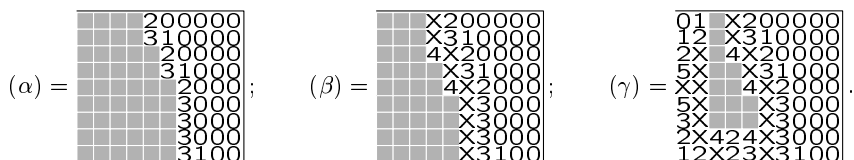
- ▶ **109.** [20] Explain how to find the lexicographically smallest solution  $x_1 \dots x_n$  to a satisfiability problem, using a SAT solver repeatedly. (See Fig. 37(a).)
- 110.** [19] What are the lexicographically (first, last) solutions to *waerden*(3, 10; 96)?
- 111.** [40] The lexicographically first and last solutions to the “Cheshire Tom” problem in Fig. 37 are based on the top-to-bottom-and-left-to-right ordering of pixels. Experiment with other pixel orderings — for example, try bottom-to-top-and-right-to-left.
- 112.** [46] Exactly how many solutions does the tomography problem of Fig. 36 have?
- ▶ **113.** [30] Prove that the digital tomography problem is NP-complete, even if the marginal sums  $r$ ,  $c$ ,  $a$ ,  $b$  are binary: Show that an efficient algorithm to decide whether or not an  $n \times n$  pixel image  $(x_{ij})$  exists, having given 0–1 values of  $r_i = \sum_j x_{ij}$ ,  $c_j = \sum_i x_{ij}$ ,  $a_d = \sum_{i+j=d+1} x_{ij}$ , and  $b_d = \sum_{i-j=d-n} x_{ij}$ , could be used to solve the binary tensor contingency problem of exercise 212(a).

**114.** [27] Each cell  $(i, j)$  of a given rectangular grid either contains a land mine ( $x_{i,j} = 1$ ) or is safe ( $x_{i,j} = 0$ ). In the game of *Minesweeper*, you are supposed to identify all of the hidden mines, by probing locations that you hope are safe: If you decide to probe a cell with  $x_{i,j} = 1$ , the mine explodes and you die (at least virtually). But if  $x_{i,j} = 0$  you're told the number  $n_{i,j}$  of neighboring cells that contain mines,  $0 \leq n_{i,j} \leq 8$ , and you live to make another probe. By carefully considering these numeric clues, you can often continue with completely safe probes, eventually touching every mine-free cell.

For example, suppose the hidden mines happen to match the  $25 \times 30$  pattern of the Cheshire cat (Fig. 36), and you start by probing the upper right corner. That cell turns out to be safe, and you learn that  $n_{1,30} = 0$ ; hence it's safe to probe all three neighbors of  $(1, 30)$ . Continuing in this vein soon leads to illustration ( $\alpha$ ) below, which depicts information about cells  $(i, j)$  for  $1 \leq i \leq 9$  and  $21 \leq j \leq 30$ ; unprobed cells are

lexicographically smallest solution  
 interactive SAT solving  
 van der Waerden numbers  
*waerden*  
 digital tomography  
 NP-complete  
 0–1 matrices  
 binary tensor contingency problem  
 contingency  
 grid  
 land mine  
 Minesweeper

shown in gray, otherwise the value of  $n_{i,j}$  appears. From this data it's easy to deduce that  $x_{1,24} = x_{2,24} = x_{3,25} = x_{4,25} = \dots = x_{9,26} = 1$ ; you'll never want to probe in those places, so you can mark such cells with X, arriving at state  $(\beta)$  since  $n_{3,24} = n_{5,25} = 4$ . Further progress downward to row 17, then leftward and up, leads without difficulty to state  $(\gamma)$ . (Notice that this process is analogous to digital tomography, because you're trying to reconstruct a binary array from information about partial sums.)



Minesweeper  
Life  
flipflops  
serial correlation coefficient  
runs of 1s  
Bailleux  
Boufkhad  
tatami tiling  
dominoes  
waerden  
conditioning  
reduced clauses  
all solutions  
satisfying assignments

- a) Now find safe probes for all thirteen of the cells that remain gray in  $(\gamma)$ .
- b) Exactly how much of the Cheshire cat can be revealed without making any unsafe guesses, if you're told in advance that (i)  $x_{1,1} = 0$ ? (ii)  $x_{1,30} = 0$ ? (iii)  $x_{25,1} = 0$ ? (iv)  $x_{25,30} = 0$ ? (v) all four corners are safe? *Hint*: A SAT solver can help.

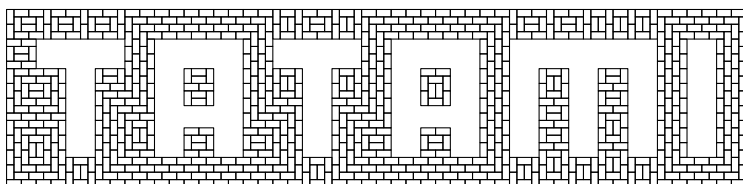
115. [25] Empirically estimate the probability that a  $9 \times 9$  game of Minesweeper, with 10 randomly placed mines, can be won with entirely safe probes after the first guess.



116. [22] Find examples of Life flipflops for which  $X$  and  $X'$  are tomographically equal.

117. [23] Given a sequence  $x = x_1 \dots x_n$ , let  $\nu^{(2)}x = x_1x_2 + x_2x_3 + \dots + x_{n-1}x_n$ . (A similar sum appears in the serial correlation coefficient, 3.3.2-(23).)

- a) Show that, when  $x$  is a binary sequence, the number of runs of 1s in  $x$  can be expressed in terms of  $\nu x$  and  $\nu^{(2)}x$ .
- b) Explain how to encode the condition  $\nu^{(2)}x \leq r$  as a set of clauses, by modifying the cardinality constraints (20)-(21) of Bailleux and Boufkhad.
- c) Similarly, encode the condition  $\nu^{(2)}x \geq r$ .

118. [20] A *tatami tiling* is a covering by dominoes in which no three share a corner:



(Notice that  is disallowed, but  would be fine.) Explain how to use a SAT solver to find a tatami tiling that covers a given set of pixels, unless no such tiling exists.

119. [18] Let  $F = waerden(3, 3; 9)$  be the 32 clauses in (9). For which literal  $l$  is the reduced formula  $F|l$  smallest? Exhibit the resulting clauses.

120. [M20] True or false:  $F|L = \{C \setminus \bar{L} \mid C \in F \text{ and } C \cap L = \emptyset\}$ , if  $\bar{L} = \{\bar{l} \mid l \in L\}$ .

121. [21] Spell out the changes to the link fields in the data structures, by expanding the higher-level descriptions that appear in steps A3, A4, A7, and A8 of Algorithm A.

► 122. [21] Modify Algorithm A so that it finds *all* satisfying assignments of the clauses.

123. [17] Show the contents of the internal data structures L, START, and LINK when Algorithm B or Algorithm D begins to process the seven clauses  $R'$  of (7).

- **124.** [21] Spell out the low-level link field operations that are sketched in step B3.
- **125.** [20] Modify Algorithm B so that it finds *all* satisfying assignments of the clauses.
- 126.** [20] Extend the computation in (59) by one more step.
- 127.** [17] What move codes  $m_1 \dots m_d$  correspond to the computation sketched in (59), just before and after backtracking occurs?
- 128.** [19] Describe the entire computation by which Algorithm D proves that Rivest's clauses (6) are unsatisfiable, using a format like (59). (See Fig. 39.)
- 129.** [20] In the context of Algorithm D, design a subroutine that, given a literal  $l$ , returns 1 or 0 according as  $l$  is or is not being watched in some clause whose other literals are entirely false.
- 130.** [22] What low-level list processing operations are needed to “clear the watch list for  $\bar{x}_k$ ” in step D6?
- **131.** [30] After Algorithm D exits step D3 without finding any unit clauses, it has examined the watch lists of every free variable. Therefore it could have computed the lengths of those watch lists, with little additional cost; and information about those lengths could be used to make a more informed decision about the variable that's chosen for branching in step D4. Experiment with different branching heuristics of this kind.
- **132.** [22] Theorem 7.1.1K tells us that every 2SAT problem can be solved in linear time. Is there a sequence of 2SAT clauses for which Algorithm D takes exponential time?
- **133.** [25] The size of a backtrack tree such as Fig. 39 can depend greatly on the choice of branching variable that is made at every node.
- Find a backtrack tree for *waerden*(3, 3; 9) that has the fewest possible nodes.
  - What's the *largest* backtrack tree for that problem?
- 134.** [22] The BIMP tables used by Algorithm L are sequential lists of dynamically varying size. One attractive way to implement them is to begin with every list having capacity 4 (say); then when a list needs to become larger, its capacity can be doubled. Adapt the buddy system (Algorithm 2.5R) to this situation. (Lists that shrink when backtracking needn't free their memory, since they're likely to grow again later.)
- **135.** [16] The literals  $l'$  in  $\text{BIMP}(l)$  are those for which  $l \rightarrow l'$  in the “implication digraph” of a given satisfiability problem. How can we easily find all of the literals  $l''$  such that  $l'' \rightarrow l$ , given  $l$ ?
- 136.** [15] What pairs will be in  $\text{TIMP}(\bar{3})$ , before and after  $x_5$  is set to zero with respect to the clauses (9) of *waerden*(3, 3; 9), assuming that we are on decision level  $d = 0$ ?
- 137.** [24] Spell out in detail the processes of (a) removing a variable  $X$  from the free list and from all pairs in TIMP lists (step L7 of Algorithm L), and of (b) restoring it again later (step L12). Exactly how do the data structures change?
- **138.** [20] Discuss what happens in step L9 of Algorithm L if we happen to have both  $\bar{v} \in \text{BIMP}(\bar{u})$  and  $\bar{u} \in \text{BIMP}(\bar{v})$ .
- 139.** [25] (*Compensation resolvents*.) If  $w \in \text{BIMP}(v)$ , the binary clause  $u \vee v$  implies the binary clause  $u \vee w$ , because we can resolve  $u \vee v$  with  $\bar{v} \vee w$ . Thus step L9 could exploit each new binary clause further, by appending  $w$  as well as  $v$  to  $\text{BIMP}(\bar{u})$ , for all such  $w$ . Discuss how to do this efficiently.

satisfying assignments  
Rivest  
watched  
unit clauses  
watch list  
branching heuristics  
2SAT  
exponential time  
backtrack tree, optimum  
BIMP tables  
sequential lists  
dynamic storage allocation  
buddy system  
implication digraph  
Compensation resolvents  
resolve

- 140.** [21] The FORCE, BRANCH, BACKF, and BACKI arrays in Algorithm L will obviously never contain more than  $n$  items each. Is there a fairly small upper bound on the maximum possible size of ISTACK?
- 141.** [18] Algorithm L might increase ISTACK so often that it overflows the size of the IST( $l$ ) fields. How can the mechanism of (63) avoid bugs in such a case?
- 142.** [24] Algorithms A, B, and D can display their current progress by exhibiting a sequence of move codes  $m_1 \dots m_d$  such as (58) and (60); but Algorithm L has no such codes. Show that an analogous sequence  $m_1 \dots m_F$  could be printed in step L2, if desired. Use the codes of Algorithm D; but extend them to show  $m_j = 6$  (or 7) if  $R_{j-1}$  is a true (or false) literal whose value was found to be forced by Algorithm X, or forced by being a unit clause in the input.
- **143.** [30] Modify Algorithm L so that it will apply to nonempty clauses of any size. Call a clause *big* if its size is greater than 2. Instead of TIMP tables, represent every big clause by ‘KINX’ and ‘CINX’ tables: Every literal  $l$  has a sequential list KINX( $l$ ) of big clause numbers; every big clause  $c$  has a sequential list CINX( $c$ ) of literals;  $c$  is in KINX( $l$ ) if and only if  $l$  is in CINX( $c$ ). The current number of active clauses containing  $l$  is indicated by KSIZE( $l$ ); the current number of active literals in  $c$  is indicated by CSIZE( $c$ ).
- 144.** [15] True or false: If  $l$  doesn’t appear in any clause,  $h'(l) = 0.1$  in (65).
- 145.** [23] Starting with  $h(l) = 1$  for each of the 18 literals  $l$  in *waerden*(3, 3; 9), find successively refined estimates  $h'(l)$ ,  $h''(l)$ ,  $\dots$ , using (65) with respect to the 32 ternary clauses (9). Then, assuming that  $x_5$  has been set false as in exercise 136, and that the resulting binary clauses 13, 19, 28, 34, 37, 46, 67, 79 have been included in the BIMP tables, do the same for the 16 literals that remain at depth  $d = 1$ .
- 146.** [25] Suggest an alternative to (64) and (65) for use when Algorithm L has been extended to nonternary clauses as in exercise 143. (Strive for simplicity.)
- 147.** [05] Evaluate  $C_{\max}$  in (66) for  $d = 0, 1, 10, 20, 30$ , using the default  $C_0$  and  $C_1$ .
- 148.** [21] Equation (66) bounds the maximum number of candidates using a formula that depends on the current depth  $d$ , but not on the total number of free variables. The same cutoffs are used in problems with any number of variables. Why is that a reasonable strategy?
- **149.** [26] Devise a data structure that makes it convenient to tell whether a given variable  $x$  is a “participant” in Algorithm L.
- 150.** [21] Continue the text’s story of lookahead in *waerden*(3, 3; 9): What happens at depth  $d = 1$  when  $l \leftarrow 7$  and  $T \leftarrow 22$  (see (70)), after literal 4 has become proto true? (Assume that no double-lookahead is done.)
- **151.** [26] The dependency digraph (68) has 16 arcs, only 8 of which are captured in the subforest (69). Show that, instead of (70), we could actually list the literals  $l$  and give them offsets  $o(l)$  in such a way that  $u$  appears before  $v$  in the list and has  $o(u) > o(v)$  if and only if  $v \rightarrow u$  in (68). Thus we could capture all 16 dependencies via levels of truth.
- 152.** [22] Give an instance of 3SAT for which no free “participants” are found in step X3, yet all clauses are satisfied. Also describe an efficient way to verify satisfaction.
- 153.** [17] What’s a good way to weed out unwanted candidates in step X3, if  $C > C_{\max}$ ?
- 154.** [20] Suppose we’re looking ahead with just four candidate variables,  $\{a, b, c, d\}$ , and that they’re related by three binary clauses  $(a \vee \bar{b}) \wedge (a \vee \bar{c}) \wedge (c \vee \bar{d})$ . Find a subforest and a sequence of truth levels to facilitate lookaheads, analogous to (69) and (70).

ISTAMP  
 stamping  
 move codes  
 big clause  
 heuristic scores  
 cutoffs  
 participant  
 participants

- 155.** [32] Sketch an efficient way to construct the lookahead forest in step X4.
- 156.** [05] Why is a pure literal a special case of an autarky?
- 157.** [10] Give an example of an autarky that is not a pure literal.
- 158.** [15] If  $l$  is a pure literal, will Algorithm X discover it?
- 159.** [M17] True or false: (a)  $A$  is an autarky for  $F$  if and only if  $F|A \subseteq F$ . (b) If  $A$  is an autarky for  $F$  and  $A' \subseteq A$ , then  $A \setminus A'$  is an autarky for  $F|A'$ .
- 160.** [18] (*Black and white principle.*) Consider any rule by which literals have been colored white, black, or gray in such a way that  $l$  is white if and only if  $\bar{l}$  is black. (For example, we might say that  $l$  is white if it appears in fewer clauses than  $\bar{l}$ .)
- Suppose every clause of  $F$  that contains a white literal also contains a black literal. Prove that  $F$  is satisfiable if and only if its all-gray clauses are satisfiable.
  - Explain why this metaphor is another way to describe the notion of an autarky.
- **161.** [21] (*Black and blue principle.*) Now consider coloring literals either white, black, orange, blue, or gray, in such a way that  $l$  is white if and only if  $\bar{l}$  is black, and  $l$  is orange if and only if  $\bar{l}$  is blue. (Hence  $l$  is gray if and only if  $\bar{l}$  is gray.) Suppose further that  $F$  is a set of clauses in which every clause containing a white literal also contains either a black literal or a blue literal (or both). Let  $A = \{a_1, \dots, a_p\}$  be the black literals and let  $L = \{l_1, \dots, l_q\}$  be the blue literals. Also let  $F'$  be the set of clauses obtained by adding  $p$  additional clauses  $(\bar{l}_1 \vee \dots \vee \bar{l}_q \vee a_j)$  to  $F$ , for  $1 \leq j \leq p$ .
- Prove that  $F$  is satisfiable if and only if  $F'$  is satisfiable.
  - Restate and simplify that result in the case that  $p = 1$ .
  - Restate and simplify that result in the case that  $q = 1$ .
  - Restate and simplify that result in the case that  $p = q = 1$ . (In this special case,  $(\bar{l} \vee a)$  is called a *blocked binary clause*.)
- 162.** [21] Devise an efficient way to discover all of the (a) blocked binary clauses  $(\bar{l} \vee a)$  and (b) size-two autarkies  $A = \{a, a'\}$  of a given  $k$ SAT problem  $F$ .
- **163.** [M25] Prove that the following recursive procedure  $R(F)$  will solve any  $n$ -variable 3SAT problem  $F$  with at most  $O(\phi^n)$  executions of steps R1, R2, or R3:
- R1.** [Check easy cases.] If  $F = \emptyset$ , return true. If  $\emptyset \in F$ , return false. Otherwise let  $\{l_1, \dots, l_s\} \in F$  be a clause of minimum size  $s$ .
- R2.** [Check autarkies.] If  $s = 1$  or if  $\{l_s\}$  is an autarky, set  $F \leftarrow F|l_s$  and return to R1. Otherwise if  $\{\bar{l}_s, l_{s-1}\}$  is an autarky, set  $F \leftarrow F|\bar{l}_s, l_{s-1}$  and return to R1.
- R3.** [Recurse.] If  $R(F|l_s)$  is true, return true. Otherwise set  $F \leftarrow F|\bar{l}_s$ ,  $s \leftarrow s - 1$ , and go back to R2. ■
- 164.** [M30] Continuing exercise 163, bound the running time when  $F$  is  $k$ SAT.
- **165.** [26] Design an algorithm to find the largest positive autarky  $A$  for a given  $F$ , namely an autarky that contains only positive literals. *Hint:* Warm up by finding the largest positive autarky for the clauses  $\{12\bar{3}, 12\bar{5}, \bar{1}3\bar{4}, 13\bar{6}, 14\bar{5}, 15\bar{6}, \bar{2}3\bar{5}, 24\bar{6}, 34\bar{5}, 35\bar{6}\}$ .
- 166.** [30] Justify the operations of step X9. *Hint:* Prove that an autarky can be constructed, if  $w = 0$  after (72) has been performed.
- **167.** [21] Justify step X11 and the similar use of X12 in step X6.
- 168.** [26] Suggest a way to choose the branch literal  $l$  in step L3, based on the heuristic scores  $H(l)$  that were compiled by Algorithm X in step L2. *Hint:* Experience shows that it's good to have both  $H(l)$  and  $H(\bar{l})$  large.

lookahead forest  
 pure literal  
 autarky  
 Black and white principle  
 Black and blue principle  
 blocked binary clause  
 kSAT  
 3SAT  
 analysis of algs  
 kSAT  
 positive autarky  
 positive literals  
 autarky  
 necessary assignments

- **169.** [HM30] (T. Ahmed, O. Kullmann.) Excellent results have been obtained in some problems when the branch variable in step L3 is chosen to minimize the quantity  $\tau(H(l), H(\bar{l}))$ , where  $\tau(a, b)$  is the positive solution to  $\tau^{-a} + \tau^{-b} = 1$ . (For example,  $\tau(1, 2) = \phi \approx 1.62$  and  $\tau(\sqrt{2}, \sqrt{2}) = 2^{1/\sqrt{2}} \approx 1.63$ , so we prefer  $(1, 2)$  to  $(\sqrt{2}, \sqrt{2})$ .) Given a list of pairs of positive numbers  $(a_1, b_1), \dots, (a_s, b_s)$ , what's an efficient way to determine an index  $j$  that minimizes  $\tau(a_j, b_j)$ , without computing logarithms?

**170.** [25] (Marijn Heule, 2013.) Show that Algorithm L solves 2SAT in linear time.

**171.** [20] What is the purpose of DFAIL in Algorithm Y?

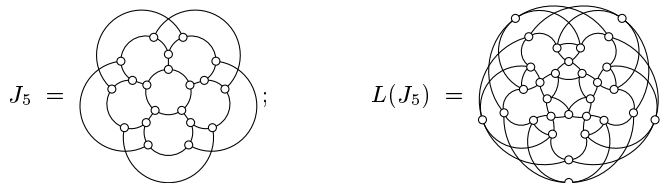
**172.** [21] Explain why '+L0[j]' appears in step Y2's formula for DT.

**173.** [40] Use an implementation of Algorithm L to experiment with random 3SAT problems such as *rand*(3, 2062, 500, 314). Examine the effects of such things as (i) disabling double lookahead; (ii) disabling "wraparound," by changing the cases  $j = S$  and  $\hat{j} = S$  in X7 and Y4 so that they simply go to X6 and Y3; (iii) disabling the lookahead forest, by letting all candidate literals have null PARENT; (iv) disabling compensation resolvents in step L9; (v) disabling "windfalls" in (72); (vi) branching on a *random* free candidate  $l$  in L3, instead of using the  $H$  scores as in exercise 168; or (vii) disabling all lookahead entirely as in "Algorithm L<sup>0</sup>."

**174.** [15] What's an easy way to accomplish (i) in the previous exercise?

**175.** [32] When Algorithm L is extended to nonternary clauses as in exercise 143, how should Algorithms X and Y also change? (Instead of using (64) and (65) to compute a heuristic for preselection, use the much simpler formula in answer 146. And instead of using  $h(u)h(v)$  in (67) to estimate the weight of a ternary clause that will be reduced to binary, consider a simulated reduced clause of size  $s \geq 2$  to have weight  $K_s \approx \gamma^{s-2}$ , where  $\gamma$  is a constant (typically 0.2).)

**176.** [M25] The "flower snark"  $J_q$  is a cubic graph with  $4q$  vertices  $t_j, u_j, v_j, w_j$ , and  $6q$  edges  $t_j - t_{j+1}, t_j - u_j, u_j - v_j, u_j - w_j, v_j - w_{j+1}, w_j - v_{j+1}$ , for  $1 \leq j \leq q$ , with subscripts treated modulo  $q$ . Here, for example, are  $J_5$  and its line graph  $L(J_5)$ :



- Give labels  $a_j, b_j, c_j, d_j, e_j$ , and  $f_j$  to the edges of  $J_q$ , for  $1 \leq j \leq q$ . (Thus  $a_j$  denotes  $t_j - t_{j+1}$  and  $b_j$  denotes  $t_j - u_j$ , etc.) What are the edges of  $L(J_q)$ ?
- Show that  $\chi(J_q) = 2$  and  $\chi(L(J_q)) = 3$  when  $q$  is even.
- Show that  $\chi(J_q) = 3$  and  $\chi(L(J_q)) = 4$  when  $q$  is odd. *Note:* Let *fsnark*( $q$ ) denote the clauses (15) and (16) that correspond to 3-coloring  $L(J_q)$ , together with  $b_{1,1} \wedge c_{1,2} \wedge d_{1,3}$  to set the colors of  $(b_1, c_1, d_1)$  to  $(1, 2, 3)$ . Also let *fsnark'*( $q$ ) be *fsnark*( $q$ ) augmented by (17). These clauses make excellent benchmark tests for SAT solvers.

**177.** [HM26] Let  $I_q$  be the number of independent sets of the flower snark line graph  $L(J_q)$ . Compute  $I_q$  for  $1 \leq q \leq 8$ , and determine the asymptotic growth rate.

- **178.** [M23] When Algorithm B is presented with the unsatisfiable clauses *fsnark*( $q$ ) of exercise 176, with  $q$  odd, its speed depends critically on the ordering of the variables.

Ahmed  
Kullmann  
 $\tau$  function  
Heule  
2SAT  
DFAIL  
random 3SAT  
*rand*  
 $\pi$   
lookahead forest  
compensation resolvents  
windfalls  
Algorithm L<sup>0</sup>  
heuristic  
preselection  
flower snark  
cubic graph  
trivalent graph, see cubic  
line graph  
chromatic number  $\chi$   
*fsnark*  
benchmark tests  
independent sets  
asymptotic



Show that the running time is  $\Theta(2^q)$  when the variables are considered in the order

$$a_{1,1} a_{1,2} a_{1,3} b_{1,1} b_{1,2} b_{1,3} c_{1,1} c_{1,2} c_{1,3} d_{1,1} d_{1,2} d_{1,3} e_{1,1} e_{1,2} e_{1,3} f_{1,1} f_{1,2} f_{1,3} a_{2,1} a_{2,2} a_{2,3} \dots;$$

but much, much more time is needed when the order is

$$a_{1,1} b_{1,1} c_{1,1} d_{1,1} e_{1,1} f_{1,1} a_{2,1} b_{2,1} c_{2,1} d_{2,1} e_{2,1} f_{2,1} \dots a_{q,1} b_{q,1} c_{q,1} d_{q,1} e_{q,1} f_{q,1} a_{1,2} b_{1,2} c_{1,2} \dots$$

**179.** [25] Show that there are exactly 4380 ways to fill the 32 cells of the 5-cube with eight 4-element subcubes. For example, one such way is to use the subcubes 000\*\*, 001\*\*, ..., 111\*\*, in the notation of 7.1.1-(29); a more interesting way is to use

$$0*0*0, \quad 1*0*0, \quad **001, \quad **110, \quad *010*, \quad *110*, \quad 0**11, \quad 1**11.$$

What does this fact tell you about the value of  $q_8$  in Fig. 40?

- **180.** [25] Explain how to use BDDs to compute the numbers  $Q_m$  that underlie Fig. 40. What is  $\max_{0 \leq m \leq 80} Q_m$ ?
- **181.** [25] Extend the idea of the previous exercise so that it is possible to determine the probability distributions  $T_m$  of Fig. 41.
- 182.** [M16] For which values of  $m$  in Fig. 41 does  $T_m$  have a constant value?
- 183.** [M30] Discuss the relation between Figs. 42 and 43.
- 184.** [M20] Why does (77) characterize the relation between  $\hat{q}_m$  and  $q_m$ ?
- 185.** [M20] Use (77) to prove the intuitively obvious fact that  $\hat{q}_m \geq q_m$ .
- 186.** [M21] Use (77) to reduce  $\sum_m \hat{q}_m$  and  $\sum_m (2m+1) \hat{q}_m$  to (78) and (79).
- 187.** [M20] Analyze random satisfiability in the case  $k = n$ : What are  $S_{n,n}$  and  $\hat{S}_{n,n}$ ?
- **188.** [HM25] Analyze random 1SAT, the case  $k = 1$ : What are  $S_{1,n}$  and  $\hat{S}_{1,n}$ ?
- 189.** [27] Apply BDD methods to random 3SAT problems on 50 variables. What is the approximate BDD size after  $m$  distinct clauses have been ANDed together, as  $m$  grows?
- 190.** [M20] Exhibit a Boolean function of 4 variables that can't be expressed in 3CNF. (No auxiliary variables are allowed: Only  $x_1, x_2, x_3$ , and  $x_4$  may appear.)
- 191.** [M25] How many Boolean functions of 4 variables *can* be expressed in 3CNF?
- **192.** [HM21] Another way to model satisfiability when there are  $N$  equally likely clauses is to study  $S(p)$ , the probability of satisfiability when each clause is independently present with probability  $p$ .
  - a) Express  $S(p)$  in terms of the numbers  $Q_m = \binom{N}{m} q_m$ .
  - b) Assign uniform random numbers in  $[0..1)$  to each clause; then at time  $t$ , for  $0 \leq t \leq N$ , consider all clauses that have been assigned a number less than  $t/N$ . (Approximately  $t$  clauses will therefore be selected, when  $N$  is large.) Show that  $\bar{S}_{k,n} = \int_0^N S_{k,n}(t/N) dt$ , the expected amount of time during which the chosen clauses remain satisfiable, is very similar to the satisfiability threshold  $S_{k,n}$  of (76).
- 193.** [HM48] Determine the satisfiability threshold (81) of random 3SAT. Is it true that  $\liminf_{n \rightarrow \infty} S_{3,n}/n = \limsup_{n \rightarrow \infty} S_{3,n}/n$ ? If so, is the limit  $\approx 4.2667$ ?
- 194.** [HM49] If  $\alpha < \liminf_{n \rightarrow \infty} S_{3,n}/n$ , is there a polynomial-time algorithm that is able to satisfy  $\lfloor \alpha n \rfloor$  random 3SAT clauses with probability  $\geq \delta$ , for some  $\delta > 0$ ?
- 195.** [HM21] (J. Franco and M. Paull, 1983.) Use the first moment principle MPR-(21) to prove that  $\lfloor (2^k \ln 2)n \rfloor$  random  $k$ SAT clauses are almost always unsatisfiable. *Hint:* Let  $X = \sum_x [x \text{ satisfies all clauses}]$ , summed over all  $2^n$  binary vectors  $x = x_1 \dots x_n$ .

subcubes  
n-cube  
BDD  
1SAT  
BDD  
3SAT-  
3CNF  
auxiliary variables  
random satisfiability  
stopping time  
Franco  
Paull  
kSAT  
first moment principle

- **196.** [HM25] (D. B. Wilson.) A clause of a satisfiability problem is “easy” if it contains one or more variables that don’t appear in any other clauses. Prove that, with probability  $1 - O(n^{-2\epsilon})$ , a  $k$ SAT problem that has  $m = \lfloor \alpha n \rfloor$  random clauses contains  $(1 - (1 - e^{-k\alpha})^k)m + O(n^{1/2+\epsilon})$  easy ones. (For example, about 0.000035  $n$  of the  $4.27n$  clauses in a random 3SAT problem near the threshold will be easy.)

**197.** [HM21] Prove that the quotient  $q(a, b, A, B, n) = \binom{(a+b)n}{an} \binom{(A+B)n}{An} / \binom{(a+b+A+B)n}{(a+A)n}$  is  $O(n^{-1/2})$  as  $n \rightarrow \infty$ , if  $a, b, A, B > 0$ .

- **198.** [HM30] Use exercises 196 and 197 to show that the phase transition in Fig. 46 is not extremely abrupt: If  $S_3(m, n) > \frac{2}{3}$  and  $S_3(m', n) < \frac{1}{3}$ , prove that  $m' = m + \Omega(\sqrt{n})$ .

**199.** [M21] Let  $p(t, m, N)$  be the probability that  $t$  specified letters each occur at least once within a random  $m$ -letter word on an  $N$ -letter alphabet.

- Prove that  $p(t, m, N) \leq m^t/N^t$ .
- Derive the exact formula  $p(t, m, N) = \sum_k \binom{t}{k} (-1)^k (N - k)^m / N^m$ .
- And  $p(t, m, N)/t! = \left\{ \begin{matrix} t \\ t \end{matrix} \right\} \binom{m}{t} / N^t - \left\{ \begin{matrix} t+1 \\ t \end{matrix} \right\} \binom{m}{t+1} / N^{t+1} + \left\{ \begin{matrix} t+2 \\ t \end{matrix} \right\} \binom{m}{t+2} / N^{t+2} - \dots$ .

- **200.** [M21] Complete the text’s proof of (84) when  $c < 1$ :

- Show that every unsatisfiable 2SAT formula contains clauses of a snare.
- Conversely, are the clauses of a snare always unsatisfiable?
- Verify the inequality (89). *Hint:* See exercise 199.

**201.** [HM29] The  $t$ -snake clauses specified by a chain  $(l_1, \dots, l_{2t-1})$  can be written  $(\bar{l}_i \vee l_{i+1})$  for  $0 \leq i < 2t$ , where  $l_0 = \bar{l}_t$  and subscripts are treated mod  $2t$ .

- Describe all ways to set two of the  $l$ ’s so that  $(\bar{x}_1 \vee x_2)$  is one of those  $2t$  clauses.
- Similarly, set three  $l$ ’s in order to obtain  $(\bar{x}_1 \vee x_2)$  and  $(\bar{x}_2 \vee x_3)$ .
- Also set three to obtain both  $(\bar{x}_0 \vee x_1)$  and  $(\bar{x}_{t-1} \vee x_t)$ ; here  $\bar{x}_0 \equiv x_t$  and  $t > 2$ .
- How can the clauses  $(\bar{x}_i \vee x_{i+1})$  for  $0 \leq i < t$  all be obtained by setting  $t$  of the  $l$ ’s?
- In general, let  $N(q, r)$  be the number of ways to choose  $r$  of the standard clauses  $(\bar{x}_i \vee x_{i+1})$ , which involve exactly  $q$  of the variables  $\{x_1, \dots, x_{2t-1}\}$ , and to set  $q$  values of  $\{l_1, \dots, l_{2t-1}\}$  in order to obtain the  $r$  chosen clauses. Evaluate  $N(2, 1)$ .
- Similarly, evaluate  $N(3, 2)$ ,  $N(t, t)$ , and  $N(2t - 1, 2t)$ .
- Show that the probability  $p_r$  in (95) is  $\leq \sum_q N(q, r) / (2^q n^2)$ .
- Therefore the upper bound (96) is valid.

**202.** [HM21] This exercise amplifies the text’s proof of Theorem C when  $c > 1$ .

- Explain the right-hand side of Eq. (93).
- Why does (97) follow from (95), (96), and the stated choices of  $t$  and  $m$ ?

- **203.** [HM33] (K. Xu and W. Li, 2000.) Beginning with the  $n$  graph-coloring clauses (15), and optionally the  $n \binom{d}{2}$  exclusion clauses (17), consider using randomly generated binary clauses instead of (16). There are  $mq$  random binary clauses, obtained as  $m$  independent sets of  $q$  clauses each, where every such set is selected by choosing distinct vertices  $u$  and  $v$ , then choosing  $q$  distinct binary clauses  $(\bar{u}_i \vee \bar{v}_j)$  for  $1 \leq i, j \leq d$ . (The number of different possible sequences of random clauses is therefore exactly  $\binom{n}{2} \binom{d^2}{q}^m$  and each sequence is equally likely.) This method of clause generation is known as “Model RB”; it generalizes random 2SAT, which is the case  $d = 2$  and  $q = 1$ .

Suppose  $d = n^\alpha$  and  $q = pd^2$ , where we require  $\frac{1}{2} < \alpha < 1$  and  $0 \leq p \leq \frac{1}{2}$ . Also let  $m = rn \ln d$ . For this range of the parameters, we will prove that there is a sharp threshold of satisfiability: The clauses are unsatisfiable q.s., as  $n \rightarrow \infty$ , if  $r \ln(1 - p) + 1 < 0$ ; but they are satisfiable a.s. if  $r \ln(1 - p) + 1 > 0$ .

Wilson  
easy clauses  
bystanders, see easy clauses  
binomial coefficients  
phase transition  
sharp threshold  
random word  
Stirling subset numbers  
2SAT  
snare  
 $t$ -snake  
Xu  
Li  
exclusion clauses  
at-most-one  
Model RB  
random 2SAT  
threshold of satisfiability  
q.s.  
a.s.

Let  $X(j_1, \dots, j_n) = [$ all clauses are satisfied when each  $i$ th variable  $v$  has  $v_{j_i} = 1]$ ; here  $1 \leq j_1, \dots, j_n \leq d$ . Also let  $X = \sum_{1 \leq j_1, \dots, j_n \leq d} X(j_1, \dots, j_n)$ . Then  $X = 0$  if and only if the clauses are unsatisfiable.

- Use the first moment principle to prove that  $X = 0$  q.s. when  $r \ln(1-p) + 1 < 0$ .
- Find a formula for  $p_s = \Pr(X(j_1, \dots, j_n) = 1 \mid X(1, \dots, 1) = 1)$ , given that exactly  $s$  of the colors  $\{j_1, \dots, j_n\}$  are equal to 1.
- Use (b) and the conditional expectation inequality MPR-(24) to prove that  $X > 0$  a.s. if

$$\sum_{s=0}^n \binom{n}{s} \left(\frac{1}{d}\right)^s \left(1 - \frac{1}{d}\right)^{n-s} \left(1 + \frac{p}{1-p} \frac{s^2}{n^2}\right)^m \rightarrow 1 \quad \text{as } n \rightarrow \infty.$$

- Letting  $t_s$  denote the term for  $s$  in that sum, prove that  $\sum_{s=0}^{3n/d} t_s \approx 1$ .
- Suppose  $r \ln(1-p) + 1 = \epsilon > 0$ , where  $\epsilon$  is small. Show that the terms  $t_s$  first increase, then decrease, then increase, then decrease again, as  $s$  grows from 0 to  $n$ . *Hint*: Consider the ratio  $x_s = t_{s+1}/t_s$ .
- Finally, prove that  $t_s$  is exponentially small for  $3n/d \leq s \leq n$ .

► **204.** [28] Figure 46 might suggest that 3SAT problems on  $n$  variables are always easy when there are fewer than  $2n$  clauses. We shall prove, however, that *any* set of  $m$  ternary clauses on  $n$  variables can be transformed mechanically into another set of ternary clauses on  $N = O(m)$  variables in which no variable occurs more than four times. The transformed problem is satisfiable if and only if the original problem was; thus it isn't any simpler, although (with at most  $4N$  literals) it has at most  $\frac{4}{3}N$  clauses.

- First replace the original  $m$  clauses by  $m$  new clauses  $(X_1 \vee X_2 \vee X_3), \dots, (X_{3m-2} \vee X_{3m-1} \vee X_{3m})$ , on  $3m$  new variables, and show how to add  $3m$  clauses of size 2 so that the resulting  $4m$  clauses have exactly as many solutions as the original.
- Construct 16 unsatisfiable ternary clauses on 15 variables, where each variable occurs at most four times. *Hint*: If  $F$  and  $F'$  are sets of clauses, let  $F \sqcup F'$  stand for any other set obtained from  $F \cup F'$  by replacing one or more clauses  $C$  of  $F$  by  $x \cup C$  and one or more clauses  $C'$  of  $F'$  by  $\bar{x} \cup C'$ , where  $x$  is a new variable; then  $F \sqcup F'$  is unsatisfiable whenever  $F$  and  $F'$  are both unsatisfiable. For example, if  $F = \{\epsilon\}$  and  $F' = \{1, \bar{1}\}$ , then  $F \sqcup F'$  is either  $\{2, 1\bar{2}, \bar{1}\bar{2}\}$  or  $\{2, 1, \bar{1}\bar{2}\}$  or  $\{2, 1\bar{2}, \bar{1}\}$ .
- Remove one of the clauses from solution (b) and find all solutions of the 15 clauses that remain. (At least three of the variables will have forced values.)
- Use (a), (b), and (c) to prove the  $N$ -variable result claimed above.

**205.** [26] Construct an unsatisfiable 4SAT problem in which every variable occurs at most 5 times. *Hint*: Use the  $\sqcup$  operation as in the previous exercise.

**206.** [M22] A set of clauses is *minimally unsatisfiable* if it is unsatisfiable, yet becomes satisfiable if any clause is deleted. Show that, if  $F$  and  $F'$  have no variables in common, then  $F \sqcup F'$  is minimally unsatisfiable if and only if  $F$  and  $F'$  are minimally unsatisfiable.

**207.** [25] Each of the literals  $\{1, \bar{1}, 2, \bar{2}, 3, \bar{3}, 4, \bar{4}\}$  occurs exactly thrice in the eight unsatisfiable clauses (6). Construct an unsatisfiable 3SAT problem with 15 variables in which each of the 30 literals occurs exactly *twice*. *Hint*: Consider  $\{\bar{1}\bar{2}, \bar{2}\bar{3}, \bar{3}\bar{1}, 123, \bar{1}\bar{2}\bar{3}\}$ .

**208.** [25] Via exercises 204(a) and 207, show that any 3SAT problem can be transformed into an equivalent set of ternary clauses where every literal occurs just twice.

**209.** [25] (C. A. Tovey.) Prove that every  $k$ SAT formula in which no variable occurs more than  $k$  times is satisfiable. (Thus the limits on occurrences in exercises 204–208 cannot be lowered, when  $k = 3$  and  $k = 4$ .) *Hint*: Use the theory of bipartite matching.

first moment principle  
conditional expectation inequality  
4SAT  
minimally unsatisfiable  
clauses per literal  
Tovey  
bipartite matching  
matching

**210.** [M36] But the result in the previous exercise can be improved when  $k$  is large. Use the Local Lemma to show that every 7SAT problem with at most 13 occurrences of each variable is satisfiable.

**211.** [30] (R. W. Irving and M. Jerrum, 1994.) Use exercise 208 to reduce 3SAT to the problem of list coloring a grid graph of the form  $K_N \square K_3$ . (Hence the latter problem, which is also called *latin rectangle construction*, is NP-complete.)

**212.** [32] Continuing the previous exercise, we shall reduce grid list coloring to another interesting problem called *partial latin square construction*. Given three  $n \times n$  binary matrices  $(r_{ik}), (c_{jk}), (p_{ij})$ , the task is to construct an  $n \times n$  array  $(X_{ij})$  such that  $X_{ij}$  is blank when  $p_{ij} = 0$ , otherwise  $X_{ij} = k$  for some  $k$  with  $r_{ik} = c_{jk} = 1$ ; furthermore the nonblank entries must be distinct in each row and column.

- Show that this problem is symmetrical in all three coordinates: It's equivalent to constructing a binary  $n \times n \times n$  tensor  $(x_{ijk})$  such that  $x_{*jk} = c_{jk}$ ,  $x_{i*k} = r_{ik}$ , and  $x_{ij*} = p_{ij}$ , for  $1 \leq i, j, k \leq n$ , where '\*' denotes summing an index from 1 to  $n$ . (Therefore it is also known as the *binary  $n \times n \times n$  contingency problem*, given  $n^2$  row sums,  $n^2$  column sums, and  $n^2$  pile sums.)
- A necessary condition for solution is that  $c_{*k} = r_{*k}$ ,  $c_{j*} = p_{*j}$ , and  $r_{i*} = p_{i*}$ . Exhibit a small example where this condition is not sufficient.
- If  $M < N$ , reduce  $K_M \square K_N$  list coloring to the problem of  $K_N \square K_N$  list coloring.
- Finally, explain how to reduce  $K_N \square K_N$  list coloring to the problem of constructing an  $n \times n$  partial latin square, where  $n = N + \sum_{I,J} |L(I, J)|$ . *Hint:* Instead of considering integers  $1 \leq i, j, k \leq n$ , let  $i, j, k$  range over a *set* of  $n$  elements. Define  $p_{ij} = 0$  for most values of  $i$  and  $j$ ; also make  $r_{ik} = c_{ik}$  for all  $i$  and  $k$ .

► **213.** [M20] Experience with the analyses of sorting algorithms in Chapter 5 suggests that random satisfiability problems might be modeled nicely if we assume that, in each of  $m$  independent clauses, the literals  $x_j$  and  $\bar{x}_j$  occur with respective probabilities  $p$  and  $q$ , independently for  $1 \leq j \leq n$ , where  $p + q \leq 1$ . Why is this *not* an interesting model as  $n \rightarrow \infty$ , when  $p$  and  $q$  are constant? *Hint:* What is the probability that  $x_1 \dots x_n = b_1 \dots b_n$  satisfies all of the clauses, when  $b_1 \dots b_n$  is a given binary vector?

**214.** [HM38] Although the random model in the preceding exercise doesn't teach us how to solve SAT problems, it does lead to interesting mathematics: Let  $0 < p < 1$  and consider the recurrence

$$T_0 = 0; \quad T_n = n + 2 \sum_{k=0}^{n-1} \binom{n}{k} p^k (1-p)^{n-k} T_k, \quad \text{for } n > 0.$$

- Find a functional relation satisfied by  $T(z) = \sum_{n=0}^{\infty} T_n z^n / n!$ .
- Deduce that we have  $T(z) = z e^z \sum_{m=0}^{\infty} (2p)^m \prod_{k=0}^{m-1} (1 - e^{-p^k(1-p)z})$ .
- Hence, if  $p \neq 1/2$ , we can use Mellin transforms (as in the derivation of 5.2.2-(50)) to show that  $T_n = C_p n^\alpha (1 + \delta(n) + O(1/n)) + n/(1-2p)$ , where  $\alpha = 1/\lg(1/p)$ ,  $C_p$  is a constant, and  $\delta$  is a small "wobble" with  $\delta(n) = \delta(pn)$ .

► **215.** [HM23] What is the expected profile of the search tree when a simple backtrack procedure is used to find all solutions to a random 3SAT problem with  $m$  independent clauses on  $n$  variables? (There is a node on level  $l$  for every partial solution  $x_1 \dots x_l$  that doesn't contradict any of the clauses.) Compute these values when  $m = 200$  and  $n = 50$ . Also estimate the total tree size when  $m = \alpha n$ , for fixed  $\alpha$  as  $n \rightarrow \infty$ .

**216.** [HM38] (P. W. Purdom, Jr., and C. A. Brown.) Extend the previous exercise to a more sophisticated kind of backtracking, where all choices forced by unit clauses are

Local Lemma  
7SAT  
Irving  
Jerrum  
list coloring  
grid graph  
latin rectangle construction  
NP-complete  
grid list coloring  
partial latin square construction  
binary matrices  
tensor  
row sums  
column sums  
pile sums  
contingency tables, 3D  
random satisfiability  
recurrence  
generating functions  
Mellin transforms  
asymptotics  
wobble  
profile  
backtrack  
asymptotics  
Purdom  
Brown

pursued before two-way branching is done. (The “pure literal rule” is not exploited, however, because it doesn’t find all solutions.) Prove that the expected tree size is greatly reduced when  $m = 200$  and  $n = 50$ . (An upper bound is sufficient.)

**217.** [20] True or false: If  $A$  and  $B$  are arbitrary clauses that are simultaneously satisfiable, and if  $l$  is any literal, then the clause  $C = (A \cup B) \setminus \{l, \bar{l}\}$  is also satisfiable. (We’re thinking here of  $A$ ,  $B$ , and  $C$  as *sets* of literals, not as disjunctions of literals.)

**218.** [20] Express the formula  $(x \vee A) \wedge (\bar{x} \vee B)$  in terms of the ternary operator  $u? v: w$ .

- **219.** [M20] Formulate a general definition of the resolution operator  $C = C' \diamond C''$  that (i) agrees with the text’s definition when  $C' = x \vee A'$  and  $C'' = \bar{x} \vee A''$ ; (ii) applies to arbitrary clauses  $C'$  and  $C''$ ; (iii) has the property that  $C' \wedge C''$  implies  $C' \diamond C''$ .

**220.** [M24] We say that clause  $C$  *subsumes* clause  $C'$ , written  $C \subseteq C'$ , if  $C' = \varnothing$  or if  $C' \neq \varnothing$  and every literal of  $C$  appears in  $C'$ .

- True or false:  $C \subseteq C'$  and  $C' \subseteq C''$  implies  $C \subseteq C''$ .
- True or false:  $(C \vee \alpha) \diamond (C' \vee \alpha') \subseteq (C \diamond C') \vee \alpha \vee \alpha'$ , with  $\diamond$  as in exercise 219.
- True or false:  $C' \subseteq C''$  implies  $C \diamond C' \subseteq C \diamond C''$ .
- The notation  $C_1, \dots, C_m \vdash C$  means that a resolution chain  $C_1, \dots, C_{m+r}$  exists with  $C_{m+r} \subseteq C$ , for some  $r \geq 0$ . Show that we might have  $C_1, \dots, C_m \vdash C$  even though  $C$  cannot be obtained from  $\{C_1, \dots, C_m\}$  by successive resolutions (104).
- Prove that if  $C_1 \subseteq C'_1, \dots, C_m \subseteq C'_m$ , and  $C'_1, \dots, C'_m \vdash C$ , then  $C_1, \dots, C_m \vdash C$ .
- Furthermore  $C_1, \dots, C_m \vdash C$  implies  $C_1 \vee \alpha_1, \dots, C_m \vee \alpha_m \vdash C \vee \alpha_1 \vee \dots \vee \alpha_m$ .

**221.** [16] Draw the search tree analogous to Fig. 38 that is implicitly traversed when Algorithm A is applied to the unsatisfiable clauses  $\{12, 2, 2\}$ . Explain why it does *not* correspond to a resolution refutation that is analogous to Fig. 48.

**222.** [M30] (Oliver Kullmann, 2000.) Prove that, for every clause  $C$  in a satisfiability problem  $F$ , there is an autarky satisfying  $C$  if and only if  $C$  cannot be used as the label of a source vertex in any resolution refutation of  $F$ .

**223.** [HM40] Step X9 deduces a binary clause that cannot be derived by resolution (see exercise 166). Prove that, nevertheless, the running time of Algorithm L on unsatisfiable input will never be less than the length of a shortest treelike refutation.

**224.** [M20] Given a resolution tree that refutes the axioms  $F | \bar{x}$ , show how to construct a resolution tree of the same size that either refutes the axioms  $F$  or derives the clause  $\{x\}$  from  $F$  without resolving on the variable  $x$ .

- **225.** [M31] (G. S. Tseytin, 1966.) If  $T$  is any resolution tree that refutes a set of axioms  $F$ , show how to convert it to a *regular* resolution tree  $T_r$  that refutes  $F$ , where  $T_r$  is no larger than  $T$ .

**226.** [M20] If  $\alpha$  is a node in a refutation tree, let  $C(\alpha)$  be its label, and let  $\|\alpha\|$  denote the number of leaves in its subtree. Show that, given a refutation tree with  $N$  leaves, the Prover can find a node with  $\|\alpha\| \leq N/2^s$  for which the current assignment falsifies  $C(\alpha)$ , whenever the Delayer has scored  $s$  points in the Prover–Delayer game.

**227.** [M27] Given an extended binary tree, exercise 7.2.1.6–124 explains how to label each node with its Horton–Strahler number. For example, the nodes at depth 2 in Fig. 48 are labeled 1, because their children have the labels 1 and 0; the root is labeled 3.

Prove that the maximum score that the Delayer can guarantee, when playing the Prover–Delayer game for a set of unsatisfiable clauses  $F$ , is equal to the minimum possible Horton–Strahler root label in a tree refutation of  $F$ .

pure literal  
if-then-else operator  
Notation  $C' \diamond C''$   
Notation  $C \subseteq C'$   
Notation  $F \vdash C$   
resolution chain  
search tree  
resolution refutation  
Kullmann  
autarky  
resolution tree  
Tseytin  
regular resolution  
treelike resolution  
refutation tree  
Prover–Delayer game  
Horton–Strahler number  
Strahler

- ▶ **228.** [M21] Stålmarck's refutation of (99)–(101) actually obtains  $\epsilon$  without using all of the axioms! Show that only about  $1/3$  of those clauses are sufficient for unsatisfiability.
- ▶ **229.** [M21] Continuing exercise 228, prove also that the set of clauses (99), (100'), (101) is unsatisfiable, where (100') denotes (100) restricted to the cases  $i \leq k$  and  $j < k$ .
- 230.** [M22] Show that the clauses with  $i \neq j$  in the previous exercise form a *minimal* unsatisfiable set: Removing any one of them leaves a satisfiable remainder.
- 231.** [M30] (Sam Buss.) Refute the clauses of exercise 229 with a resolution chain of length  $O(m^3)$ . *Hint:* Derive the clauses  $G_{ij} = (x_{ij} \vee x_{i(j+1)} \vee \dots \vee x_{im})$  for  $1 \leq i \leq j \leq m$ .
- ▶ **232.** [M28] Prove that the clauses *fsnark*( $q$ ) of exercise 176 can be refuted by treelike resolution in  $O(q^6)$  steps.
- 233.** [16] Explain why (105) satisfies (104), by exhibiting  $j(i)$  and  $k(i)$  for  $9 \leq i \leq 22$ .
- 234.** [20] Show that the Delayer can score at least  $m$  points against any Prover who tries to refute the pigeonhole clauses (106) and (107).
- ▶ **235.** [30] Refute those pigeonhole clauses with a chain of length  $m(m+3)2^{m-2}$ .
- 236.** [48] Is the chain in the previous exercise as short as possible?
- ▶ **237.** [28] Show that a polynomial number of steps suffice to refute the pigeonhole clauses (106), (107), if the *extended resolution* trick is used to append new clauses.
- 238.** [HM21] Complete the proof of Lemma B. *Hint:* Make  $r \leq \rho^{-b}$  when  $W = b$ .
- ▶ **239.** [M21] What clauses  $\alpha_0$  on  $n$  variables make  $\|\alpha_0 \vdash \epsilon\|$  as large as possible?
- ▶ **240.** [HM23] Choose integers  $f_{ij} \in \{1, \dots, m\}$  uniformly at random, for  $1 \leq i \leq 5$  and  $0 \leq j \leq m$ , and let  $G_0$  be the bipartite graph with edges  $a_j \text{ --- } b_k$  if and only if  $k \in \{f_{1j}, \dots, f_{5j}\}$ . Show that  $\Pr(G_0 \text{ satisfies the strong expansion condition (108)}) \geq 1/2$ .
- 241.** [20] Prove that any set of at most  $m/3000$  pigeons can be matched to distinct holes, under the restricted pigeonhole constraints  $G_0$  of Theorem B.
- 242.** [M20] The pigeonhole axioms (106) and (107) are equivalent to the clauses (15) and (16) that arise if we try to color the complete graph  $K_{m+1}$  with  $m$  colors.  
Suppose we include further axioms corresponding to (17), namely

$$(\bar{x}_{jk} \vee \bar{x}_{jk'}), \quad \text{for } 0 \leq j \leq m \text{ and } 1 \leq k < k' \leq m.$$

Does Theorem B still hold, or do these additional axioms decrease the refutation width?

**243.** [HM31] (E. Ben-Sasson and A. Wigderson.) Let  $F$  be a set of  $\lfloor \alpha n \rfloor$  random 3SAT clauses on  $n$  variables, where  $\alpha > 1/e$  is a given constant. For any clause  $C$  on those variables, define  $\mu(C) = \min\{|F'| \mid F' \subseteq F \text{ and } F' \vdash C\}$ . Also let  $V(F')$  denote the variables that occur in a given family of clauses  $F'$ .

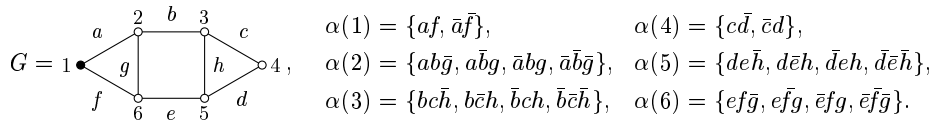
- a) Prove that  $|V(F')| \geq |F'|$  a.s., when  $F' \subseteq F$  and  $|F'| \leq n/(2\alpha e^2)$ .
- b) Therefore either  $F$  is satisfiable or  $\mu(\epsilon) > n/(2\alpha e^2)$ , a.s.
- c) Let  $n' = n/(1000000\alpha^4)$ , and assume that  $n' \geq 2$ . Prove that  $2|V(F')| - 3|F'| \geq n'/4$  q.s., when  $F' \subseteq F$  and  $n'/2 \leq |F'| < n'$ .
- d) Consequently either  $F$  is satisfiable or  $w(F \vdash \epsilon) \geq n'/4$ , a.s.

**244.** [M20] If  $A$  is a set of variables, let  $[A]^0$  or  $[A]^1$  stand for the set of all clauses that can be formed from  $A$  with an even or odd number of negative literals, respectively; each clause should involve all of the variables. (For example,  $\{\{1, 2, 3\}\}^1 = \{\bar{1}\bar{2}\bar{3}, \bar{1}\bar{2}3, \bar{1}2\bar{3}, \bar{1}23\}$ .) If  $A$  and  $B$  are disjoint, express  $[A \cup B]^0$  in terms of the sets  $[A]^0, [A]^1, [B]^0, [B]^1$ .

Stålmarck  
anti-maximal-element clauses  
*minimal* unsatisfiable set  
Buss  
*fsnark*  
treelike resolution  
flower snark  
Delayer  
pigeonhole  
pigeonhole  
color  
complete graph  
exclusion clauses  
Ben-Sasson  
Wigderson  
random 3SAT  
3SAT  
notation:  $\mu(C)$   
notation  $F \vdash C$   
a.s.: almost surely  
q.s.: quite surely

- **245.** [M27] Let  $G$  be a connected graph whose vertices  $v \in V$  have each been labeled 0 or 1, where the sum of all labels is odd. We will construct clauses on the set of variables  $e_{uv}$ , one for each edge  $u - v$  in  $G$ . The axioms are  $\alpha(v) = [E(v)]^{l(v) \oplus 1}$  for each  $v \in V$  (see exercise 244), where  $E(v) = \{e_{uv} \mid u - v\}$  and  $l(v)$  is the label of  $v$ .  
 For example, vertex 1 of the graph below is shown as a black dot in order to indicate that  $l(1) = 1$ , while the other vertices appear as white dots and are labeled  $l(2) = \dots = l(6) = 0$ . The graph and its axioms are

graph-based axioms  
 cubic  
 Ramanujan graph  
 Tseytin  
 extended resolution  
 variable elimination  
 quantified formula  
 CNF  
 Cook  
 Method IA  
 clause learning



Notice that, when  $v$  has  $d > 0$  neighbors in  $G$ , the set  $\alpha(v)$  consists of  $2^{d-1}$  clauses of size  $d$ . Furthermore, the axioms of  $\alpha(v)$  are all satisfied if and only if

$$\bigoplus_{e_{uv} \in E(v)} e_{uv} = l(v).$$

If we sum this equation over all vertices  $v$ , mod 2, we get 0 on the left, because each edge  $e_{uv}$  occurs exactly twice (once in  $E(u)$  and once in  $E(v)$ ). But we get 1 on the right. Therefore the clauses  $\alpha(G) = \bigcup_v \alpha(v)$  are unsatisfiable.

- The axioms  $\alpha(G) \mid b$  and  $\alpha(G) \mid \bar{b}$  in this example turn out to be  $\alpha(G')$  and  $\alpha(G'')$ , where  $G' =$  and  $G'' =$  . Explain what happens in general.
  - Let  $\mu(C) = \min\{|V'| \mid V' \subseteq V \text{ and } \bigcup_{v \in V'} \alpha(v) \vdash C\}$ , for every clause  $C$  involving the variables  $e_{uv}$ . Show that  $\mu(C) = 1$  for every axiom  $C \in \alpha(G)$ . What is  $\mu(\epsilon)$ ?
  - If  $V' \subseteq V$ , let  $\partial V' = \{e_{uv} \mid u \in V' \text{ and } v \notin V'\}$ . Prove that, if  $|V'| = \mu(C)$ , every variable of  $\partial V'$  appears in  $C$ .
  - A nonbipartite cubic Ramanujan graph  $G$  on  $m$  vertices  $V$  has three edges  $v - v\rho$ ,  $v - v\sigma$ ,  $v - v\tau$  touching each vertex, where  $\rho$ ,  $\sigma$ , and  $\tau$  are permutations with the following properties: (i)  $\rho = \rho^-$  and  $\tau = \sigma^-$ ; (ii)  $G$  is connected; (iii) If  $V'$  is any subset of  $s$  vertices, and if there are  $t$  edges between  $V'$  and  $V \setminus V'$ , then we have  $s/(s+t) \leq (s/n+8)/9$ . Prove that  $w(\alpha(G) \vdash \epsilon) > m/78$ .
- **246.** [M28] (G. S. Tseytin.) Given a labeled graph  $G$  with  $m$  edges,  $n$  vertices, and  $N$  unsatisfiable clauses  $\alpha(G)$  as in the previous exercise, explain how to refute those clauses with  $O(mn + N)$  steps of extended resolution.

**247.** [I8] Apply variable elimination to just five of the six clauses (112), omitting '12'.

**248.** [M20] Formally speaking, SAT is the problem of evaluating the quantified formula

$$\exists x_1 \dots \exists x_{n-1} \exists x_n F(x_1, \dots, x_{n-1}, x_n),$$

where  $F$  is a Boolean function given in CNF as a conjunction of clauses. Explain how to transform the CNF for  $F$  into the CNF for  $F'$  in the reduced problem

$$\exists x_1 \dots \exists x_{n-1} F'(x_1, \dots, x_{n-1}), \quad F'(x_1, \dots, x_{n-1}) = F(x_1, \dots, x_{n-1}, 0) \vee F(x_1, \dots, x_{n-1}, 1).$$

**249.** [I8] Apply Algorithm I to (112) using Cook's Method IA.

**250.** [25] Since the clauses  $R'$  in (7) are satisfiable, Algorithm I might discover a solution without ever reaching step I4. Try, however, to make the choices in steps I2, I3, and I4 so that the algorithm takes as long as possible to discover a solution.

- **251.** [30] Show that Algorithm I can prove the unsatisfiability of the anti-maximal-element clauses (99)–(101) by making  $O(m^3)$  resolutions, if suitably clairvoyant choices are made in steps I2, I3, and I4.
- 252.** [M26] Can the unsatisfiability of (99)–(101) be proved in polynomial time by repeatedly performing variable elimination and subsumption?
- **253.** [18] What are the next two clauses learned if decision ‘5’ follows next after (114)?
- 254.** [16] Given the binary clauses  $\{12, \bar{1}3, 2\bar{3}, \bar{2}\bar{4}, \bar{3}4\}$ , what clause will a CDCL solver learn first if it begins by deciding that 1 is true?
- **255.** [20] Construct a satisfiability problem with ternary clauses, for which a CDCL solver that is started with decision literals ‘1’, ‘2’, ‘3’ on levels 1, 2, and 3 will learn the clause ‘45’ after a conflict on level 3.
- 256.** [20] How might the clause ‘\*\*’ in Table 3 have been easily learned?
- **257.** [30] (Niklas Sörensson.) A literal  $\bar{l}$  is said to be *redundant*, with respect to a given clause  $c$  and the current trail, if  $l$  is in the trail and either (i)  $l$  is defined at level 0, or (ii)  $l$  is not a decision literal and every false literal in  $l$ ’s reason is either in  $c$  or (recursively) redundant. (This definition is stronger than the special cases by which (115) reduces to (116), because  $\bar{l}$  itself needn’t belong to  $c$ .) If, for example,  $c = (\bar{l} \vee \bar{b}_1 \vee \bar{b}_2 \vee \bar{b}_3 \vee \bar{b}_4)$ , let the reason for  $b_4$  be  $(b_4 \vee \bar{b}_1 \vee \bar{a}_1)$ , where the reason for  $a_1$  is  $(a_1 \vee \bar{b}_2 \vee \bar{a}_2)$  and the reason for  $a_2$  is  $(a_2 \vee \bar{b}_1 \vee \bar{b}_3)$ . Then  $\bar{b}_4$  is redundant, because  $\bar{a}_2$  and  $\bar{a}_1$  are redundant.
- Suppose  $c = (\bar{l} \vee \bar{b}_1 \vee \dots \vee \bar{b}_r)$  is a newly learned clause. Prove that if  $\bar{b}_j \in c$  is redundant, some other  $\bar{b}_i \in c$  became false on the same level of the trail as  $\bar{b}_j$  did.
  - Devise an efficient algorithm that discovers all of the redundant literals  $\bar{b}_i$  in a given newly learned clause  $c = (\bar{l} \vee \bar{b}_1 \vee \dots \vee \bar{b}_r)$ . *Hint:* Use stamps.
- 258.** [21] A non-decision literal  $l$  in Algorithm C’s trail always has a reason  $R_l = (l_0 \vee l_1 \vee \dots \vee l_{k-1})$ , where  $l_0 = l$  and  $\bar{l}_1, \dots, \bar{l}_{k-1}$  precede  $l$  in the trail. Furthermore, the algorithm discovered this clause while looking at the watch list of  $l_1$ . True or false:  $\bar{l}_2, \dots, \bar{l}_{k-1}$  precede  $\bar{l}_1$  in the trail. *Hint:* Consider Table 3 and its sequel.
- 259.** [M20] Can  $\text{ACT}(j)$  exceed  $\text{ACT}(k)$  for values of  $\rho$  near 0 or 1, but not for *all*  $\rho$ ?
- 260.** [18] Describe in detail step C1’s setting-up of MEM, the watch lists, and the trail.
- 261.** [21] The main loop of Algorithm C is the unit-propagation process of steps C3 and C4. Describe the low-level details of link adjustment, etc., to be done in those steps.
- 262.** [20] What low-level operations underlie changes to the heap in steps C6–C8?
- 263.** [21] Write out the gory details by which step C7 constructs a new clause and step C9 puts it into the data structures of Algorithm C.
- 264.** [20] Suggest a way by which Algorithm C could indicate progress by displaying “move codes” analogous to those of Algorithms A, B, D, and L. (See exercise 142.)
- 265.** [21] Describe several circumstances in which the watched literals  $l_0$  and/or  $l_1$  of a clause  $c$  actually become false during the execution of Algorithm C.
- 266.** [20] In order to keep from getting into a rut, CDCL solvers are often designed to make decisions at random, with a small probability  $p$  (say  $p = .02$ ), instead of always choosing a variable of maximum activity. How would this policy change step C6?
- **267.** [25] Instances of SAT often contain numerous binary clauses, which are handled efficiently by the unit-propagation loop (62) of Algorithm L but not by the corresponding loop in step C3 of Algorithm C. (The technique of watched literals is great for long

anti-maximal-element  
variable elimination  
subsumption  
CDCL solver  
Sörensson  
redundant  
stamps  
activity scores  
damping factor  
unit-propagation  
move codes  
watched literals  
random decisions  
binary clauses  
unit-propagation  
watched literals



clauses, but it is comparatively cumbersome for short ones.) What additional data structures will speed up Algorithm C's inner loop, when binary clauses are abundant?

**268.** [21] When Algorithm C makes a literal false at level 0 of the trail, we can remove it from all of the clauses. Such updating might take a long time, if we did it “eagerly”; but there's a lazy way out: We can delete a permanently false literal if we happen to encounter it in step C3 while looking for a new literal to watch (see exercise 261).

Explain how to adapt the MEM data structure conventions so that such deletions can be done *in situ*, without copying clauses from one location into another.

**269.** [23] Suppose Algorithm C reaches a conflict at level  $d$  of the trail, after having chosen the decision literals  $u_1, u_2, \dots, u_d$ . Then the “trivial clause”  $(\bar{l}' \vee \bar{u}_1 \vee \dots \vee \bar{u}_{d'})$  must be true if the given clauses are satisfiable, where  $l'$  and  $d'$  are defined in step C7.

- Show that, if we start with the clause  $(\bar{l}' \vee \bar{b}_1 \vee \dots \vee \bar{b}_r)$  that is obtained in step C7 and then resolve it somehow with zero or more known clauses, we can always reach a clause that subsumes the trivial clause.
- Sometimes, as in (115), the clause that is slated to be learned in step C9 is much longer than the trivial clause. Construct an example in which  $d = 3$ ,  $d' = 1$ , and  $r = 10$ , yet none of  $\bar{b}_1, \dots, \bar{b}_r$  are redundant in the sense of exercise 257.
- Suggest a way to improve Algorithm C accordingly.

**270.** [25] (*On-the-fly subsumption.*) The intermediate clauses that arise in step C7, immediately after resolving with a reason  $R_l$ , occasionally turn out to be equal to the shorter clause  $R_l \setminus l$ . In such cases we have an opportunity to *strengthen* that clause by deleting  $l$  from it, thus making it potentially more useful in the future.

- Construct an example where two clauses can each be subsumed in this way while resolving a single conflict. The subsumed clauses should both contain two literals assigned at the current level in the trail, as well as one literal from a lower level.
- Show that it's easy to recognize such opportunities, and to strengthen such clauses efficiently, by modifying the steps of answer 263.

- **271.** [25] The sequence of learned clauses  $C_1, C_2, \dots$  often includes cases where  $C_i$  subsumes its immediate predecessor,  $C_{i-1}$ . In such cases we might as well *discard*  $C_{i-1}$ , which appears at the very end of MEM, and store  $C_i$  in its place, unless  $C_{i-1}$  is still in use as a reason for some literal on the trail. (For example, more than 8,600 of the 52,000 clauses typically learned from *waerden*(3, 10; 97) by Algorithm C can be discarded in this way. Such discards are different from the on-the-fly subsumptions considered in exercise 270, because the subsumed  $C_{i-1}$  includes only one literal from its original conflict level; furthermore, learned clauses have usually been significantly simplified by the procedure of exercise 257, unless they're trivial.)

Design an efficient way to discover when  $C_{i-1}$  can be safely discarded.

**272.** [30] Experiment with the following idea: The clauses of *waerden*( $j, k; n$ ) are symmetrical under reflection, in the sense that they remain unchanged overall if we replace  $x_k$  by  $x_k^R = x_{n+1-k}$  for  $1 \leq k \leq n$ . Therefore, whenever Algorithm C learns a clause  $C = (\bar{l}' \vee \bar{b}_1 \vee \dots \vee \bar{b}_r)$ , it is also entitled to learn the reflected clause  $C^R = (\bar{l}'^R \vee \bar{b}_1^R \vee \dots \vee \bar{b}_r^R)$ .

**273.** [27] A clause  $C$  that is learned from *waerden*( $j, k; n$ ) is valid also with respect to *waerden*( $j, k; n'$ ) when  $n' > n$ ; and so are the clauses  $C + i$  that are obtained by adding  $i$  to each literal of  $C$ , for  $1 \leq i \leq n' - n$ . For example, the fact that '35' follows from *waerden*(3, 3; 7) allows us to add the clauses 35, 46, 57 to *waerden*(3, 3; 9).

- Exploit this idea to speed up the calculation of van der Waerden numbers.

level 0  
eagerly  
lazy  
MEM  
in situ  
trivial clause  
redundant  
On-the-fly subsumption  
subsumption  
strengthen  
learned clauses, sequence of  
subsumes  
discard  
MEM  
*waerden*  
on-the-fly subsumptions  
*waerden*  
symmetrical  
reflection  
van der Waerden numbers

b) Explain how to apply it also to bounded model checking.

**274.** [35] Algorithm C sets the “reason” for a literal  $l$  as soon as it notices a clause that forces  $l$  to be true. Later on, other clauses that force  $l$  are often encountered, in practice; but Algorithm C ignores them, even though one of them might be a “better reason.” (For example, another forcing clause might be significantly shorter.) Explore a modification of Algorithm C that tries to improve the reasons of non-decision literals.

► **275.** [22] Adapt Algorithm C to the problem of finding the lexicographically smallest solution to a satisfiability problem, by incorporating the ideas of exercise 109.

**276.** [M15] True or false: If  $F$  is a family of clauses and  $L$  is a set of strictly distinct literals, then  $F \wedge L \vdash_1 \epsilon$  if and only if  $(F|L) \vdash_1 \epsilon$ .

**277.** [M18] If  $(C_1, \dots, C_t)$  is a certificate of unsatisfiability for  $F$ , and if all clauses of  $F$  have length  $\geq 2$ , prove that some  $C_i$  is a unit clause.

**278.** [22] Find a six-step certificate of unsatisfiability for *waerden*(3, 3; 9).

**279.** [M20] True or false: Every unsatisfiable 2SAT problem has a certificate  $(l, \epsilon)$ .

► **280.** [M26] The problem *cook*( $j, k$ ) consists of all  $\binom{n}{j}$  positive  $j$ -clauses and all  $\binom{n}{k}$  negative  $k$ -clauses on  $\{1, \dots, n\}$ , where  $n = j + k - 1$ . For example, *cook*(2, 3) is

$$\{12, 13, 14, 23, 24, 34, \bar{1}\bar{2}\bar{3}, \bar{1}\bar{2}\bar{4}, \bar{1}\bar{3}\bar{4}, \bar{2}\bar{3}\bar{4}\}.$$

a) Why are these clauses obviously unsatisfiable?

b) Find a totally positive certificate for *cook*( $j, k$ ), of length  $\binom{n-1}{j-1}$ .

c) Prove in fact that Algorithm C always learns *exactly*  $\binom{n-1}{j-1}$  clauses when it proves the unsatisfiability of *cook*( $j, k$ ), if  $M_p = M_t = \infty$  (no purging or flushing).

**281.** [21] Construct a certificate of unsatisfiability that refutes (99), (100), (101).

► **282.** [M33] Construct a certificate of unsatisfiability for the clauses *fsnark*( $q$ ) of exercise 176 when  $q \geq 3$  is odd, using  $O(q)$  clauses, all having length  $\leq 4$ . *Hint:* Include the clauses  $(\bar{a}_{j,p} \vee \bar{e}_{j,p})$ ,  $(\bar{a}_{j,p} \vee \bar{f}_{j,p})$ ,  $(\bar{e}_{j,p} \vee \bar{f}_{j,p})$ , and  $(a_{j,p} \vee e_{j,p} \vee f_{j,p})$  for  $1 \leq j \leq q$ ,  $1 \leq p \leq 3$ .

**283.** [HM46] Does Algorithm C solve the flower snark problem in linear time? More precisely, let  $p_q(M)$  be the probability that the algorithm refutes *fsnark*( $q$ ) while making at most  $M$  references to MEM. Is there a constant  $N$  such that  $p_q(Nq) > \frac{1}{2}$  for all  $q$ ?

**284.** [23] Given  $F$  and  $(C_1, \dots, C_t)$ , a certificate-checking program tests condition (119) by verifying that  $F$  and clauses  $C_1, \dots, C_{i-1}$  will force a conflict when they are augmented by the unit literals of  $\bar{C}_i$ . While doing this, it can mark each clause of  $F \cup \{C_1, \dots, C_{i-1}\}$  that was reduced to a unit during the forcing process; then the truth of  $C_i$  does not depend on the truth of any unmarked clause.

In practice, many clauses of  $F$  are never marked at all, hence  $F$  will remain unsatisfiable even if we leave them out. Furthermore, many clauses  $C_i$  are not marked during the verification of any of their successors,  $\{C_{i+1}, \dots, C_t\}$ ; such clauses  $C_i$  needn't be verified, nor need we mark any of the clauses on which they depend.

Therefore we can save work by checking the certificate backwards: Start by marking the final clause  $C_t$ , which is  $\epsilon$  and always needs to be verified. Then, for  $i = t, t-1, \dots$ , check  $C_i$  only if it has been marked.

The unit propagations can all be done without recording the “reason”  $R_l$  that has caused any literal  $l$  to be forced. In practice, however, many of the forced literals don't actually contribute to the conflicts that arise, and we don't want to mark any clauses that aren't really involved.

Explain how to use reasons, as in Algorithm C, so that clauses are marked by the verifier only if they actually participate in the proof of a marked clause  $C_i$ .

bounded model checking  
better reason  
lexicographically smallest solution  
notation  $F|L$   
notation  $F \vdash_1 \epsilon$   
certificate of unsatisfiability  
unit clause  
2SAT  
positive  $j$ -clauses  
negative  $k$ -clauses  
*cook*  
purging  
flushing  
maximal elements  
fsnark  
flower snark  
reason

- 285.** [19] Using the data in Fig. 50, the text observes that Eq. (124) gives  $j = 95$ ,  $s_j = 3081$ , and  $m_j = 59$  when  $\alpha = \frac{15}{16}$ . What are  $j$ ,  $s_j$ , and  $m_j$  when (a)  $\alpha = \frac{9}{16}$ ? (b)  $\alpha = \frac{1}{2}$ ? (c)  $\alpha = \frac{7}{16}$ ? Also compare the effectiveness of different  $\alpha$ 's by computing the number  $b_j$  of “black” clauses (those with  $0 < \text{RANGE}(c) < j$  that proved to be useful).
- 286.** [M24] What choice of signatures-to-keep in Fig. 50 is *optimum*, in the sense that it maximizes  $\sum b_{pq}x_{pq}$  subject to the conditions  $\sum a_{pq}x_{pq} \leq 3114$ ,  $x_{pq} \in \{0, 1\}$ , and  $x_{pq} \geq x_{p'q'}$  for  $1 \leq p \leq p' \leq 7$ ,  $0 \leq q \leq q' \leq 8$ ? Here  $a_{pq}$  and  $b_{pq}$  are the areas of the gray and black clauses that have signature  $(p, q)$ , as given by the matrices in the text. [This is a special case of the “knapsack problem with a partial ordering.”]
- 287.** [25] What changes to Algorithm C are necessary to make it do a “full run,” and later to learn from all of the conflicts that arose during that run?
- 288.** [28] Spell out the details of computing RANGE scores and then compressing the database of learned clauses, during a round of purging.
- 289.** [M20] Assume that the  $k$ th round of purging begins with  $y_k$  clauses in memory after  $k\Delta + \binom{k}{2}\delta$  clauses have been learned, and that purging removes  $\frac{1}{2}y_k$  of those clauses. Find a closed formula for  $y_k$  as a function of  $k$ .
- 290.** [17] Explain how to find  $x_k$ , the unassigned variable of maximum activity that is used for flushing literals. *Hint:* It's in the HEAP array.
- 291.** [20] In the text's hypothetical scenario about flushing Table 3 back to level 15, why will 49 soon appear on the trail, instead of 49?
- 292.** [M21] How large can AGILITY get after repeatedly executing (127)?
- 293.** [21] Spell out the details of updating  $M_f$  to  $M + \Delta_f$  when deciding whether or not to flush. Also compute the agility threshold that's specified in Table 4. *Hint:* See (131).
- 294.** [HM21] For each binary vector  $\alpha = x_1x_2x_3x_4$ , find the generating function  $g_\alpha(z) = \sum_{j=0}^{\infty} p_{\alpha,j}z^j$ , where  $p_{\alpha,j}$  is the probability that Algorithm P will solve the seven clauses of (7) after making exactly  $j$  flips, given the initial values  $\alpha$  in step P1. Deduce the mean and variance of the number of steps needed to find a solution.
- 295.** [M23] Algorithm P often finds solutions much more quickly than predicted by Corollary W. But show that some 3SAT clauses will indeed require  $\Omega((4/3)^n)$  trials.
- 296.** [HM20] Complete the proof of Theorem U by (approximately) maximizing the quantity  $f(p, q)$  in (129). *Hint:* Consider  $f(p+1, q)/f(p, q)$ .
- **297.** [HM26] (Emo Welzl.) Let  $G_q(z) = \sum_p C_{p,p+q-1}(z/3)^{p+q}(2z/3)^p$  be the generating function for stopping time  $t = 2p + q$  when  $Y_0 = q$  in the proof of Theorem U.
- Find a closed form for  $G_q(z)$ , using formulas from Section 7.2.1.6.
  - Explain why  $G_q(1)$  is less than 1.
  - Evaluate and interpret the quantity  $G'_q(1)/G_q(1)$ .
  - Use Markov's inequality to bound the probability that  $Y_t = 0$  for some  $t \leq N$ .
  - Show that Corollary W follows from this analysis.
- 298.** [HM22] Generalize Theorem U and Corollary W to the case where each clause has at most  $k$  literals, where  $k \geq 3$ .
- 299.** [HM23] Continuing the previous exercise, investigate the case  $k = 2$ .
- **300.** [25] Modify Algorithm P so that it can be implemented with bitwise operations, thereby running (say) 64 independent trials simultaneously.
- **301.** [25] Discuss implementing the algorithm of exercise 300 efficiently on MMIX.

signature  
knapsack problem with a partial ordering  
full run  
compressing  
purging  
flushing literals  
**HEAP**  
**AGILITY**  
agility threshold  
generating function  
variance  
Welzl  
Markov's inequality  
bitwise operations  
broadword computation  
**MMIX**

**302.** [26] Expand the text's high-level description of steps W4 and W5, by providing low-level details about exactly what the computer should do.

**303.** [HM20] Solve exercise 294 with Algorithm W in place of Algorithm P.

**304.** [HM34] Consider the 2SAT problem with  $n(n-1)$  clauses  $(\bar{x}_j \vee x_k)$  for all  $j \neq k$ . Find the generating functions for the number of flips taken by Algorithms P and W. *Hint:* Exercises 1.2.6–68 and MPR-105 are helpful for finding the exact formulas.

► **305.** [M25] Add one more clause,  $(\bar{x}_1 \vee \bar{x}_2)$ , to the previous exercise and find the resulting generating functions when  $n = 4$ . What happens when  $p = 0$  in Algorithm W?

► **306.** [HM32] (Luby, Sinclair, and Zuckerman, 1993.) Consider a “Las Vegas algorithm” that succeeds or fails; it succeeds at step  $t$  with probability  $p_t$ , and fails with probability  $p_\infty < 1$ . Let  $q_t = p_1 + p_2 + \dots + p_t$  and  $E_t = p_1 + 2p_2 + \dots + tp_t$ ; also let  $E_\infty = \infty$  if  $p_\infty > 0$ , otherwise  $E_\infty = \sum_t tp_t$ . (The latter sum might be  $\infty$ .)

- Suppose we abort the algorithm and restart it again, whenever the first  $N$  steps have not succeeded. Show that if  $q_N > 0$ , this strategy will succeed after performing an average of  $l(N) < \infty$  steps. What is  $l(N)$ ?
- Compute  $l(N)$  when  $p_m = \frac{m}{n}$ ,  $p_\infty = \frac{n-m}{n}$ , otherwise  $p_t = 0$ , where  $1 \leq m \leq n$ .
- Given the uniform distribution,  $p_t = \frac{1}{n}$  for  $1 \leq t \leq n$ , what is  $l(N)$ ?
- Find all probability distributions such that  $l(N) = l(1)$  for all  $N \geq 1$ .
- Find all probability distributions such that  $l(N) = l(n)$  for all  $N \geq n$ .
- Find all probability distributions such that  $q_{n+1} = 1$  and  $l(n) \leq l(n+1)$ .
- Find all probability distributions such that  $q_3 = 1$  and  $l(1) < l(3) < l(2)$ .
- Let  $l = \inf_{N \geq 1} l(N)$ , and let  $N^*$  be the least positive integer such that  $l(N^*) = l$ , or  $\infty$  if no such integer exists. Prove that  $N^* = \infty$  implies  $l = E_\infty < \infty$ .
- Find  $N^*$  for the probability distribution  $p_t = [t > n] / ((t-n)(t+1-n))$ , given  $n \geq 0$ .
- Exhibit a simple example of a probability distribution for which  $N^* = \infty$ .
- Let  $L = \min_{t \geq 1} t/q_t$ . Prove that  $l \leq L \leq 2l - 1$ .

**307.** [HM28] Continuing exercise 306, consider a more general strategy defined by an infinite sequence of positive integers  $(N_1, N_2, \dots)$ : “Set  $j \leftarrow 0$ ; then, while success has not yet been achieved, set  $j \leftarrow j + 1$  and run the algorithm with cutoff parameter  $N_j$ .”

- Explain how to compute  $EX$ , where  $X$  is the number of steps taken before this strategy succeeds.
- Let  $T_j = N_1 + \dots + N_j$ . Prove that  $EX = \sum_{j=1}^{\infty} \Pr(T_{j-1} < X \leq T_j) l(N_j)$ , if we have  $q_{N_j} > 0$  for all  $j$ .
- Consequently the steady strategy  $(N^*, N^*, \dots)$  is best:  $EX \geq l(N^*) = l$ .
- Given  $n$ , exercise 306(b) defines  $n$  simple probability distributions  $p^{(m)}$  that have  $l(N^*) = n$ , but the value of  $N^* = m$  is different in each case. Prove that any sequence  $(N_1, N_2, \dots)$  must have  $EX > \frac{1}{4}nH_n - \frac{1}{2}n = \frac{1}{4}lH_l - \frac{1}{2}l$  on at least one of those  $p^{(m)}$ . *Hint:* Consider the smallest  $r$  such that, for each  $m$ , the probability is  $\geq \frac{1}{2}$  that  $r$  trial runs suffice; show that  $\geq n/(2m)$  of  $\{N_1, \dots, N_r\}$  are  $\geq m$ .

**308.** [M29] This exercise explores the “reluctant doubling” sequence (130).

- What is the smallest  $n$  such that  $S_n = 2^a$ , given  $a \geq 0$ ?
- Show that  $\{n \mid S_n = 1\} = \{2k + 1 - \nu k \mid k \geq 0\}$ ; hence the generating function  $\sum_n z^n [S_n = 1]$  is the infinite product  $z(1+z)(1+z^3)(1+z^7)(1+z^{15}) \dots$ .
- Find similar expressions for  $\{n \mid S_n = 2^a\}$  and  $\sum_n z^n [S_n = 2^a]$ .
- Let  $\Sigma(a, b, k) = \sum_{n=1}^{r(a,b,k)} S_n$ , where  $S_{r(a,b,k)}$  is the  $2^b k$ th occurrence of  $2^a$  in  $\langle S_n \rangle$ . For example,  $\Sigma(1, 0, 3) = S_1 + \dots + S_{10} = 16$ . Evaluate  $\Sigma(a, b, 1)$  in closed form.
- Show that  $\Sigma(a, b, k+1) - \Sigma(a, b, k) \leq (a + b + 2k - 1)2^{a+b}$ , for all  $k \geq 1$ .

WalkSAT+  
analysis of algorithms  
Luby  
Sinclair  
Zuckerman  
Las Vegas algorithm  
uniform distribution  
reluctant doubling  
generating function

- f) Given any probability distribution as in exercise 306(k), let  $a = \lceil \lg t \rceil$  and  $b = \lceil \lg 1/q_t \rceil$ , where  $t/q_t = L$ ; thus  $L \leq 2^{a+b} < 4L$ . Prove that if the strategy of exercise 307 is used with  $N_j = S_j$ , we have

$$E X \leq \Sigma(a, b, 1) + \sum_{k \geq 1} Q^k (\Sigma(a, b, k+1) - \Sigma(a, b, k)), \quad \text{where } Q = (1 - q_{2^a})^{2^b}.$$

- g) Therefore  $\langle S_n \rangle$  gives  $E X < 13l \lg l + 49l$ , for every probability distribution.

**309.** [20] Exercise 293 explains how to use the reluctant doubling sequence with Algorithm C. Is Algorithm C a Las Vegas algorithm?

**310.** [M25] Explain how to compute the “reluctant Fibonacci sequence”

1, 1, 2, 1, 2, 3, 1, 1, 2, 3, 5, 1, 1, 2, 1, 2, 3, 5, 8, 1, 1, 2, 1, 2, 3, 1, 1, 2, 3, 5, 8, 13, 1, \dots,

which is somewhat like (130) and useful as in exercise 308, but its elements are Fibonacci numbers instead of powers of 2.

**311.** [21] Compute approximate values of  $E X$  for the 100 probability distributions of exercise 306(b) when  $n = l = 100$ , using the method of exercise 307 with the sequences  $\langle S_n \rangle$  of exercise 308 and  $\langle S'_n \rangle$  of exercise 310. Also consider the more easily generated “ruler doubling” sequence  $\langle R_n \rangle$ , where  $R_n = n$  &  $-n = 2^{\rho n}$ . Which sequence is best?

**312.** [HM24] Let  $T(m, n) = E X$  when the reluctant doubling method is applied to the probability distribution defined in exercise 306(b). Express  $T(m, n)$  in terms of the generating functions in exercise 308(c).

► **313.** [22] Algorithm W always flips a cost-free literal if one is present in  $C_j$ , without considering its parameter  $p$ . Show that such a flip always decreases the number of unsatisfied clauses,  $r$ ; but it might *increase* the distance from  $x$  to the nearest solution.

► **314.** [36] (H. H. Hoos, 1998.) If the given clauses are satisfiable, and if  $p > 0$ , can there be an initial  $x$  for which Algorithm W always loops forever?

**315.** [M18] What value of  $p$  is appropriate in Theorem J when  $d = 1$ ?

**316.** [HM20] Is Theorem J a consequence of Theorem L?

► **317.** [M26] Let  $\alpha(G) = \Pr(\bar{A}_1 \cap \dots \cap \bar{A}_m)$  under the assumptions of (133), when  $p_i = p = (d-1)^{d-1}/d^d$  for  $1 \leq i \leq m$  and every vertex of  $G$  has degree at most  $d > 1$ . Prove, by induction on  $m$ , that  $\alpha(G) > 0$  and that  $\alpha(G) > \frac{d-1}{d} \alpha(G \setminus v)$  when  $v$  has degree  $< d$ .

**318.** [HM27] (J. B. Shearer.) Prove that Theorem J is the best possible result of its kind: If  $p > (d-1)^{d-1}/d^d$  and  $d > 1$ , there is a graph  $G$  of maximum degree  $d$  for which  $(p, \dots, p) \notin \mathcal{R}(G)$ . *Hint:* Consider complete  $t$ -ary trees, where  $t = d - 1$ .

**319.** [HM20] Show that  $pde < 1$  implies  $p \leq (d-1)^{d-1}/d^d$ .

**320.** [M24] Given a lopsided dependency graph  $G$ , the *occurrence threshold*  $\rho(G)$  is the smallest value  $p$  such that it's sometimes impossible to avoid all events when each event occurs with probability  $p$ . For example, the Möbius polynomial for the path  $P_3$  is  $1 - p_1 - p_2 - p_3 + p_1 p_3$ ; so the occurrence threshold is  $\phi^{-2}$ , the least  $p$  with  $1 - 3p + p^2 \leq 0$ .

a) Prove that the occurrence threshold for  $P_m$  is  $1/(4 \cos^2 \frac{\pi}{m+2})$ .

b) What is the occurrence threshold for the cycle graph  $C_m$ ?

**321.** [M24] Suppose each of four random events  $A, B, C, D$  occurs with probability  $p$ , where  $\{A, C\}$  and  $\{B, D\}$  are independent. According to exercise 320(b) with  $m = 4$ , there's a joint distribution of  $(A, B, C, D)$  such that at least one of the events always occurs, whenever  $p \geq (2 - \sqrt{2})/2 \approx 0.293$ . Exhibit such a distribution when  $p = 3/10$ .

Las Vegas  
reluctant Fibonacci sequence  
Fibonacci numbers  
ruler doubling  
Hoos  
Shearer  
complete  $t$ -ary trees  
lopsided dependency graph  
Möbius polynomial  
path  
phi  
cycle graph

- **322.** [HM35] (K. Kolipaka and M. Szegedy, 2011.) Surprisingly, the previous exercise *cannot* be solved in the setting of Algorithm M! Suppose we have independent random variables  $(W, X, Y, Z)$  such that  $A$  depends on  $W$  and  $X$ ,  $B$  depends on  $X$  and  $Y$ ,  $C$  depends on  $Y$  and  $Z$ ,  $D$  depends on  $Z$  and  $W$ . Here  $W$  equals  $j$  with probability  $w_j$  for all integers  $j$ ;  $X, Y$ , and  $Z$  are similar. This exercise will prove that the constraint  $\overline{A} \cap \overline{B} \cap \overline{C} \cap \overline{D}$  is always satisfiable, even when  $p$  is as large as 0.333.

- a) Express the probability  $\Pr(\overline{A} \cap \overline{B} \cap \overline{C} \cap \overline{D})$  in a convenient way.  
 b) Suppose there's a distribution of  $W, X, Y, Z$  with  $\Pr(A) = \Pr(B) = \Pr(C) = \Pr(D) = p$  and  $\Pr(\overline{A} \cap \overline{B} \cap \overline{C} \cap \overline{D}) = 0$ . Show that there are ten values such that

$$\begin{aligned} 0 \leq a, b, c, d, a', b', c', d' \leq 1, & & 0 < \mu, \nu < 1, \\ \mu a + (1 - \mu)a' \leq p, & & \mu b + (1 - \mu)b' \leq p, \\ \nu c + (1 - \nu)c' \leq p, & & \nu d + (1 - \nu)d' \leq p, \\ a + d \geq 1 \text{ or } b + c \geq 1, & & a + d' \geq 1 \text{ or } b + c' \geq 1, \\ a' + d \geq 1 \text{ or } b' + c \geq 1, & & a' + d' \geq 1 \text{ or } b' + c' \geq 1. \end{aligned}$$

- c) Find all solutions to those constraints when  $p = 1/3$ .  
 d) Convert those solutions to distributions that have  $\Pr(\overline{A} \cap \overline{B} \cap \overline{C} \cap \overline{D}) = 0$ .

**323.** [10] What trace precedes  $ccb$  in the list (135)?

- **324.** [22] Given a trace  $\alpha = x_1 x_2 \dots x_n$  for a graph  $G$ , explain how to find all strings  $\beta$  that are equivalent to  $\alpha$ , using Algorithm 7.2.1.2V. How many strings yield (136)?

- **325.** [20] An *acyclic orientation* of a graph  $G$  is an assignment of directions to each of its edges so that the resulting digraph has no oriented cycles. Show that the number of traces for  $G$  that are *permutations* of the vertices (with each vertex appearing exactly once in the trace) is the number of acyclic orientations of  $G$ .

**326.** [20] True or false: If  $\alpha$  and  $\beta$  are traces with  $\alpha = \beta$ , then  $\alpha^R = \beta^R$ . (See (137).)

- **327.** [22] Design an algorithm to multiply two traces  $\alpha$  and  $\beta$ , when clashing is defined by territory sets  $T(a)$  in some universe  $U$ . Assume that  $U$  is small (say  $|U| \leq 64$ ), so that bitwise operations can be used to represent the territories.

**328.** [20] Continuing exercise 327, design an algorithm that computes  $\alpha/\beta$ . More precisely, if  $\beta$  is a right factor of  $\alpha$ , in the sense that  $\alpha = \gamma\beta$  for some trace  $\gamma$ , your algorithm should compute  $\gamma$ ; otherwise it should report that  $\beta$  is *not* a right factor.

**329.** [21] Similarly, design an algorithm that either computes  $\alpha \setminus \beta$  or reports that  $\alpha$  isn't a left factor of  $\beta$ .

- **330.** [21] Given any graph  $G$ , explain how to define territory sets  $T(a)$  for its vertices  $a$  in such a way that we have  $a = b$  or  $a \text{ --- } b$  if and only if  $T(a) \cap T(b) \neq \emptyset$ . (Thus traces can always be modeled by empilements of pieces.) Under what circumstances is it possible to do this with  $|T(a)| = 2$  for all  $a$ , as in the text's example (136)?

**331.** [M20] What happens if the right-hand side of (139) is expanded without allowing *any* of the variables to commute with each other?

**332.** [20] When a trace is represented by its lexicographically smallest string, no letter in that representative string is followed by a smaller letter with which it commutes. (For example, no  $c$  is followed by  $a$  in (135), because we could get an equivalent smaller string by changing  $ca$  to  $ac$ .)

Conversely, given any ordered set of letters, some of which commute, consider all strings having no letter followed by a smaller letter with which it commutes. Is every such string the lexicographically smallest of its trace?

Kolipaka  
 Szegedy  
 acyclic orientation  
 digraph  
 territory sets  
 bitwise operations  
 territories  
 multiplication of traces  
 right factor  
 right division of traces  
 left factor  
 left division of traces  
 territory sets  
 intersection graphs  
 empilements

- ▶ **333.** [M20] (Carlitz, Scoville, and Vaughan, 1976.) Let  $D$  be a digraph on  $\{1, \dots, m\}$ , and let  $A$  be the set of all strings  $a_{j_1} \dots a_{j_n}$  such that  $j_i \rightarrow j_{i+1}$  in  $D$  for  $1 \leq i < n$ . Similarly let  $B$  be the set of all strings  $a_{j_1} \dots a_{j_n}$  such that  $j_i \not\rightarrow j_{i+1}$  for  $1 \leq i < n$ . Prove that

$$\sum_{\alpha \in A} \alpha = 1 / \sum_{\beta \in B} (-1)^{|\beta|} \beta = \sum_{k \geq 0} \left( 1 - \sum_{\beta \in B} (-1)^{|\beta|} \beta \right)^k$$

is an identity in the noncommutative variables  $\{a_1, \dots, a_m\}$ . (For example, we have

$$1 + a + b + ab + ba + aba + bab + \dots = \sum_{k \geq 0} (a + b - aa - bb + aaa + bbb - \dots)^k$$

in the case  $m = 2$ ,  $1 \not\rightarrow 1$ ,  $1 \rightarrow 2$ ,  $2 \rightarrow 1$ ,  $2 \not\rightarrow 2$ .)

- ▶ **334.** [25] Design an algorithm to generate all traces of length  $n$  that correspond to a given graph on the alphabet  $\{1, \dots, m\}$ , representing each trace by its lexicographically smallest string.

**335.** [HM26] If the vertices of  $G$  can be ordered in such a way that  $x < y < z$  and  $x \not\rightarrow y$  and  $y \not\rightarrow z$  implies  $x \not\rightarrow z$ , show that the Möbius series  $M_G$  can be expressed as a determinant. For example,

if  $G = \begin{array}{ccc} a & \circ & b \\ c & \circ & d \\ e & \circ & f \end{array}$  then  $M_G = \det \begin{pmatrix} 1-a & -b & -c & 0 & 0 & 0 \\ -a & 1-b & 0 & -d & 0 & 0 \\ -a & -b & 1-c & -d & -e & 0 \\ -a & -b & -c & 1-d & 0 & -f \\ -a & -b & -c & -d & 1-e & -f \\ -a & -b & -c & -d & -e & 1-f \end{pmatrix}$ .

- ▶ **336.** [M20] If graphs  $G$  and  $H$  on distinct vertices have the Möbius series  $M_G$  and  $M_H$ , what are the Möbius series for (a)  $G \oplus H$  and (b)  $G \text{---} H$ ?

**337.** [M20] Suppose we obtain the graph  $G'$  from  $G$  by substituting a clique of vertices  $\{a_1, \dots, a_k\}$  for some vertex  $a$ , then including edges from  $a_j$  to each neighbor of  $a$  for  $1 \leq j \leq k$ . Describe the relation between  $M_{G'}$  and  $M_G$ .

**338.** [M21] Prove Viennot's general identity (144) for source-constrained traces.

- ▶ **339.** [HM26] (G. Viennot.) This exercise explores factorization of traces into pyramids.
  - Each letter  $x_j$  of a given trace  $\alpha = x_1 \dots x_n$  lies at the top of a unique pyramid  $\beta_j$  such that  $\beta_j$  is a left factor of  $\alpha$ . For example, in the trace  $bcebafdc$  of (136), the pyramids  $\beta_1, \dots, \beta_8$  are respectively  $b, bc, e, bcb, bcb, ef, bced$ , and  $bcebdc$ . Explain intuitively how to find these pyramidal left factors from  $\alpha$ 's empilement.
  - A *labeled trace* is an assignment of distinct numbers to the letters of a trace; for example,  $abca$  might become  $a_4 b_7 c_6 a_3$ . A *labeled pyramid* is the special case when the pyramid's top element is required to have the smallest label. Prove that every labeled trace is uniquely factorizable into labeled pyramids whose topmost labels are in ascending order. (For example,  $b_6 c_2 e_4 b_7 a_8 f_5 d_1 c_3 = b_6 c_2 e_4 d_1 \cdot b_7 a_8 c_3 \cdot f_5$ .)
  - Suppose there are  $t_n$  traces of length  $n$ , and  $p_n$  pyramids. Then there are  $T_n = n! t_n$  labeled traces and  $P_n = (n-1)! p_n$  labeled pyramids (because only the relative order of the labels is significant). Letting  $T(z) = \sum_{n \geq 0} T_n z^n / n!$  and  $P(z) = \sum_{n \geq 1} P_n z^n / n!$ , prove that the number of labeled traces of length  $n$  whose factorization in part (b) has exactly  $l$  pyramids is  $n! [z^n] P(z)^l / l!$ .
  - Consequently  $T(z) = e^{P(z)}$ .
  - Therefore (and this is the punch line!)  $\ln M_G(z) = -\sum_{n \geq 1} p_n z^n / n$ .

Carlitz  
 Scoville  
 Vaughan  
 digraph  
 digraph  
 noncommutative variables  
 lexicographically smallest  
 determinant  
 Möbius series  
 direct sum of graphs  
 join of graphs  
 clique  
 Viennot  
 Viennot  
 pyramids  
 left factor  
 labeled trace  
 labeled pyramid  
 generating functions, exponential

- **340.** [M20] If we assign a weight  $w(\sigma)$  to every cyclic permutation  $\sigma$ , then every permutation  $\pi$  has a weight  $w(\pi)$  that is the product of the weights of its cycles. For example, if  $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 1 & 4 & 2 & 7 & 6 & 5 \end{pmatrix} = (1\ 3\ 4\ 2)(5\ 7)(6)$  then  $w(\pi) = w((1\ 3\ 4\ 2))w((5\ 7))w((6))$ .

The *permutation polynomial* of a set  $S$  is the sum of  $w(\pi)$  over all permutations of  $S$ . Given any  $n \times n$  matrix  $A = (a_{ij})$ , show that it's possible to define appropriate cycle weights so that the permutation polynomial of  $\{1, \dots, n\}$  is the determinant of  $A$ .

- 341.** [M25] The *involution polynomial* of a set  $S$  is the special case of the permutation polynomial when the cycle weights have the form  $w_{jj}x$  for the 1-cycle  $(j)$  and  $-w_{ij}$  for the 2-cycle  $(ij)$ , otherwise  $w(\sigma) = 0$ . For example, the involution polynomial of  $\{1, 2, 3, 4\}$  is  $w_{11}w_{22}w_{33}w_{44}x^4 - w_{11}w_{22}w_{34}x^2 - w_{11}w_{23}w_{44}x^2 - w_{11}w_{24}w_{33}x^2 - w_{12}w_{33}w_{44}x^2 - w_{13}w_{22}w_{44}x^2 - w_{14}w_{22}w_{33}x^2 + w_{12}w_{34} + w_{13}w_{24} + w_{14}w_{23}$ .

Prove that, if  $w_{ij} > 0$  for  $1 \leq i \leq j \leq n$ , the involution polynomial of  $\{1, \dots, n\}$  has  $n$  distinct real roots. *Hint:* Show also that, if the roots for  $\{1, \dots, n-1\}$  are  $q_1 < \dots < q_{n-1}$ , then the roots  $r_k$  for  $\{1, \dots, n\}$  satisfy  $r_1 < q_1 < r_2 < \dots < q_{n-1} < r_n$ .

- 342.** [HM25] (Cartier and Foata, 1969.) Let  $G_n$  be the graph whose vertices are the  $\sum_{k=1}^n \binom{n}{k}(k-1)!$  cyclic permutations of subsets of  $\{1, \dots, n\}$ , with  $\sigma \text{ --- } \tau$  when  $\sigma$  and  $\tau$  intersect. For example, the vertices of  $G_3$  are  $(1), (2), (3), (12), (13), (23), (123), (132)$ ; and they're mutually adjacent except that  $(1) \not\text{---} (2), (1) \not\text{---} (3), (1) \not\text{---} (23), (2) \not\text{---} (3), (2) \not\text{---} (13), (12) \not\text{---} (3)$ . Find a beautiful relation between  $M_{G_n}$  and the characteristic polynomial of an  $n \times n$  matrix.

- **343.** [M25] If  $G$  is any cograph, show that  $(p_1, \dots, p_m) \in \mathcal{R}(G)$  if and only if we have  $M_G(p_1, \dots, p_m) > 0$ . Exhibit a non-cograph for which the latter statement is *not* true.

- 344.** [M33] Given a graph  $G$  as in Theorem S, let  $B_1, \dots, B_m$  have the joint probability distribution of exercise MPR-31, with  $\pi_I = 0$  whenever  $I$  contains distinct vertices  $\{i, j\}$  with  $i \text{ --- } j$ , otherwise  $\pi_I = \prod_{i \in I} p_i$ .

- Show that this distribution is legal (see exercise MPR-32) if  $(p_1, \dots, p_m) \in \mathcal{R}(G)$ .
- Show that this "extreme distribution" also satisfies condition (147).
- Let  $\beta(G) = \Pr(\overline{B_1} \cap \dots \cap \overline{B_m})$ . If  $J \subseteq \{1, \dots, m\}$ , express  $\beta(G|J)$  in terms of  $M_G$ .
- Defining  $\alpha(G)$  as in exercise 317, with events  $A_j$  satisfying (133) and probabilities  $(p_1, \dots, p_m) \in \mathcal{R}(G)$ , show that  $\alpha(G|J) \geq \beta(G|J)$  for all  $J \subseteq \{1, \dots, m\}$ .
- If  $p_i$  satisfies (134), show that  $\beta(G|J) \geq \prod_{j \in J} (1 - \theta_j)$ .

- 345.** [M30] Construct unavoidable events that satisfy (147) when  $(p_1, \dots, p_m) \notin \mathcal{R}(G)$ .

- **346.** [HM28] Write (142) as  $M_G = M_{G \setminus a}(1 - aK_{a,G})$  where  $K_{a,G} = M_{G \setminus a^*}/M_{G \setminus a}$ .
- If  $(p_1, \dots, p_m) \in \mathcal{R}(G)$ , prove that  $K_{a,G}$  is monotonic in all of its parameters: It does not increase if any of  $p_1, \dots, p_m$  are decreased.
  - Exploit this fact to design an algorithm that computes  $M_G(p_1, \dots, p_m)$  and decides whether or not  $(p_1, \dots, p_m) \in \mathcal{R}(G)$ , given a graph  $G$  and probabilities  $(p_1, \dots, p_m)$ . Illustrate your algorithm on the graph  $G = P_3 \square P_2$  of exercise 335.

- **347.** [M28] A graph is called *chordal* when it has no induced cycle  $C_k$  for  $k > 3$ . Equivalently (see Section 7.4.2), a graph is chordal if and only if its edges can be defined by territory sets  $T(a)$  that induce connected subgraphs of some tree. For example, interval graphs and forests are chordal.

- Say that a graph is *tree-ordered* if its vertices can be arranged as nodes of a forest in such a way that

$$\begin{aligned} a \text{ --- } b &\text{ implies } a \succ b \text{ or } b \succ a; \\ a \succ b \succ c &\text{ and } a \text{ --- } c \text{ implies } a \text{ --- } b. \end{aligned} \quad (*)$$

cyclic permutation  
weighted permutations  
permutations, weighted  
permutation polynomial  
determinant  
involution polynomial  
real roots of polynomials  
interlaced roots  
Cartier  
Foata  
characteristic polynomial  
cograph  
extreme distribution  
monotonic  
chordal  
territory sets  
interval graphs  
forests



(Here ' $a \succ b$ ' means that  $a$  is a proper ancestor of  $b$  in the forest.) Prove that every tree-ordered graph is chordal.

- b) Conversely, show that every chordal graph can be tree-ordered.
- c) Show that the algorithm in the previous exercise becomes quite simple when it is applied to a tree-ordered graph, if  $a$  is eliminated before  $b$  whenever  $a \succ b$ .
- d) Consequently Theorem L can be substantially strengthened when  $G$  is a chordal graph: When  $G$  is tree-ordered by  $\succ$ , the probability vector  $(p_1, \dots, p_m)$  is in  $\mathcal{R}(G)$  if and only if there are numbers  $0 \leq \theta_1, \dots, \theta_m < 1$  such that

$$p_i = \theta_i \prod_{i-j \text{ in } G, i \succ j} (1 - \theta_j).$$

proper ancestor  
tree-ordered graph  
Pringsheim  
nonnegative coefficients  
analysis of algorithms  
variance  
Pegden  
asymptotic  
Local Lemma  
dependency graph  
resolvable  
lopsidependency graph

**348.** [HM26] (A. Pringsheim, 1894.) Show that any power series  $f(z) = \sum_{n=0}^{\infty} a_n z^n$  with  $a_n \geq 0$  and radius of convergence  $\rho$ , where  $0 < \rho < \infty$ , has a singularity at  $z = \rho$ .

- ▶ **349.** [M24] Analyze Algorithm M *exactly* in the two examples considered in the text (see (150)): For each binary vector  $x = x_1 \dots x_7$ , compute the generating function  $g_x(z) = \sum_t p_{x,t} z^t$ , where  $p_{x,t}$  is the probability that step M3 will be executed exactly  $t$  times after step M1 produces  $x$ . Assume that step M2 always chooses the smallest possible value of  $j$ . (Thus the 'Case 2' scenario in (150) will never occur.)

What are the mean and variance of the running times, in (i) Case 1? (ii) Case 2?

- ▶ **350.** [HM26] (W. Pegden.) Suppose Algorithm M is applied to the  $m = n + 1$  events

$$A_j = x_j \quad \text{for } 1 \leq j \leq n; \quad A_m = x_1 \vee \dots \vee x_n.$$

Thus  $A_m$  is true whenever any of the other  $A_j$  is true; so we could implement step M2 by never setting  $j \leftarrow m$ . Alternatively, we could decide to set  $j \leftarrow m$  whenever possible. Let  $(N_i, N_{ii}, N_{iii}, N_{iv}, N_v)$  be the number of resamplings performed when parameter  $\xi_k$  of the algorithm is (i)  $1/2$ ; (ii)  $1/(2n)$ ; (iii)  $1/2^n$ ; (iv)  $1/(n+k)$ ; (v)  $1/(n+k)^2$ .

- a) Find the asymptotic mean and variance of each  $N$ , if  $j$  is never equal to  $m$ .
- b) Find the asymptotic mean and variance of each  $N$ , if  $j$  is never less than  $m$ .
- c) Let  $G$  be the graph on  $\{1, \dots, n+1\}$  with edges  $j \text{ --- } (n+1)$  for  $1 \leq j \leq n$ , and let  $p_j = \Pr(A_j)$ . For which of the five choices of  $\xi_k$  is  $(p_1, \dots, p_{n+1}) \in \mathcal{R}(G)$ ?

- ▶ **351.** [25] The Local Lemma can be applied to the satisfiability problem for  $m$  clauses on  $n$  variables if we let  $A_j$  be the event " $C_j$  is not satisfied." The dependency graph  $G$  then has  $i \text{ --- } j$  whenever two clauses  $C_i$  and  $C_j$  share at least one common variable. If, say,  $C_i$  is  $(x_3 \vee \bar{x}_5 \vee x_6)$ , then (133) holds whenever  $p_i \geq (1 - \xi_3)\xi_5(1 - \xi_6)$ , assuming that each  $x_k$  is true with probability  $\xi_k$ , independent of the other  $x$ 's.

But if, say,  $C_j$  is  $(\bar{x}_2 \vee x_3 \vee x_7)$ , condition (133) remains true even if we don't stipulate that  $i \text{ --- } j$ . Variable  $x_3$  appears in both clauses, yet the cases when  $C_j$  is satisfied are never bad news for  $C_i$ . We need to require that  $i \text{ --- } j$  in condition (133) only when  $C_i$  and  $C_j$  are "resolvable" clauses, namely when some variable occurs positively in one and negatively in the other.

Extend this reasoning to the general setting of Algorithm M, where we have arbitrary events  $A_j$  that depend on variables  $\Xi_j$ : Define a lopsidependency graph  $G$  for which (133) holds even though we might have  $i \not\text{---} j$  in some cases when  $\Xi_i \cap \Xi_j \neq \emptyset$ .

**352.** [M21] Show that  $E_j \leq \theta_j / (1 - \theta_j)$  in (152), when (134) holds.

**353.** [M21] Consider Case 1 and Case 2 of Algorithm M as illustrated in (150).

- a) How many solutions  $x_1 \dots x_n$  are possible? (Generalize from  $n = 7$  to any  $n$ .)
- b) How many solutions are predicted by Theorem S?

c) Show that in Case 2 the lopsidedependency graph is much smaller than the dependency graph. How many solutions are predicted when the smaller graph is used?

**354.** [HM20] Show that the expected number  $EN$  of resampling steps in Algorithm M is at most  $-M_G^*(1)/M_G^*(1)$ .

**355.** [HM21] In (152), prove that  $E_j \leq 1/\delta$  when  $(p_1, \dots, p_m)$  has positive slack  $\delta$ . *Hint:* Consider replacing  $p_j$  by  $p_j + \delta p_j$ .

► **356.** [M33] (*The Clique Local Lemma.*) Let  $G$  be a graph on  $\{1, \dots, m\}$ , and let  $G|U_1, \dots, G|U_t$  be cliques that cover all the edges of  $G$ . Assign numbers  $\theta_{ij} \geq 0$  to the vertices of each  $U_j$ , such that  $\Sigma_j = \sum_{i \in U_j} \theta_{ij} < 1$ . Assume that

$$\Pr(A_i) = p_i \leq \theta_{ij} \prod_{k \neq j, i \in U_k} (1 + \theta_{ik} - \Sigma_k) \quad \text{whenever } 1 \leq i \leq m \text{ and } i \in U_j.$$

a) Prove that  $(p_1, \dots, p_m) \in \mathcal{R}(G)$ . *Hint:* Letting  $\bar{A}_S$  denote  $\bigcap_{i \in S} \bar{A}_i$ , show that

$$\Pr(A_i | \bar{A}_S) \leq \theta_{ij} \quad \text{whenever } 1 \leq i \leq m \text{ and } i \in U_j \text{ and } S \cap U_j = \emptyset.$$

b) Also  $E_i$  in (152) is at most  $\min_{i \rightarrow j \text{ in } G} \theta_{ij} / (1 - \Sigma_j)$ . (See Theorems M and K.)

c) Improve Theorem L by showing that, if  $0 \leq \theta_j < \frac{1}{2}$ , then  $(p_1, \dots, p_m) \in \mathcal{R}(G)$  when

$$p_i = \theta_i \left( \prod_{i \rightarrow j \text{ in } G} (1 - \theta_j) \right) / \max_{i \rightarrow j \text{ in } G} (1 - \theta_j).$$

► **357.** [M20] Let  $x = \pi_{\bar{v}}$  and  $y = \pi_v$  in (155), and suppose the field of variable  $v$  is  $(p, q, r)$ . Express  $x$  and  $y$  as functions of  $p, q$ , and  $r$ .

**358.** [M20] Continuing exercise 357, prove that  $r = \max(p, q, r)$  if and only if  $x, y \geq \frac{1}{2}$ .

**359.** [20] Equations (156) and (157) should actually have been written

$$\gamma_{l \rightarrow C} = \frac{(1 - \pi_{\bar{l}})(1 - \eta_l) \prod_{l \in C' \neq C} (1 - \eta_{C \rightarrow l})}{\pi_{\bar{l}} + (1 - \pi_{\bar{l}})(1 - \eta_l) \prod_{l \in C' \neq C} (1 - \eta_{C \rightarrow l})} \quad \text{and} \quad \eta'_{C \rightarrow l} = \prod_{C \ni l' \neq l} \gamma_{l' \rightarrow C},$$

to avoid division by zero. Suggest an efficient way to implement these calculations.

**360.** [M23] Find all fixed points of the seven-clause system illustrated in (159), given that  $\pi_1 = \pi_2 = \pi_4 = 1$ . Assume also that  $\eta_l \eta_{\bar{l}} = 0$  for all  $l$ .

► **361.** [M22] Describe all fixed points  $\eta_{C \rightarrow l} = \eta'_{C \rightarrow l}$  of the equations (154), (156), (157), for which each  $\eta_{C \rightarrow l}$  and each  $\eta_l$  is either 0 or 1.

**362.** [20] Spell out the computations needed to finish Algorithm S in step S8.

► **363.** [M30] (*Lattices of partial assignments.*) A partial assignment to the variables of a satisfiability problem is called *stable* (or “valid”) if it is consistent and cannot be extended by unit propagation. In other words, it’s stable if and only if no clause is entirely false, or entirely false except for at most one unassigned literal. Variable  $x_k$  of a partial assignment is called *constrained* if it appears in a clause where  $\pm x_k$  is true but all the other literals are false (thus its value has a “reason”).

The  $3^n$  partial assignments of an  $n$ -variable problem can be represented either as strings  $x = x_1 \dots x_n$  on the alphabet  $\{0, 1, *\}$  or as sets  $L$  of strictly distinct literals. For example, the string  $x = *1*01*$  corresponds to the set  $L = \{2, \bar{4}, 5\}$ . We write  $x \prec x'$  if  $x'$  is equal to  $x$  except that  $x_k = *$  and  $x'_k \in \{0, 1\}$ ; equivalently  $L \prec L'$  if  $L' = L \cup k$  or  $L' = L \cup \bar{k}$ . Also  $x \sqsubseteq x'$  if there are  $t \geq 0$  stable partial assignments  $x^{(j)}$  with

$$x = x^{(0)} \prec x^{(1)} \prec \dots \prec x^{(t)} = x'.$$

lopsidedependency graph  
dependency graph  
Clique Local Lemma  
clique cover  
field  
Lattices of partial assignments  
partial assignment  
stable  
valid partial assignment  
consistent  
unit propagation  
constrained  
reason  
strictly distinct literals

Let  $p_1, \dots, p_n, q_1, \dots, q_n$  be probabilities, with  $p_k + q_k = 1$  for  $1 \leq k \leq n$ . Define the weight  $W(x)$  of a partial assignment to be 0 if  $x$  is unstable, otherwise

$$W(x) = \prod \{p_k \mid x_k = *\} \cdot \prod \{q_k \mid x_k \neq * \text{ and } x_k \text{ is unconstrained}\}.$$

[E. Maneva, E. Mossel, and M. J. Wainwright, in *JACM* **54** (2007), 17:1–17:41, studied general message-passing algorithms on partial assignments that are distributed with probability proportional to their weights, in the case  $p_1 = \dots = p_n = p$ , showing that survey propagation (Algorithm S) corresponds to the limit as  $p \rightarrow 1$ .]

- a) True or false: The partial assignment specified by the literals currently on the trail in step C5 of Algorithm C is stable.
  - b) What weights  $W(x)$  correspond to the clauses  $F$  in (1)?
  - c) Let  $x$  be a stable partial assignment with  $x_k = 1$ , and let  $x'$  and  $x''$  be obtained from  $x$  by setting  $x'_k \leftarrow 0$ ,  $x''_k \leftarrow *$ . True or false:  $x_k$  is unconstrained in  $x$  if and only if (i)  $x'$  is consistent; (ii)  $x'$  is stable; (iii)  $x''$  is stable.
  - d) If the only clause is  $123 = (x_1 \vee x_2 \vee x_3)$ , find all sets  $L$  such that  $L \sqsubseteq \{1, \bar{2}, \bar{3}\}$ .
  - e) What are the weights when there's only a single clause  $123 = (x_1 \vee x_2 \vee x_3)$ ?
  - f) Find clauses such that the sets  $L$  with  $L \sqsubseteq \{1, 2, 3, 4, 5\}$  are  $\emptyset, \{4\}, \{5\}, \{1, 4\}, \{2, 5\}, \{4, 5\}, \{1, 4, 5\}, \{2, 4, 5\}, \{3, 4, 5\}, \{1, 3, 4, 5\}, \{2, 3, 4, 5\}, \{1, 2, 3, 4, 5\}$ .
  - g) Let  $\mathcal{L}$  be a family of sets  $\subseteq \{1, \dots, n\}$ , closed under intersection, with the property that  $L \in \mathcal{L}$  implies  $L = L^{(0)} \prec L^{(1)} \prec \dots \prec L^{(t)} = \{1, \dots, n\}$  for some  $L^{(j)} \in \mathcal{L}$ . (The sets in (d) form one such family, with  $n = 5$ .) Construct strict Horn clauses with the property that  $L \in \mathcal{L}$  if and only if  $L \sqsubseteq \{1, \dots, n\}$ .
  - h) True or false: If  $L, L', L''$  are stable and  $L' \prec L, L'' \prec L$ , then  $L' \cap L''$  is stable.
  - i) If  $L' \sqsubseteq L$  and  $L'' \sqsubseteq L$ , prove that  $L' \cap L'' \sqsubseteq L$ .
  - j) Prove that  $\sum_{x' \sqsubseteq x} W(x') = \prod \{p_k \mid x_k = *\}$  whenever  $x$  is stable.
- **364.** [M21] A *covering assignment* is a stable partial assignment in which every assigned variable is constrained. A *core assignment* is a covering assignment  $L$  that satisfies  $L \sqsubseteq L'$  for some total assignment  $L'$ .
- a) True or false: The empty partial assignment  $L = \emptyset$  is always covering.
  - b) Find all the covering and core assignments of the clauses  $F$  in (1).
  - c) Find all the covering and core assignments of the clauses  $R'$  in (7).
  - d) Show that every satisfying assignment  $L'$  has a unique core.
  - e) The satisfying assignments form a graph, if two of them are adjacent when they differ by complementing just one literal. The connected components of this graph are called *clusters*. Prove that the elements of each cluster have the same core.
  - f) If  $L'$  and  $L''$  have the same core, do they belong to the same cluster?
- 365.** [M27] Prove that the clauses *waarden*(3, 3;  $n$ ) have a nontrivial (i.e., nonempty) covering assignment for all sufficiently large  $n$  (although they're unsatisfiable).
- **366.** [18] Preprocess the clauses  $R'$  of (7). What erp rules are generated?
- **367.** [20] Justify the erp rule (161) for elimination by resolution.
- 368.** [16] Show that subsumption and downhill resolution imply unit conditioning: Any preprocessor that does transformations 2 and 4 will also do transformation 1.
- **369.** [21] (N. Eén and A. Biere.) Suppose  $l$  appears only in clauses  $C_1, \dots, C_p$  and  $\bar{l}$  appears only in clauses  $C'_1, \dots, C'_q$ , where we have  $C_1 = (l \vee l_1 \vee \dots \vee l_r)$  and  $C'_j = (\bar{l} \vee \bar{l}_j)$  for  $1 \leq j \leq q$ . Prove that we can eliminate  $|l|$  by using the erp rule  $\bar{l} \leftarrow (l_1 \vee \dots \vee l_r)$  and replacing those  $p + q$  clauses by only  $(p - 2)r + q$  others, namely

$$\{C_1 \diamond C'_j \mid r < j \leq q\} \cup \{C_i \diamond C'_j \mid 1 < i \leq p, 1 \leq j \leq r\}.$$

Maneva  
Mossel  
Wainwright  
message-passing algorithms  
survey propagation  
trail  
Horn clauses  
covering assignment  
core assignment  
empty partial assignment  
satisfying assignment  
clusters  
*waarden*  
Preprocess  
erp rules  
elimination by resolution  
downhill resolution  
unit conditioning  
Eén  
Biere  
erp rule

(The case  $r = 1$  is especially important. In many applications—for example in the examples of fault testing, tomography, and the “Life in 4” problem about extending Fig. 35—more than half of all variable eliminations admit this simplification.)

**370.** [20] The clauses obtained by resolution might be needlessly complex even when exercise 369 doesn’t apply. For example, suppose that variable  $x$  appears only in the clauses  $(x \vee a) \wedge (x \vee \bar{a} \vee c) \wedge (\bar{x} \vee b) \wedge (\bar{x} \vee \bar{b} \vee \bar{c})$ . Resolution replaces those four clauses by three others:  $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee b \vee c)$ . Show, however, that only *two* clauses, both *binary*, would actually suffice in this particular case.

**371.** [24] By preprocessing repeatedly with transformations 1–4, and using exercise 369, prove that the 32 clauses (9) of *waerden*(3, 3; 9) are unsatisfiable.

**372.** [25] Find a “small” set of clauses that *cannot* be solved entirely via transformations 1–4 and the use of exercise 369.

**373.** [25] The answer to exercise 228 defines  $2m + \sum_{j=1}^m (j-1)^2 \approx m^3/3$  clauses in  $m^2$  variables that suffice to refute the anti-maximal-element axioms of (99)–(101). Algorithm L needs exponential time to handle these clauses, according to Theorem R; and experiments show that they are bad news for Algorithm C too. Show, however, that preprocessing with transformations 1–4 will rapidly prove them unsatisfiable.

► **374.** [32] Design data structures for the efficient representation of clauses within a SAT preprocessor. Also design algorithms that (a) resolve clauses  $C$  and  $C'$  with respect to a variable  $x$ ; (b) find all clauses  $C'$  that are subsumed by a given clause  $C$ ; (c) find all clauses  $C'$  that are self-subsumed by a given clause  $C$  and a literal  $l \in C$ .

**375.** [21] Given  $|l|$ , how can one test efficiently whether or not the special situation in exercise 369 applies, using (and slightly extending) the data structures of exercise 374?

► **376.** [32] After a preprocessor has found a transformation that reduces the current set of clauses, it is supposed to try again and look for further simplifications. (See (160).) Suggest methods that will avoid unnecessary repetition of previous work, by using (and slightly extending) the data structures of exercise 374.

**377.** [22] (V. Vassilevska Williams.) If  $G$  is a graph with  $n$  vertices and  $m$  edges, construct a 2SAT problem  $F$  with  $3n$  variables and  $6m$  clauses, such that  $G$  contains a triangle (a 3-clique) if and only if  $F$  has a failed literal.

**378.** [20] (*Blocked clause elimination.*) Clause  $C = (l \vee l_1 \vee \dots \vee l_q)$  is said to be blocked by the literal  $l$  if every clause that contains  $\bar{l}$  also contains either  $\bar{l}_1$  or  $\dots$  or  $\bar{l}_q$ . Exercise 161(b) proves that clause  $C$  can be removed without making an unsatisfiable problem satisfiable. Show that this transformation requires an erp rule, even though it doesn’t eliminate any of the variables. What erp rule works?

► **379.** [20] (*Blocked self-subsumption.*) Consider the clause  $(a \vee b \vee c \vee d)$ , and suppose that every clause containing  $\bar{a}$  but not  $\bar{b}$  nor  $\bar{c}$  also contains  $d$ . Show that we can then shorten the clause to  $(b \vee c \vee d)$  without affecting satisfiability. Is an erp rule needed?

**380.** [21] Sometimes we can use self-subsumption backwards, for example by weakening the clause  $(l_1 \vee l_2 \vee l_3)$  to  $(l_1 \vee \dots \vee l_k)$  if each intermediate replacement of  $(l_1 \vee \dots \vee l_j)$  by  $(l_1 \vee \dots \vee l_{j-1})$  is justifiable for  $3 < j \leq k$ . Then, if we’re lucky, the clause  $(l_1 \vee \dots \vee l_k)$  is weak enough to be eliminated; in such cases we are allowed to eliminate  $(l_1 \vee l_2 \vee l_3)$ .

a) Show that  $(a \vee b \vee c)$  can be eliminated if it is accompanied by the additional clauses  $(a \vee b \vee \bar{d})$ ,  $(a \vee d \vee e)$ ,  $(b \vee d \vee \bar{e})$ .

b) Show that  $(a \vee b \vee c)$  can also be eliminated when accompanied by  $(a \vee b \vee \bar{d})$ ,  $(a \vee \bar{c} \vee \bar{d})$ ,  $(b \vee d \vee \bar{e})$ ,  $(b \vee \bar{c} \vee \bar{e})$ , provided that no other clauses contain  $\bar{c}$ .

fault testing  
tomography  
Life in 4  
resolution  
*waerden*  
anti-maximal-element  
data structures  
preprocessor  
resolution, implementation of  
subsumption, implementation of  
self-subsumed  
Vassilevska Williams  
clique  
failed literal  
triangle-free graph  
Blocked clause elimination  
erp rule  
Blocked self-subsumption  
self-subsumption

c) What erp rules, if any, are needed for those eliminations?

**381.** [22] Combining exercises 379 and 380, show that any one of the clauses in

$$(\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge \cdots \wedge (\bar{x}_{n-1} \vee x_n) \wedge (\bar{x}_n \vee x_1)$$

can be removed if there are no other clauses with negative literals. State the erp rules.

**382.** [30] Although the techniques in the preceding exercises are computationally difficult to apply, show that a lookahead forest based on the dependency digraph can be used to discover some of those simplifications efficiently.

► **383.** [23] (*Inprocessing.*) A SAT solver can partition its database of current clauses into two parts, the “hard” clauses  $\Phi$  and the “soft” clauses  $\Psi$ . Initially  $\Psi$  is empty, while  $\Phi$  is  $F$ , the set of all input clauses. Four kinds of changes are subsequently allowed:

- **Learning.** We can append a new soft clause  $C$ , provided that  $\Phi \cup \Psi \cup C$  is satisfiable whenever  $\Phi \cup \Psi$  is satisfiable.

- **Forgetting.** We can discard (purge) any soft clause.

- **Hardening.** We can reclassify any soft clause and call it hard.

- **Softening.** We can reclassify any hard clause  $C$  and call it soft, provided that  $\Phi$  is satisfiable whenever  $\Phi \setminus C$  is satisfiable. In this case we also should output any necessary erp rules, which change the settings of variables in such a way that any solution to  $\Phi \setminus C$  becomes a solution to  $\Phi$ .

- a) Prove that, throughout any such procedure,  $F$  is satisfiable  $\iff \Phi$  is satisfiable  $\iff \Phi \cup \Psi$  is satisfiable.
- b) Furthermore, given any solution to  $\Phi$ , we obtain a solution to  $F$  by applying the erp rules in reverse order.
- c) What is wrong with the following scenario? Start with one hard clause,  $(x)$ , and no soft clauses. Reclassify  $(x)$  as soft, using the erp rule  $x \leftarrow 1$ . Then append a new soft clause  $(\bar{x})$ .
- d) If  $C$  is certifiable for  $\Phi$  (see exercise 385), can we safely learn  $C$ ?
- e) If  $C$  is certifiable for  $\Phi \setminus C$ , can we safely forget  $C$ ?
- f) In what cases is it legitimate to discard a clause, hard or soft, that is subsumed by another clause, hard or soft?
- g) In what cases is self-subsumption permissible?
- h) Explain how to eliminate all clauses that involve a particular variable  $x$ .
- i) Show that, if  $z$  is a new variable, we can safely learn the three new soft clauses  $(x \vee z)$ ,  $(y \vee z)$ ,  $(\bar{x} \vee \bar{y} \vee \bar{z})$  in Tseytin’s concept of extended resolution.

**384.** [25] Continuing the previous exercise, show that we can always safely forget any clause  $C$  that contains a literal  $l$  for which  $C \diamond C'$  is certifiable for  $\Phi \setminus C$  whenever  $C' \in \Phi$  contains  $\bar{l}$ . What erp rule is appropriate?

**385.** [22] Clause  $C$  is called *certifiable* for a set of clauses  $F$  if  $F \wedge \overline{C} \vdash_1 \epsilon$ , as in (119). It is said to be *absorbed* by  $F$  if it is nonempty and  $F \wedge \overline{C} \setminus l \vdash_1 l$  for every  $l \in C$ , or if it is empty and  $F \vdash_1 \epsilon$ . (Every clause of  $F$  is obviously absorbed by  $F$ .)

- a) True or false: If  $C$  is absorbed by  $F$ , it is certifiable for  $F$ .
- b) Which of  $\{\bar{1}, \bar{1}2, \bar{1}23\}$  are implied by, certifiable for, or absorbed by  $R'$  in (7)?
- c) If  $C$  is certifiable for  $F$  and if all clauses of  $F$  are absorbed by  $F'$ , prove that  $C$  is certifiable for  $F'$ .
- d) If  $C$  is absorbed by  $F$  and if all clauses of  $F$  are absorbed by  $F'$ , prove that  $C$  is absorbed by  $F'$ .

lookahead forest  
 dependency digraph  
 Inprocessing  
 hard  
 soft  
 Learning  
 Forgetting  
 purge  
 erp rules  
 certifiable  
 subsumed  
 self-subsumption  
 variable elimination  
 Tseytin  
 extended resolution  
 erp rule  
 certifiable  
 absorbed  
 asymmetric tautology, see certifiable clause

- **386.** [M25] Let Algorithm  $C_0$  be a variant of Algorithm  $C$  that (i) makes all decisions at random; (ii) never forgets a learned clause; and (iii) restarts whenever a new clause has been learned. (Thus, step C5 ignores  $M_p$  and  $M_f$ ; step C6 chooses  $l$  uniformly at random from among the  $2(n-F)$  currently unassigned literals; step C8 backjumps while  $F > i_1$ , instead of while  $F > i_{d+1}$ ; and after step C9 has stored a new clause, with  $d > 0$ , it simply sets  $d \leftarrow 0$  and returns to C2. The data structures **HEAP**, **OVAL**, and **ACT** are no longer used.) We will prove that Algorithm  $C_0$  is, nevertheless, quite powerful.

In the remainder of this exercise,  $F$  denotes the set of clauses known by Algorithm  $C_0$ , both original and learned; in particular, the unit clauses of  $F$  will be the first literals  $L_0, L_1, \dots, L_{i_1-1}$  on the trail. If  $C$  is any clause and if  $l \in C$ , we define

$$\text{score}(F, C, l) = \begin{cases} \infty, & \text{if } F \wedge \overline{C \setminus l} \vdash l; \\ |\{l' \mid F \wedge \overline{C \setminus l} \vdash l'\}|, & \text{otherwise.} \end{cases}$$

Thus  $\text{score}(F, C, l)$  represents the total number of literals on the trail after making all the unforced decisions of  $\overline{C \setminus l}$ , if no conflict arises. We say that Algorithm  $C_0$  performs a “helpful round” for  $C$  and  $l$  if (i) every decision literal belongs to  $\overline{C}$ ; and (ii)  $l$  is chosen as a decision literal only if the other elements of  $\overline{C}$  are already in the trail.

- Let  $C$  be certifiable for  $F$ , and suppose that  $\text{score}(F, C, l) < \infty$  for some  $l \in C$ . Prove that if  $F'$  denotes  $F$  together with a clause learned on a helpful round, then  $\text{score}(F', C, l) > \text{score}(F, C, l)$ .
  - Furthermore  $\text{score}(F', C, l) \geq \text{score}(F, C, l)$  after an unhelpful round.
  - Therefore  $C$  will be absorbed by the set  $F'$  of known clauses after at most  $|C|n$  helpful rounds have occurred.
  - If  $|C| = k$ , show that  $\Pr(\text{helpful round}) \geq (k-1)!/(2n)^k \geq 1/(4n^k)$ .
  - Consequently, by exercise 385(c), if there exists a certificate of unsatisfiability  $(C_1, \dots, C_t)$  for a family of clauses  $F$  with  $n$  variables, Algorithm  $C_0$  will prove  $F$  unsatisfiable after learning an average of  $\mu \leq 4 \sum_{i=1}^t |C_i| n^{1+|C_i|}$  clauses. (And it will q.s. need to learn at most  $\mu \ln n \ln \ln n$  clauses, by exercise MPR-102.)
- **387.** [21] Graph  $G$  is said to be *embedded* in graph  $G'$  if every vertex  $v$  of  $G$  corresponds to a distinct vertex  $v'$  of  $G'$ , where  $u' - v'$  in  $G'$  whenever  $u - v$  in  $G$ . Explain how to construct clauses that are satisfiable if and only if  $G$  can be embedded in  $G'$ .

**388.** [20] Show that the problems of deciding whether or not a given graph  $G$  (a) contains a  $k$ -clique, (b) can be  $k$ -colored, or (c) has a Hamiltonian cycle can all be regarded as graph embedding problems.

- **389.** [22] In this  $4 \times 4$  diagram, it's possible to trace out the phrase 'THE ART OF COMPUTER PROGRAMMING' by making only king moves and knight moves, *except* for the final step from N to G.

|   |   |   |   |
|---|---|---|---|
| N | T | E | F |
| H | I | R | □ |
| U | P | O | A |
| M | M | C | G |

Rearrange the letters so that the entire phrase can be traced.

- **390.** [23] Let  $G$  be a graph with vertices  $V$ , edges  $E$ ,  $|E| = m$ ,  $|V| = n$ , and  $s, t \in V$ .
- Construct  $O(kn)$  clauses that are satisfiable if and only if there's a path of length  $k$  or less from  $s$  to  $t$ , given  $k$ .
  - Construct  $O(m)$  clauses that are satisfiable if and only if there's at least one path from  $s$  to  $t$ .
  - Construct  $O(n^2)$  clauses that are satisfiable if and only if  $G$  is connected.
  - Construct  $O(km)$  clauses that are *unsatisfiable* if and only if there's a path of length  $k$  or less from  $s$  to  $t$ , given  $k$ .

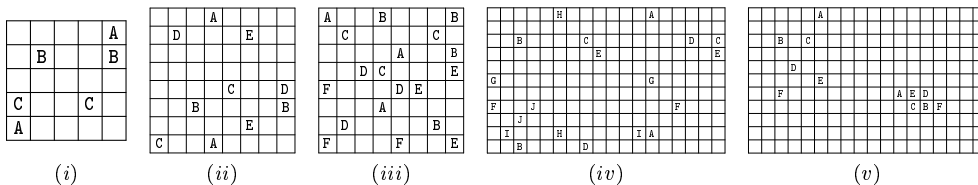
helpful round  
certificate of unsatisfiability  
embedded  
homomorphic embedding  
clique  
Hamiltonian cycle  
king moves  
knight moves  
path  
reachability in a graph  
connected  
path

- e) Construct  $O(m)$  clauses that are *unsatisfiable* if and only if there's at least one path from  $s$  to  $t$ .
- f) Construct  $O(m)$  clauses that are *unsatisfiable* if and only if  $G$  is connected. (This construction is much better than (c), in a sparse graph.)

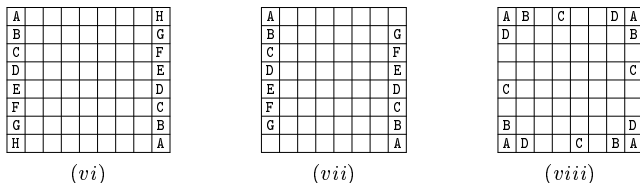
**391.** [M25] The values of two integer variables satisfy  $0 \leq x, y < d$ , and they are to be represented as  $l$ -bit quantities  $x_{l-1} \dots x_0, y_{l-1} \dots y_0$ , where  $l = \lceil \lg d \rceil$ . Specify three different ways to encode the relation  $x \neq y$ :

- a) Let  $x = (x_{l-1} \dots x_0)_2$  and  $y = (y_{l-1} \dots y_0)_2$ ; and let the encoding enforce the conditions  $(x_{l-1} \dots x_0)_2 < d$ , and  $(y_{l-1} \dots y_0)_2 < d$ , as well as ensuring that  $x \neq y$  by introducing  $2l + 1$  additional clauses in  $l$  auxiliary variables.
- b) Like (a), but there are  $d$  additional clauses (not  $2l + 1$ ), and no auxiliaries.
- c) All bit patterns  $x_{l-1} \dots x_0$  and  $y_{l-1} \dots y_0$  are valid, but some values might have two different patterns. The encoding has  $d$  clauses and no auxiliary variables.

**392.** [22] The blank spaces in the following diagrams can be filled with letters in such a way that all occurrences of the same letter are rookwise connected:



- a) Demonstrate how to do it. (Puzzle (i) is easy; the others less so.)
- b) Similarly, solve the following puzzles — but use *kingwise* connectedness instead.



- c) Construct clauses with which a SAT solver can solve general puzzles of this kind: Given a graph  $G$  and disjoint sets of vertices  $T_1, T_2, \dots, T_t$ , a solution should exhibit disjoint *connected* sets of vertices  $S_1, S_2, \dots, S_t$ , with  $T_j \subseteq S_j$  for  $1 \leq j \leq t$ .

**393.** [25] (T. R. Dawson, 1911.) Show that it's possible for each white piece in the accompanying chess diagram to capture the corresponding black piece, via a path that doesn't intersect any of the other paths. How can SAT help to solve this problem?



**394.** [25] One way to encode the at-most-one constraint  $S_{\leq 1}(y_1, \dots, y_p)$  is to introduce  $l = \lceil \lg p \rceil$  auxiliary variables together with the following  $nl + n - 2^l$  clauses, which essentially "broadcast" the value of  $j$  when  $y_j$  becomes true:

$$(\bar{y}_j \vee (-1)^{b_t} a_t) \quad \text{for } 1 \leq j \leq p, 1 \leq t \leq q = \lfloor \lg(2p - j) \rfloor, \text{ where } 2p - j = (1b_1 \dots b_q)_2.$$

For example, the clauses when  $p = 3$  are  $(\bar{y}_1 \vee a_1) \wedge (\bar{y}_1 \vee \bar{a}_2) \wedge (\bar{y}_2 \vee a_1) \wedge (\bar{y}_2 \vee a_2) \wedge (\bar{y}_3 \vee \bar{a}_1)$ .

Experiment with this encoding by applying it to Langford's problem, using it in place of (13) whenever  $p \geq 7$ .

**395.** [20] What clauses should replace (15), (16), and (17) if we want to use the order encoding for a graph coloring problem?

connected  
 encode  
 rookwise connected  
 multicommodity flow  
 routing, disjoint  
 connection puzzles  
*kingwise* connectedness  
 Dawson  
 chess diagram  
 nonintersecting paths  
 at-most-one  
 auxiliary variables  
 broadcast  
 Langford's problem  
 order encoding

- **396.** [23] (*Double clique hints.*) If  $x$  has one of the  $d$  values  $\{0, 1, \dots, d-1\}$ , we can represent it binarywise with respect to *two* different orderings by letting  $x^j = [x \geq j]$  and  $\hat{x}^j = [x\pi \geq j]$  for  $1 \leq j < d$ , where  $\pi$  is any given permutation. For example, if  $d = 4$  and  $(0\pi, 1\pi, 2\pi, 3\pi) = (2, 3, 0, 1)$ , the representations  $x^1x^2x^3:\hat{x}^1\hat{x}^2\hat{x}^3$  of 0, 1, 2, and 3 are respectively 000:110, 100:111, 110:000, and 111:100. This double ordering allows us to encode graph coloring problems by including not only the hints (162) but also

$$\overline{(\hat{v}_1^{d-k+1} \vee \dots \vee \hat{v}_k^{d-k+1})} \wedge (\hat{v}_1^{k-1} \vee \dots \vee \hat{v}_k^{k-1}),$$

whenever the vertices  $\{v_1, \dots, v_k\}$  form a  $k$ -clique.

Explain how to construct clauses for this encoding, and experiment with coloring the  $n \times n$  queens graph when  $(0\pi, 1\pi, 2\pi, 3\pi, 4\pi, \dots) = (0, d-1, 1, d-2, 2, \dots)$  is the inverse of the organ-pipe permutation.

- **397.** [22] (N. Tamura, 2014.) Suppose  $x_0, x_1, \dots, x_{p-1}$  are integer variables with the range  $0 \leq x_i < d$ , represented in order encoding by Boolean variables  $x_i^j = [x_i \geq j]$  for  $0 \leq i < p$  and  $1 \leq j < d$ . Show that the *all-different* constraint, “ $x_i \neq x_j$  for  $0 \leq i < j < p$ ,” can be nicely encoded by introducing auxiliary integer variables  $y_0, y_1, \dots, y_{d-1}$  with the range  $0 \leq y_j < p$ , represented in order encoding by Boolean variables  $y_j^i = [y_j \geq i]$  for  $1 \leq i < p$  and  $0 \leq j < d$ , and by devising clauses to enforce the condition  $x_i = j \implies y_j = i$ . Furthermore, hints analogous to (162) can be given.

**398.** [18] Continuing exercise 397, what’s an appropriate way to enforce the all-different constraint when  $x_0, \dots, x_{p-1}$  are represented in the *direct* encoding?

- **399.** [23] If the variables  $u$  and  $v$  range over  $d$  values  $\{1, \dots, d\}$ , it’s natural to encode them directly as sequences  $u_1 \dots u_d$  and  $v_1 \dots v_d$ , where  $u_i = [u = i]$  and  $v_j = [v = j]$ , using the at-least-one clauses (15) and the at-most-one clauses (17). A *binary constraint* tells us which pairs  $(i, j)$  are legal; for example, the graph-coloring constraint says that  $i \neq j$  when  $i$  and  $j$  are the colors of adjacent vertices in some graph.

One way to specify such a constraint is to assert the *preclusion clauses*  $(\bar{u}_i \vee \bar{v}_j)$  for all *illegal* pairs  $(i, j)$ , as we did for graph coloring in (16). But there’s also another general way: We can assert the *support clauses*

$$\bigwedge_{i=1}^d (\bar{u}_i \vee \bigvee \{v_j \mid (i, j) \text{ is legal}\}) \wedge \bigwedge_{j=1}^d (\bar{v}_j \vee \bigvee \{u_i \mid (i, j) \text{ is legal}\})$$

instead. Graph coloring with  $d$  colors would then be represented by clauses such as  $(\bar{u}_3 \vee v_1 \vee v_2 \vee v_4 \vee \dots \vee v_d)$ , when  $u$  and  $v$  are adjacent.

- Suppose  $t$  of the  $d^2$  pairs  $(i, j)$  are legal. How many preclusion clauses are needed? How many support clauses?
- Prove that the support clauses are always at least as strong as the preclusion clauses, in the sense that all consequences of the preclusion clauses under unit propagation are also consequences of the support clauses under unit propagation, given any partial assignment to the binary variables  $\{u_1, \dots, u_d, v_1, \dots, v_d\}$ .
- Conversely, in the case of the graph-coloring constraint, the preclusion clauses are also at least as strong as the support clauses (hence equally strong).
- However, exhibit a binary constraint for which the support clauses are strictly stronger than the preclusion clauses.

**400.** [25] Experiment with preclusion clauses versus support clauses by applying them to the  $n$  queens problem. Use Algorithms L, C, and W for comparison.

**401.** [16] If  $x$  has the unary representation  $x^1x^2 \dots x^{d-1}$ , what is the unary representation of (a)  $y = \lceil x/2 \rceil$ ? (b)  $z = \lfloor (x+1)/3 \rfloor$ ?

Double clique hints  
order encoding  
graph coloring  
hints  
redundant representation  
clique  
queens graph  
organ-pipe permutation  
Tamura  
order encoding  
all-different  
*direct* encoding  
direct encoding  
at-least-one  
at-most-one  
binary constraint  
graph-coloring  
preclusion clauses  
conflict clauses, see also preclusion clauses  
support clauses  
unit propagation  
 $n$  queens problem  
unary representation



**402.** [18] If  $x$  has the unary representation  $x^1 x^2 \dots x^{d-1}$ , encode the further condition that  $x$  is (a) even; (b) odd.

**403.** [20] Suppose  $x, y, z$  have the order encoding, with  $0 \leq x, y, z < d$ . What clauses enforce (a)  $\min(x, y) \leq z$ ? (b)  $\max(x, y) \leq z$ ? (c)  $\min(x, y) \geq z$ ? (d)  $\max(x, y) \geq z$ ?

- ▶ **404.** [21] Continuing exercise 403, encode the condition  $|x - y| \geq a$ , for a given constant  $a \geq 1$ , using either (a)  $d$  clauses of length  $\leq 4$  and no auxiliary variables; or (b)  $2d - O(a)$  clauses of length  $\leq 3$ , and one auxiliary variable.
- ▶ **405.** [M23] The purpose of this exercise is to encode the constraint  $ax + by \leq c$ , when  $a, b, c$  are integer constants, assuming that  $x, y$  are order-encoded with range  $[0 \dots d)$ .
  - a) Prove that it suffices to consider cases where  $a, b, c > 0$ .
  - b) Exhibit a suitable encoding for the special case  $13x - 8y \leq 7, d = 8$ .
  - c) Exhibit a suitable encoding for the special case  $13x - 8y \geq 1, d = 8$ .
  - d) Specify an encoding that works for general  $a, b, c, d$ .

**406.** [M24] Order-encode (a)  $xy \leq a$  and (b)  $xy \geq a$ , when  $a$  is an integer constant.

- ▶ **407.** [M22] If  $x, y, z$  are order-encoded, with  $0 \leq x, y < d$  and  $0 \leq z < 2d - 1$ , the clauses

$$\bigwedge_{k=1}^{2d-2} \bigwedge_{j=\max(0, k+1-d)}^k (\bar{x}^j \vee \bar{y}^{k-j} \vee z^k)$$

are satisfiable if and only if  $x + y \leq z$ ; this is the basic idea underlying (20). Another way to encode the same relation is to introduce new order-encoded variables  $u$  and  $v$ , and to construct clauses for the relations  $\lfloor x/2 \rfloor + \lfloor y/2 \rfloor \leq u$  and  $\lceil x/2 \rceil + \lceil y/2 \rceil \leq v$ , recursively using methods for numbers less than  $\lceil d/2 \rceil$  and  $\lfloor d/2 \rfloor + 1$ . Then we can finish the job by letting  $z^1 = v^1, z^{2d-2} = v^d$  ( $d$  even) or  $u^{d-1}$  ( $d$  odd), and appending the clauses

$$(\bar{u}^j \vee z^{2j}) \wedge (\bar{v}^{j+1} \vee z^{2j}) \wedge (\bar{u}^j \vee \bar{v}^{j+1} \vee z^{2j+1}), \quad \text{for } 1 \leq j \leq d - 2.$$

- a) Explain why the alternative method is valid.
  - b) For what values of  $d$  does that method produce fewer clauses?
  - c) Consider analogous methods for the relation  $x + y \geq z$ .
- ▶ **408.** [25] (*Open shop scheduling.*) Consider a system of  $m$  machines and  $n$  jobs, together with an  $m \times n$  matrix of nonnegative integer weights  $W = (w_{ij})$  that represent the amount of uninterrupted time on machine  $i$  that is needed by job  $j$ .

The open shop scheduling problem seeks a way to get all the work done in  $t$  units of time, without assigning two jobs simultaneously to the same machine and without having two machines simultaneously assigned to the same job. We want to minimize  $t$ , which is called the “makespan” of the schedule.

For example, suppose  $m = n = 3$  and  $W = \begin{pmatrix} 703 & & \\ 172 & & \\ 235 & & \end{pmatrix}$ . A “greedy” algorithm, which repeatedly fills the lexicographically smallest time slot  $(t, i, j)$  such that  $w_{ij} > 0$  but neither machine  $i$  nor job  $j$  have yet been scheduled at time  $t$ , achieves a makespan of 12 with the following schedule:

|     |  |    |    |  |  |    |  |    |
|-----|--|----|----|--|--|----|--|----|
| M1: |  |    | J1 |  |  | J3 |  |    |
| M2: |  |    | J2 |  |  | J1 |  | J3 |
| M3: |  | J3 |    |  |  | J2 |  | J1 |

- a) Is 12 the optimum makespan for this  $W$ ?

auxiliary variables  
linear inequalities  
recursively  
Open shop scheduling  
job shop problems  
makespan  
“greedy” algorithm

- b) Prove that the greedy algorithm always produces a schedule whose makespan is less than  $(\max_{i=1}^m \sum_{j=1}^n w_{ij}) + (\max_{j=1}^n \sum_{i=1}^m w_{ij})$ , unless  $W$  is entirely zero.
- c) Suppose machine  $i$  begins to work on job  $j$  at time  $s_{ij}$ , when  $w_{ij} > 0$ . What conditions should these starting times satisfy, in order to achieve the makespan  $t$ ?
- d) Show that the *order encoding* of these variables  $s_{ij}$  yields SAT clauses that nicely represent any open shop scheduling problem.
- e) Let  $\lfloor W/k \rfloor$  be the matrix obtained by replacing each element  $w_{ij}$  of  $W$  by  $\lfloor w_{ij}/k \rfloor$ . Prove that if the open shop scheduling problem for  $\lfloor W/k \rfloor$  and  $t$  is unsatisfiable, so is the problem for  $W$  and  $kt$ .
- **409.** [M26] Continuing exercise 408, find the best makespans in the following cases:
- $m = 3$ ,  $n = 3r + 1$ ;  $w_{1j} = w_{2(r+j)} = w_{3(2r+j)} = a_j$  for  $1 \leq j \leq r$ ;  $w_{1n} = w_{2n} = w_{3n} = \lfloor (a_1 + \dots + a_r)/2 \rfloor$ ; otherwise  $w_{ij} = 0$ . (The positive integers  $a_j$  are given.)
  - $m = 4$ ,  $n = r + 2$ ;  $w_{1j} = (r + 1)a_j$  and  $w_{2j} = 1$  for  $1 \leq j \leq r$ ;  $w_{2(n-1)} = w_{2n} = (r + 1)\lfloor (a_1 + \dots + a_r)/2 \rfloor$ ;  $w_{3(n-1)} = w_{4n} = w_{2n} + r$ ; otherwise  $w_{ij} = 0$ .
  - $m = n$ ;  $w_{jj} = n - 2$ ,  $w_{jn} = w_{nj} = 1$  for  $1 \leq j < n$ ; otherwise  $w_{ij} = 0$ .
  - $m = 2$ ;  $w_{1j} = a_j$  and  $w_{2j} = b_j$  for  $1 \leq j \leq n$ , where  $a_1 + \dots + a_n = b_1 + \dots + b_n = s$  and  $a_j + b_j \leq s$  for  $1 \leq j \leq n$ .
- 410.** [24] Exhibit clauses for the constraint  $13x - 8y \leq 7$  when  $x$  and  $y$  are *log-encoded* as 3-bit integers  $x = (x_2x_1x_0)_2$  and  $y = (y_2y_1y_0)_2$ . (Compare with exercise 405(b).)
- **411.** [25] If  $x = (x_m \dots x_1)_2$ ,  $y = (y_n \dots y_1)_2$ , and  $z = (z_{m+n} \dots z_1)_2$  stand for binary numbers, the text explains how to encode the relation  $xy = z$  with fewer than  $20mn$  clauses, using Napier–Dadda multiplication. Explain how to encode the relations  $xy \leq z$  and  $xy \geq z$  with fewer than  $9mn$  and  $11mn$  clauses, respectively.
- 412.** [40] Experiment with the encoding of somewhat large numbers by using a radix- $d$  representation in which each digit has the order encoding.
- 413.** [M20] How many clauses will remain after the auxiliary variables  $a_1, \dots, a_{n-1}$  of (169) have been eliminated by resolution?
- **414.** [M22] Generalize (169) to an encoding of lexicographic order on  $d$ -ary vectors,  $(x_1 \dots x_n)_d \leq (y_1 \dots y_n)_d$ , where each  $x_k = x_k^1 + \dots + x_k^{d-1}$  and  $y_k = y_k^1 + \dots + y_k^{d-1}$  has the order encoding. What modifications to your construction will encode the *strict* relation  $x_1 \dots x_n < y_1 \dots y_n$ ?
- 415.** [M22] Find all CNF formulas for the function  $(x_1 \oplus y_1) \vee \dots \vee (x_n \oplus y_n)$ .
- 416.** [20] Encode the condition ‘if  $x_1 \dots x_n = y_1 \dots y_n$  then  $u_1 \dots u_m = v_1 \dots v_m$ ’, using  $2m + 2n + 1$  clauses and  $n + 1$  auxiliary variables. *Hint:*  $2n$  of the clauses are in (172).
- 417.** [21] Continuing exercise 42, what is the Tseytin encoding of the ternary mux operation ‘ $s \leftarrow t? u: v$ ’? Use it to justify the translation of branching programs via (174).
- 418.** [23] Use a branching program to construct clauses that are satisfiable if and only if  $(x_{ij})$  is an  $m \times n$  Boolean matrix whose rows satisfy the hidden weighted bit function  $h_n$  and whose columns satisfy the complementary function  $\bar{h}_m$ . In other words,
- $$r_i = \sum_{j=1}^n x_{ij}, \quad c_j = \sum_{i=1}^m x_{ij}, \quad \text{and} \quad x_{ir_i} = 1, \quad x_{c_j j} = 0, \quad \text{assuming that } x_{i0} = x_{0j} = 0.$$
- 419.** [M21] If  $m, n \geq 3$ , find (by hand) all solutions to the problem of exercise 418 such that (a)  $\sum x_{ij} = m + 1$  (the minimum); (b)  $\sum x_{ij} = mn - n - 1$  (the maximum).
- 420.** [18] Derive (175) mechanically (that is, “without thinking”) from the Boolean chain  $s \leftarrow x_1 \oplus x_2$ ,  $c \leftarrow x_1 \wedge x_2$ ,  $t \leftarrow s \oplus x_3$ ,  $c' \leftarrow s \wedge x_3$ , requiring  $c = c' = 0$ .

order encoding  
 linear inequalities  
 log-encoded  
 Napier  
 Dadda  
 multiplication  
 radix- $d$  representation  
 order encoding  
 auxiliary variables  
 eliminated  
 resolution  
 lexicographic order  
 order encoding  
 CNF  
 Tseytin encoding  
 mux operation  
 branching programs  
 hidden weighted bit function  
 Boolean chain

- 421.** [18] Derive (176) mechanically from the branching program  $I_5 = (\bar{1} ? 4 : 3)$ ,  $I_4 = (\bar{2} ? 1 : 2)$ ,  $I_3 = (\bar{2} ? 2 : 0)$ ,  $I_2 = (\bar{3} ? 1 : 0)$ , beginning at  $I_5$ .
- 422.** [11] What does unit propagation deduce when the additional clause  $(x_1)$  or  $(x_2)$  is appended to (a)  $F$  in (175)? (b)  $G$  in (176)?
- 423.** [22] A representation  $F$  that satisfies a condition like (180) but with  $l$  replaced by  $\epsilon$  can be called “weakly forcing.” Exercise 422 shows that (175) and (176) are weakly forcing. Does the BDD of *every* function define a weakly forcing encoding, via (173)?
- **424.** [20] The dual of the Pi function has the prime clauses  $\{\bar{1}\bar{2}\bar{3}, \bar{1}\bar{3}\bar{4}, \bar{2}\bar{3}\bar{4}, 234, 12\}$  (see 7.1.1–(30)). Can any of them be omitted from a forcing representation?
- 425.** [18] A clause with exactly one positive literal is called a definite Horn clause, and Algorithm 7.1.1C computes the “core” of such clauses. If  $F$  consists of definite Horn clauses, prove that  $x$  is in the core if and only if  $F \vdash_1 x$ , if and only if  $F \wedge (\bar{x}) \vdash_1 \epsilon$ .
- **426.** [M20] Suppose  $F$  is a set of clauses that represent  $f(x_1, \dots, x_n)$  using auxiliary variables  $\{a_1, \dots, a_m\}$  as in (170), where  $m > 0$ . Let  $G$  be the clauses that result after variable  $a_m$  has been eliminated as in (112).
- True or false: If  $F$  is forcing then  $G$  is forcing.
  - True or false: If  $F$  is not forcing then  $G$  is not forcing.
- 427.** [M30] Exhibit a function  $f(x_1, \dots, x_n)$  for which every set of forcing clauses that uses no auxiliary variables has size  $\Omega(3^n/n^2)$ , although  $f$  can actually be represented by a *polynomial* number of forcing clauses when auxiliary variables are introduced. *Hint:* See exercise 7.1.1–116.
- 428.** [M27] A generic graph  $G$  on vertices  $\{1, \dots, n\}$  can be characterized by  $\binom{n}{2}$  Boolean variables  $X = \{x_{ij} \mid 1 \leq i < j \leq n\}$ , where  $x_{ij} = [i \text{---} j \text{ in } G]$ . Properties of  $G$  can therefore be regarded as Boolean functions,  $f(X)$ .
- Let  $f_{nd}(X) = [\chi(G) \leq d]$ ; that is,  $f_{nd}$  is true if and only if  $G$  has a  $d$ -coloring. Construct clauses  $F_{nd}$  that represent the function  $f_{nd}(X) \vee y$ , using auxiliary variables  $Z = \{z_{jk} \mid 1 \leq j \leq n, 1 \leq k \leq d\}$  that mean “vertex  $j$  has color  $k$ .”
  - Let  $G_{nd}$  be a forcing representation of the Boolean function  $F_{nd}(X, y, Z)$ , and suppose that  $G_{nd}$  has  $M$  clauses in  $N$  variables. (These  $N$  variables should include the  $\binom{n}{2} + 1 + nd$  variables of  $F_{nd}$ , along with an arbitrary number of additional auxiliaries.) Explain how to construct a monotone Boolean chain of cost  $O(MN^2)$  for the function  $\bar{f}_{nd}$  (see exercise 7.1.2–84), given the clauses of  $G_{nd}$ .
- Note:* Noga Alon and Ravi B. Boppana, *Combinatorica* **7** (1987), 1–22, proved that every monotone chain for this function has length  $\exp \Omega((n/\log n)^{1/3})$  when  $d + 1 = \lfloor (n/\lg n)^{2/3}/4 \rfloor$ . Hence  $M$  and  $N$  cannot be of polynomial size.
- 429.** [22] Prove that Bailleux and Boufkhad’s clauses (20), (21) are forcing: If any  $r$  of the  $x$ ’s have been set to 1, then unit propagation will force all the others to 0.
- 430.** [25] Similarly, Sinz’s clauses (18) and (19) are forcing.
- **431.** [20] Construct efficient, forcing clauses for the relation  $x_1 + \dots + x_m \leq y_1 + \dots + y_n$ .
- 432.** [24] Exercise 404 gives clauses for the relation  $|x - y| \geq a$ . Are they forcing?
- 433.** [25] Are the lexicographic-constraint clauses in (169) forcing?
- 434.** [21] Let  $L_l$  be the language defined by the regular expression  $0^*1^l0^*$ ; in other words, the binary string  $x_1 \dots x_n$  is in  $L_l$  if and only if it consists of zero or more 0s followed by exactly  $l$  1s followed by zero or more 0s.
- Explain why the following clauses are satisfiable if and only if  $x_1 \dots x_n \in L_l$ :
    - $(\bar{p}_k \vee \bar{x}_k)$ ,  $(\bar{p}_k \vee p_{k-1})$ , and  $(\bar{p}_{k-1} \vee x_k \vee p_k)$  for  $1 \leq k \leq n$ , also  $(p_0)$ ; (ii)  $(\bar{q}_k \vee \bar{x}_k)$ ,

branching program  
 weakly forcing  
 BDD  
 dual  
 Pi function  
 prime clauses  
 forcing representation  
 definite Horn clause  
 Horn core  
 eliminated  
 auxiliary variables  
 generic graph  
 forcing  
 Alon  
 Boppana  
 Bailleux  
 Boufkhad  
 forcing  
 unit propagation  
 Sinz  
 equal sums  
 regular expression

$(\bar{q}_k \vee q_{k+1})$ , and  $(\bar{q}_{k+1} \vee x_k \vee q_k)$  for  $1 \leq k \leq n$ , also  $(q_{n+1})$ ; (iii)  $(\bar{r}_k \vee p_{k-1}) \wedge \bigwedge_{0 \leq d < l} (\bar{r}_k \vee x_{k+d}) \wedge (\bar{r}_k \vee q_{k+l})$  for  $1 \leq k \leq n+1-l$ , also  $(r_1 \vee \dots \vee r_{n+1-l})$ .

b) Show that those clauses are forcing when  $l = 1$  but not when  $l = 2$ .

► **435.** [28] Given  $l \geq 2$ , construct a set of  $O(n \log l)$  clauses that characterize the language  $L_l$  of exercise 434 and are forcing.

**436.** [M32] (*Nondeterministic finite-state automata.*) A regular language  $L$  on the alphabet  $A$  can be defined in the following well-known way: Let  $Q$  be a finite set of “states,” and let  $I \subseteq Q$  and  $O \subseteq Q$  be designated “input states” and “output states.” Also let  $T \subseteq Q \times A \times Q$  be a set of “transition rules.” Then the string  $x_1 \dots x_n$  is in  $L$  if and only if there’s a sequence of states  $q_0, q_1, \dots, q_n$  such that  $q_0 \in I, (q_{k-1}, x_k, q_k) \in T$  for  $1 \leq k \leq n$ , and  $q_n \in O$ .

Given such a definition, where  $A = \{0, 1\}$ , use auxiliary variables to construct clauses that are satisfiable if and only if  $x_1 \dots x_n \in L$ . The clauses should be forcing, and there should be at most  $O(n|T|)$  of them.

As an example, write out the clauses for the language  $L_2 = 0^*1^20^*$  of exercise 434.

**437.** [M21] Extend exercise 436 to the general case where  $A$  has more than two letters.

**438.** [21] Construct a set of forcing clauses that are satisfiable if and only if a given binary string  $x_1 \dots x_n$  contains exactly  $t$  runs of 1s, having lengths  $(l_1, l_2, \dots, l_t)$  from left to right. (Equivalently, the string  $x_1 \dots x_n$  should belong to the language defined by the regular expression  $0^*1^{l_1}0^+1^{l_2}0^+ \dots 0^+1^{l_t}0^*$ .)

► **439.** [30] Find efficient forcing clauses for the constraint that  $x_1 + \dots + x_n = t$  and that there are no two consecutive 1s. (This is the special case  $l_1 = \dots = l_t = 1$  of the previous exercise, but a much simpler construction is possible.)

**440.** [M33] Extend exercise 436 to *context free languages*, which can be defined by a set  $S \subseteq N$  and by production rules  $U$  and  $W$  of the following well-known forms:  $U \subseteq \{P \rightarrow a \mid P \in N, a \in A\}$  and  $W \subseteq \{P \rightarrow QR \mid P, Q, R \in N\}$ , where  $N$  is a set of “nonterminal symbols.” A string  $x_1 \dots x_n$  with each  $x_j \in A$  belongs to the language if and only if it can be produced from a nonterminal symbol  $P \in S$ .

**441.** [M35] Show that any threshold function  $f(x_1, \dots, x_n) = [w_1x_1 + \dots + w_nx_n \geq t]$  has a forcing representation whose size is polynomial in  $\log |w_1| + \dots + \log |w_n|$ .

► **442.** [M27] The unit propagation relation  $\vdash_1$  can be generalized to  $k$ th order propagation  $\vdash_k$  as follows: Let  $F$  be a family of clauses and let  $l$  be a literal. If  $(l_1, l_2, \dots, l_p)$  is a sequence of literals, we write  $L_q^- = \{l_1, \dots, l_{q-1}, \bar{l}_q\}$  for  $1 \leq q \leq p$ . Then

$$\begin{aligned} F \vdash_0 l &\iff \epsilon \in F; \\ F \vdash_{k+1} l &\iff F | L_1^- \vdash_k \epsilon, F | L_2^- \vdash_k \epsilon, \dots, \text{ and } F | L_p^- \vdash_k \epsilon \\ &\quad \text{for some strictly distinct literals } l_1, l_2, \dots, l_p \text{ with } l_p = l; \\ F \vdash_k \epsilon &\iff F \vdash_k l \text{ and } F \vdash_k \bar{l} \text{ for some literal } l. \end{aligned}$$

- Verify that  $\vdash_1$  corresponds to unit propagation according to this definition.
- Describe  $\vdash_2$  informally, using the concept of “failed literals.”
- Prove that  $F \vdash_k \epsilon$  or  $F \vdash_k \bar{l}$  implies  $F | l \vdash_k \epsilon$  for all literals  $l$ , and furthermore that  $F \vdash_k \epsilon$  implies  $F \vdash_{k+1} \epsilon$ , for all  $k \geq 0$ .
- True or false:  $F \vdash_k l$  implies  $F \vdash_{k+1} l$ .
- Let  $L_k(F) = \{l \mid F \vdash_k l\}$ . What is  $L_k(R')$ , where  $R'$  appears in (7) and  $k \geq 0$ ?
- Given  $k \geq 1$ , explain how to compute  $L_k(F)$  and  $F | L_k(F)$  in  $O(n^{2k-1}m)$  steps, when  $F$  has  $m$  clauses in  $n$  variables.

forcing  
 Nondeterministic finite-state automata  
 finite-state automata  
 regular language  
 input states  
 output states  
 transition rules  
 forcing  
 runs of 1s  
 regular expression  
 consecutive 1s  
 context free languages  
 production rules  
 nonterminal symbols  
 threshold function  
 Pseudo-Boolean constraints, see threshold f  
 unit propagation  
 propagation,  $k$ th order  
 failed literals

**443.** [M24] (*A hierarchy of hardness.*) Continuing the previous exercise, a family of clauses  $F$  is said to belong to class  $UC_k$  if it has the property that

$$F|L \vdash \epsilon \text{ implies } F|L \vdash_k \epsilon \quad \text{for all sets of strictly distinct literals } L.$$

(“Whenever a partial assignment yields unsatisfiable clauses, the inconsistency can be detected by  $k$ th order propagation.”) And  $F$  is said to belong to class  $PC_k$  if

$$F|L \vdash l \text{ implies } F|L \vdash_k l \quad \text{for all sets of strictly distinct literals } L \cup l.$$

- Prove that  $PC_0 \subset UC_0 \subset PC_1 \subset UC_1 \subset PC_2 \subset UC_2 \subset \dots$ , where the set inclusions are strict (each class is contained in but unequal to its successor).
- Describe all families  $F$  that belong to the smallest class,  $PC_0$ .
- Give interesting examples of families in the next smallest class,  $UC_0$ .
- True or false: If  $F$  contains  $n$  variables,  $F \in PC_n$ .
- True or false: If  $F$  contains  $n$  variables,  $F \in UC_{n-1}$ .
- Where do the clauses  $R'$  of (7) fall in the hierarchy?

**444.** [M26] The following *single lookahead unit resolution* algorithm, called SLUR, returns either ‘sat’, ‘unsat’, or ‘maybe’, depending on whether a given set  $F$  of clauses is satisfiable, unsatisfiable, or beyond its ability to decide via easy propagations:

**E1.** [Propagate.] If  $F \vdash_1 \epsilon$ , terminate (‘unsat’). Otherwise set  $F \leftarrow F|\{l \mid F \vdash_1 l\}$ .

**E2.** [Satisfied?] If  $F = \emptyset$ , terminate (‘sat’). Otherwise set  $l$  to any literal within  $F$ .

**E3.** [Lookahead and propagate.] If  $F|l \not\vdash_1 \epsilon$ , set  $F \leftarrow F|l|\{l' \mid F|l \vdash_1 l'\}$  and return to E2. Otherwise if  $F|\bar{l} \not\vdash_1 \epsilon$ ,  $F \leftarrow F|\bar{l}|\{l' \mid F|\bar{l} \vdash_1 l'\}$  and return to E2. Otherwise terminate (‘maybe’). ■

Notice that this algorithm doesn’t backtrack after committing itself in E2 to either  $l$  or  $\bar{l}$ .

- If  $F$  consists of Horn clauses, possibly renamed (see exercise 7.1.1–55), prove that SLUR will never return ‘maybe’, regardless of how it chooses  $l$  in step E2.
  - Find four clauses  $F$  on three variables such that SLUR always returns ‘sat’, although  $F$  is *not* a set of possibly renamed Horn clauses.
  - Prove that SLUR never returns ‘maybe’ if and only if  $F \in UC_1$  (see exercise 443).
  - Explain how to implement SLUR in linear time with respect to total clause length.
- **445.** [22] Find short certificates of unsatisfiability for the pigeonhole clauses (106)–(107), when they are supplemented by (a) (181); (b) (182); (c) (183).
- 446.** [M10] What’s the maximum number of edges in a subgraph of  $K_{m,n}$  that has girth  $\geq 6$ ? (Express your answer in terms of  $Z(m, n)$ .)
- **447.** [22] Determine the maximum number of edges in a girth-8 subgraph of  $K_{8,8}$ .
- 448.** [M25] What is  $Z(m, n)$  when  $m$  is odd and  $n = m(m-1)/6$ ? *Hint:* See 6.5–(16).
- 449.** [21] Exhibit  $n \times n$  quad-free matrices that contain the maximum number of 1s and obey the lexicographic constraints (185), (186), for  $8 \leq n \leq 16$ .
- 450.** [25] Prove that there is essentially only one  $10 \times 10$  quad-free system of points and lines with 34 incidences. *Hint:* First show that every line must contain either 3 points or 4 points; hence every point must belong to either 3 lines or 4 lines.
- **451.** [28] Find a way to color the squares of a  $10 \times 10$  board with three colors, so that no rectangle has four corners of the same color. Prove furthermore that every such “nonchromatic rectangle” board has the color distribution  $\{34, 34, 32\}$ , not  $\{34, 33, 33\}$ . But show that if any square of the board is removed, a nonchromatic rectangle is possible with 33 squares of each color.

hierarchy of hardness  
 $UC_k$   
 partial assignment  
 propagation completeness, see  $UC_1$   
 $PC_k$   
 single lookahead unit resolution  
 SLUR  
 lookahead  
 backtrack  
 Horn clauses  
 renamed  
 certificates of unsatisfiability  
 pigeonhole  
 $K_{m,n}$   
 girth  
 $Z(m, n)$   
 nonchromatic rectangle

**452.** [34] Find a nonchromatic rectangle with *four* colors on an  $18 \times 18$  board.

**453.** [M23] An  $m \times n$  matrix  $X = (x_{ij})$  is said to be *decomposable* if it has row indices  $R \subseteq \{1, \dots, m\}$  and column indices  $C \subseteq \{1, \dots, n\}$  such that  $0 < |R| + |C| < m + n$ , with  $x_{ij} = 0$  whenever  $(i \in R \text{ and } j \notin C)$  or  $(i \notin R \text{ and } j \in C)$ . It represents a bipartite graph on the vertices  $\{u_1, \dots, u_m\}$  and  $\{v_1, \dots, v_n\}$ , if  $[u_i - v_j] = [x_{ij} \neq 0]$ .

- Prove that  $X$  is indecomposable if and only if its bipartite graph is connected.
- The *direct sum*  $X' \oplus X''$  of matrices  $X'$  and  $X''$ , where  $X'$  is  $m' \times n'$  and  $X''$  is  $m'' \times n''$ , is the  $(m' + m'') \times (n' + n'')$  “block diagonal” matrix  $X$  that has  $X'$  in its upper left corner,  $X''$  in the lower right corner, and zeros elsewhere (see 7-(40)). True or false: If the rows and columns of  $X'$  and  $X''$  are nonnegative and lexicographically ordered as in (185) and (186), so are the rows and columns of  $X$ .
- Let  $X$  be any nonnegative matrix whose rows and columns are lexicographically nonincreasing, as in (185) and (186). True or false:  $X$  is decomposable if and only if  $X$  is a direct sum of smaller matrices  $X'$  and  $X''$ .

**454.** [15] If  $\tau$  is an endomorphism for the solutions of  $f$ , show that  $f(x) = f(x\tau)$  for every cyclic element  $x$  (every element that's in a cycle of  $\tau$ ).

**455.** [M20] Suppose we know that (187) is an endomorphism of some given clauses  $F$  on the variables  $\{x_1, x_2, x_3, x_4\}$ . Can we be sure that  $F$  is satisfiable if and only if  $F \wedge C$  is satisfiable, when (a)  $C = \bar{1}2\bar{4}$ , i.e.,  $C = (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$ ? (b)  $C = 2\bar{3}\bar{4}$ ? (c)  $C = 123$ ? (d)  $C = 1\bar{3}4$ ?

**456.** [M21] For how many functions  $f(x_1, x_2, x_3, x_4)$  is (187) an endomorphism?

**457.** [HM19] Show that every Boolean  $f(x_1, x_2, x_3, x_4)$  has more than 51 quadrillion endomorphisms, and an  $n$ -variable function has more than  $2^{2^n(n-1)}$ .

**458.** [20] The simplification of clauses by removing an autarky can be regarded as the exploitation of an endomorphism. Explain why.

- **459.** [20] Let  $X_{ij}$  denote the submatrix of  $X$  consisting of the first  $i$  rows and the first  $j$  columns. Show that the numbers  $\text{sweep}(X_{ij})$  satisfy a simple recurrence, from which it's easy to compute  $\text{sweep}(X) = \text{sweep}(X_{mn})$ .

**460.** [21] Given  $m, n, k$ , and  $r$ , construct clauses that are satisfied by an  $m \times n$  binary matrix  $X = (x_{ij})$  if and only if  $\text{sweep}(X) \leq k$  and  $\sum_{i,j} x_{ij} \geq r$ .

**461.** [20] What additional clauses will rule out non-fixed points of  $\tau_1$  and  $\tau_2$ ?

**462.** [M22] Explain why  $\tau_1, \tau_2$ , and  $\tau_3$  preserve satisfiability in the sweep problem.

- **463.** [M21] Show that  $X$  is a fixed point of  $\tau_1, \tau_2$ , and  $\tau_3$  if and only if its rows and columns are nondecreasing. Therefore the maximum of  $\nu X = \sum_{i,j} x_{ij}$  over all binary matrices of sweep  $k$  is a simple function of  $m, n$ , and  $k$ .

- **464.** [M25] Transformations  $\tau_1$  and  $\tau_2$  don't change the text's example  $10 \times 10$  matrix. Prove that they will never change *any*  $10 \times 10$  matrix of sweep 3 that has  $\nu X = 51$ .

**465.** [M21] Justify the text's rule for simultaneous endomorphisms in the perfect matching problem: Any perfect matching must lead to one that's fixed by every  $\tau_{uv}$ .

**466.** [M23] Prove that when  $mn$  is even, the text's even-odd rule (190) for endomorphisms of  $m \times n$  domino coverings has exactly one fixed point.

**467.** [20] Mutilate the  $7 \times 8$  and  $8 \times 7$  boards by removing the upper right and lower left cells. What domino coverings are fixed by all the even-odd endomorphisms like (190)?

decomposable  
bipartite graph  
indecomposable  
connected  
direct sum  
notation  $A \oplus B$   
block diagonal  
lexicographically  
endomorphism  
autarky  
submatrix  
sweep  
recurrence  
perfect matching  
domino coverings  
fixed point  
Mutilate  
even-odd endomorphisms

- 468.** [20] Experiment with the mutilated chessboard problem when the even-odd endomorphisms are modified so that (a) they use the *same* rule for all  $i$  and  $j$ ; or (b) they each make an independent random choice between horizontal and vertical.
- **469.** [M25] Find a certificate of unsatisfiability  $(C_1, C_2, \dots, C_t)$  for the fact that an  $8 \times 8$  chessboard minus cells  $(1, 8)$  and  $(8, 1)$  cannot be exactly covered by dominoes  $h_{ij}$  and  $v_{ij}$  that are fixed under all of the even-odd endomorphisms. Each  $C_k$  for  $1 \leq k < t$  should be a single positive literal. (Therefore the clauses for this problem belong to the relatively simple class  $PC_2$  in the hierarchy of exercise 443.)
- **470.** [M22] Another class of endomorphisms, one for *every* 4-cycle, can also be used in perfect matching problems: Let the *vertices* (instead of the edges) be totally ordered in some fashion. Every 4-cycle can be written  $v_0 - v_1 - v_2 - v_3 - v_0$ , with  $v_0 > v_1 > v_3$  and  $v_0 > v_2$ ; the corresponding endomorphism changes any solution for which  $v_0 v_1 = v_2 v_3 = 1$  by setting  $v_0 v_1 \leftarrow v_2 v_3 \leftarrow 0$  and  $v_1 v_2 \leftarrow v_3 v_0 \leftarrow 1$ . Prove that every perfect matching leads to a fixed point of all these transformations.
- 471.** [16] Find all fixed points of the mappings in exercise 470 when the graph is  $K_{2n}$ .
- 472.** [M25] Prove that even-odd endomorphisms such as (190) in the domino covering problem can be regarded as instances of the endomorphisms in exercise 470.
- **473.** [M23] Generalize exercise 470 to endomorphisms for the unsatisfiable clauses of Tseytin's graph parity problems in exercise 245.
- 474.** [M20] A signed permutation is a symmetry of  $f(x)$  if and only if  $f(x) = f(x\sigma)$  for all  $x$ , and it is an *antisymmetry* if and only if we have  $f(x) = \bar{f}(x\sigma)$  for all  $x$ .
- How many signed permutations of  $n$  elements are possible?
  - Write  $75\bar{1}\bar{4}\bar{2}6\bar{3}$  in cycle form, as an unsigned permutation of  $\{1, \dots, 7, \bar{1}, \dots, \bar{7}\}$ .
  - For how many functions  $f$  of four variables is  $\bar{4}13\bar{2}$  a symmetry?
  - For how many functions  $f$  of four variables is  $\bar{4}13\bar{2}$  an antisymmetry?
  - For how many  $f(x_1, \dots, x_7)$  is  $75\bar{1}\bar{4}\bar{2}6\bar{3}$  a symmetry or antisymmetry?
- 475.** [M22] Continuing exercise 474, a Boolean function is called *asymmetric* if the identity is its only symmetry; it is *totally asymmetric* if it is asymmetric and has no antisymmetries.
- If  $f$  is totally asymmetric, how many functions are equivalent to  $f$  under the operations of permuting variables, complementing variables, and/or complementing the function?
  - According to (a) and 7.1.1-(95), the function  $(x \vee y) \wedge (x \oplus z)$  is not totally asymmetric. What is its nontrivial symmetry?
  - Prove that if  $f$  is not asymmetric, it has an automorphism of prime order  $p$ .
  - Show that if  $(uvw)(\bar{u}\bar{v}\bar{w})$  is a symmetry of  $f$ , so is  $(uv)(\bar{u}\bar{v})$ .
  - Make a similar statement if  $f$  has a symmetry of the form  $(uvwxy)(\bar{u}\bar{v}\bar{w}\bar{x}\bar{y})$ .
  - Conclude that, if  $n \leq 5$ , the Boolean function  $f(x_1, \dots, x_n)$  is totally asymmetric if and only if no signed involution is a symmetry or antisymmetry of  $f$ .
  - However, exhibit a counterexample to that statement when  $n = 6$ .
- 476.** [M23] For  $n \leq 5$ , find Boolean functions of  $n$  variables that are (a) asymmetric but not totally asymmetric; (b) totally asymmetric. Furthermore, your functions should be the easiest to evaluate (in the sense of having a smallest possible Boolean chain), among all functions that qualify. *Hint:* Combine exercises 475 and 477.
- **477.** [23] (*Optimum Boolean evaluation.*) Construct clauses that are satisfiable if and only if there is an  $r$ -step normal Boolean chain that computes  $m$  given functions  $g_1$ ,

certificate of unsatisfiability  
 $PC_2$   
hierarchy  
4-cycle  
complete graph  
Tseytin  
graph-based axioms  
parity-related clauses  
signed permutation  
antisymmetry  
cycle form  
asymmetric  
equivalence of Boolean functions  
evaluation of Boolean functions  
Boolean chain

$\dots, g_m$  on  $n$  variables. (For example, if  $n = 3$  and  $g_1 = \langle x_1x_2x_3 \rangle$ ,  $g_2 = x_1 \oplus x_2 \oplus x_3$ , such clauses with  $r = 4$  and  $5$  enable a SAT solver to discover a “full adder” of minimum cost; see 7.1.2-(1) and 7.1.2-(22).) *Hint*: Represent each bit of the truth tables.

- ▶ **478.** [23] Suggest ways to break symmetry in the clauses of exercise 477.
- ▶ **479.** [25] Use SAT technology to find optimum circuits for the following problems:
  - a) Compute  $z_2, z_1$ , and  $z_0$ , when  $x_1 + x_2 + x_3 + x_4 = (z_2z_1z_0)_2$  (see 7.1.2-(27)).
  - b) Compute  $z_2, z_1$ , and  $z_0$ , when  $x_1 + x_2 + x_3 + x_4 + x_5 = (z_2z_1z_0)_2$ .
  - c) Compute all four symmetric functions  $S_0, S_1, S_2, S_3$  of  $\{x_1, x_2, x_3\}$ .
  - d) Compute all five symmetric functions  $S_0, S_1, S_2, S_3, S_4$  of  $\{x_1, x_2, x_3, x_4\}$ .
  - e) Compute the symmetric function  $S_3(x_1, x_2, x_3, x_4, x_5, x_6)$ .
  - f) Compute the symmetric function  $S_{0,4}(x_1, \dots, x_6) = [(x_1 + \dots + x_6) \bmod 4 = 0]$ .
  - g) Compute all eight minterms of  $\{x_1, x_2, x_3\}$  (see 7.1.2-(30)).
- 480.** [25] Suppose the values 0, 1, 2 are encoded by the two-bit codes  $x_1x_r = 00, 01$ , and  $1*$ , respectively, where 10 and 11 both represent 2. (See Eq. 7.1.3-(120).)
  - a) Find an optimum circuit for mod 3 addition:  $z_iz_r = (x_1x_r + y_1y_r) \bmod 3$ .
  - b) Find an optimum circuit that computes  $z_iz_r = (x_1 + x_2 + x_3 + y_1y_r) \bmod 3$ .
  - c) Conclude that  $[x_1 + \dots + x_n \equiv a \pmod{3}]$  can be computed in  $< 3n$  steps.
- ▶ **481.** [28] An ordered bit pair  $xy$  can be encoded by another ordered bit pair  $[xy] = (x \oplus y)y$  without loss of information, because  $[xy] = uv$  implies  $[uv] = xy$ .
  - a) Find an optimum circuit that computes  $([zz'])_2 = x_1 + x_2 + x_3$ .
  - b) Let  $\nu[uv] = (u \oplus v) + v$ , and note that  $\nu[00] = 0, \nu[01] = 2, \nu[1*] = 1$ . Find an optimum circuit that, given  $x_1 \dots x_5$ , computes  $z_1z_2z_3$  such that we have  $\nu[x_1x_2] + \nu[x_3x_4] + x_5 = 2\nu[z_1z_2] + z_3$ .
  - c) Use that circuit to prove by induction that the “sideways sum”  $(z_{[1g_n]} \dots z_1z_0)_2 = x_1 + x_2 + \dots + x_n$  can always be computed with fewer than  $4.5n$  gates.
- ▶ **482.** [26] (*Erdős discrepancy patterns.*) The binary sequence  $y_1 \dots y_t$  is called *strongly balanced* if we have  $|\sum_{j=1}^k (2y_j - 1)| \leq 2$  for  $1 \leq k \leq t$ .
  - a) Show that this balance condition needs to be checked only for odd  $k \geq 3$ .
  - b) Describe clauses that efficiently characterize a strongly balanced sequence.
  - c) Construct clauses that are satisfied by  $x_1x_2 \dots x_n$  if and only if  $x_dx_{2d} \dots x_{\lfloor n/d \rfloor d}$  is strongly balanced for  $1 \leq d \leq n$ .

**483.** [21] Symmetry between colors was broken in the coloring problems of Table 6 by assigning fixed colors to a large clique in each graph. But many graphs have no large clique, so a different strategy is necessary. Explain how to encode the “restricted growth string” principle (see Section 7.2.1.5) with appropriate clauses, given an ordering  $v_1v_2 \dots v_n$  of the vertices: The color of  $v_j$  must be at most one greater than the largest color assigned to  $\{v_1, \dots, v_{j-1}\}$ . (In particular,  $v_1$  always has color 1.)

Experiment with this scheme by applying it to the *book* graphs *anna*, *david*, *homer*, *huck*, and *jean* of the Stanford GraphBase.

**484.** [22] (*Graph quenching.*) A graph with vertices  $(v_1, \dots, v_n)$  is called “quenchable” if either (i)  $n = 1$ ; or (ii) there's a  $k$  such that  $v_k \text{ --- } v_{k+1}$  and the graph on  $(v_1, \dots, v_{k-1}, v_{k+1}, \dots, v_n)$  can be quenched; or (iii) there's an  $l$  such that  $v_l \text{ --- } v_{l+3}$  and the graph on  $(v_1, \dots, v_{l-1}, v_{l+3}, v_{l+1}, v_{l+2}, v_{l+4}, \dots, v_n)$  can be quenched.

- a) Find a 4-element graph that is quenchable although  $v_3 \not\text{---} v_4$ .
- b) Construct clauses that are satisfiable if and only if a given graph is quenchable.

*Hint*: Use the following three kinds of variables for this model-checking problem:

full adder  
truth tables  
break symmetry  
symmetric functions  
Mod 4 parity  
minterms  
binary decoder  
encoding of ternary data  
representing three states with two bits  
mapping three items into two-bit codes  
Mod 3 parity  
sideways sum  
Erdős discrepancy patterns  
strongly balanced  
Symmetry between colors  
coloring problems  
clique  
encode  
restricted growth string  
*book* graphs  
Stanford GraphBase  
Graph quenching  
quenchable  
model-checking problem



$x_{t,i,j} = [v_i - v_j \text{ at time } t]$ , for  $1 \leq i < j \leq n-t$ ;  $q_{t,k} = [\text{a quenching move of type (ii) leads to time } t+1]$ ;  $s_{t,l} = [\text{a quenching move of type (iii) leads to time } t+1]$ .

- **485.** [23] Sometimes successive transitions in the previous exercise are commutative: For example, the effect of  $q_{t,k}$  and  $q_{t+1,k+1}$  is the same as  $q_{t,k+2}$  and  $q_{t+1,k}$ . Explain how to break symmetry in such cases, by allowing only one of the two possibilities.

**486.** [21] (*Late Binding solitaire.*) Shuffle a deck and deal out 18 cards; then try to reduce these 18 piles to a single pile, using a sequence of “captures” in which one pile is placed on top of another pile. A pile can capture only the pile to its immediate left, or the pile found by skipping left over two other piles. Furthermore a capture is permitted only if the top card in the capturing pile has the same suit or the same rank as the top card in the captured pile. For example, consider the following deal:

J♥ 5♥ 10♣ 8♦ J♣ A♣ K♠ A♥ 4♣ 8♠ 5♠ 5♦ 2♦ 10♠ A♠ 6♥ 3♥ 10♦

Ten captures are initially possible, including  $5♥ \times J♥$ ,  $A♣ \times 10♣$ , and  $5♦ \times 5♠$ . Some captures then make others possible, as in  $8♠ \times K♠$  and  $8♦$ .

If captures must be made “greedily” from left to right as soon as possible, this game is the same as the first 18 steps of a classic one-player game called “Idle Year,” and we wind up with five piles [see *Dick’s Games of Patience* (1883), 50–52]. But if we cleverly hold back until all 18 cards have been dealt, we can do much better.

Show that one can win from this position, but not if the first move is  $A♣ \times J♣$ .

- **487.** [27] There are  $\binom{64}{8} = 4426165368$  ways to place eight queens on a chessboard. Long ago, W. H. Turton asked which of them causes the maximum number of vacant squares to remain unattacked. [See W. W. Rouse Ball, *Mathematical Recreations and Problems*, third edition (London: Macmillan, 1896), 109–110.]

Every subset  $S$  of the vertices of a graph has three *boundary sets* defined thus:

- $\partial S =$  the set of all edges with exactly one endpoint  $\in S$ ;
- $\partial_{\text{out}} S =$  the set of all vertices  $\notin S$  with at least one neighbor  $\in S$ ;
- $\partial_{\text{in}} S =$  the set of all vertices  $\in S$  with at least one neighbor  $\notin S$ .

Find the minimum and maximum sizes of  $\partial S$ ,  $\partial_{\text{out}} S$ , and  $\partial_{\text{in}} S$ , over all 8-element sets  $S$  in the queen graph  $Q_8$  (exercise 7.1.4–241). Which set answers Turton’s question?

- **488.** [24] (*Peaceable armies of queens.*) Prove that armies of nine white queens and nine black queens can coexist on a chessboard without attacking each other, but armies of size 10 cannot, by devising appropriate sets of clauses and applying Algorithm C. Also examine the effects of symmetry breaking. (This problem has sixteen symmetries, because we can swap colors and/or rotate and/or reflect the board.) How large can coexisting armies of queens be on  $n \times n$  boards, for  $n \leq 11$ ?

**489.** [M21] Find a recurrence for  $T_n$ , the number of signed involutions on  $n$  elements.

- **490.** [15] Does Theorem E hold also when  $p_1 p_2 \dots p_n$  is any *signed* permutation?
- **491.** [22] The unsatisfiable clauses  $R$  in (6) have the signed permutation  $234\bar{1}$  as an automorphism. How can this fact help us to verify their unsatisfiability?

**492.** [M20] Let  $\tau$  be a *signed mapping* of the variables  $\{x_1, \dots, x_n\}$ ; for example, the signed mapping ‘ $\bar{4}13\bar{3}$ ’ stands for the operation  $(x_1, x_2, x_3, x_4) \mapsto (x_{\bar{4}}, x_1, x_3, x_{\bar{3}}) = (\bar{x}_4, x_1, x_3, \bar{x}_3)$ . When a signed mapping is applied to a clause, some of the resulting literals might coincide; or two literals might become complementary, making a tautology. When  $\tau = \bar{4}13\bar{3}$ , for instance, we have  $(123)\tau = \bar{4}13$ ,  $(13\bar{4})\tau = \bar{4}3$ ,  $(1\bar{3}\bar{4})\tau = \emptyset$ .

commutative  
break symmetry  
Late Binding solitaire  
solitaire  
patience  
playing cards  
Idle Year  
Dick  
queens  
chessboard  
Turton  
Ball  
boundary sets  
queen graph  
armies of queens  
queens  
symmetry breaking  
coexisting armies of queens  
signed involutions  
signed  
automorphism  
signed mapping  
tautology

A family  $F$  of clauses is said to be “closed” under a signed mapping  $\tau$  if  $C\tau$  is subsumed by some clause of  $F$  whenever  $C \in F$ . Prove that  $\tau$  is an endomorphism of  $F$  in such a case.

**493.** [20] The problem *waerden*(3, 3; 9) has four symmetries, because we can reflect and/or complement all the variables. How can we speed up the proof of unsatisfiability by adding clauses to break those symmetries?

**494.** [21] Show that if  $(uvw)(\bar{u}\bar{v}\bar{w})$  is a symmetry of some clauses  $F$ , we’re allowed to break symmetries as if  $(uv)(\bar{u}\bar{v})$ ,  $(uw)(\bar{u}\bar{w})$ , and  $(vw)(\bar{v}\bar{w})$  were also symmetries. For example, if  $i < j < k$  and if  $(ijk)(\bar{i}\bar{j}\bar{k})$  is a symmetry, we can assert  $(\bar{x}_i \vee x_j) \wedge (\bar{x}_j \vee x_k)$  with respect to the global ordering  $p_1 \dots p_n = 1 \dots n$ . What are the corresponding binary clauses when the symmetry is (i)  $(ijk)(\bar{i}\bar{j}\bar{k})$ ? (ii)  $(i\bar{j}k)(\bar{i}\bar{j}\bar{k})$ ? (iii)  $(i\bar{j}\bar{k})(\bar{i}\bar{j}\bar{k})$ ?

**495.** [M22] Spell out the details of how we can justify appending clauses to assert (185) and (186), using Corollary E, whenever we have an  $m \times n$  problem whose variables  $x_{ij}$  possess both row and column symmetry. (In other words we assume that  $x_{ij} \mapsto x_{(i\pi)(j\rho)}$  is an automorphism for all permutations  $\pi$  of  $\{1, \dots, m\}$  and  $\rho$  of  $\{1, \dots, n\}$ .)

► **496.** [M20] B. C. Dull reasoned as follows: “The pigeonhole clauses have row and column symmetry. Therefore we can assume that the rows are lexicographically increasing from top to bottom, and the columns are lexicographically increasing from right to left. Consequently the problem is easily seen to be unsatisfiable.” Was he correct?

**497.** [22] Use BDD methods to determine the number of  $8 \times 8$  binary matrices that have both rows and columns in nondecreasing lexicographic order. How many of them have exactly  $r$  1s, for  $r = 24$ ,  $r = 25$ ,  $r = 64 - 25 = 39$ , and  $r = 64 - 24 = 40$ ?

**498.** [22] Justify adding the symmetry-breakers (183) to the pigeonhole clauses.

**499.** [21] In the pigeonhole problem, is it legitimate to include the clauses (183) together with clauses that enforce lexicographic row and column order?

**500.** [16] The precocious student J. H. Quick decided to extend the monkey wrench principle, arguing that if  $F_0 \cup S \vdash l$  then the original clauses  $F$  can be replaced by  $F|l$ . But he soon realized his mistake. What was it?

**501.** [22] Martin Gardner introduced an interesting queen placement problem in *Scientific American* **235**, 4 (October 1976), 134–137: “Place  $r$  queens on an  $m \times n$  chessboard so that (i) no three are in the same row, column, or diagonal; (ii) no empty square can be occupied without breaking rule (i); and (iii)  $r$  is as small as possible.” Construct clauses that are satisfiable if and only if there’s a solution to conditions (i) and (ii) with at most  $r$  queens. (A similar problem was considered in exercise 7.1.4–242.)

**502.** [16] (*Closest strings*.) Given binary strings  $s_1, \dots, s_m$  of length  $n$ , and threshold parameters  $r_1, \dots, r_m$ , construct clauses that are satisfiable by  $x = x_1 \dots x_n$  if and only if  $x$  differs from  $s_j$  in at most  $r_j$  positions, for  $1 \leq j \leq m$ .

**503.** [M20] (*Covering strings*.) Given  $s_j$  and  $r_j$  as in exercise 502, show that every string of length  $n$  is within  $r_j$  bits of some  $s_j$  if and only if the closest string problem has no solution with parameters  $r'_j = n - 1 - r_j$ .

► **504.** [M21] The problem in exercise 502 can be proved NP-complete as follows:

- Let  $w_j$  be the string of length  $2n$  that is entirely 0 except for 1s in positions  $2j - 1$  and  $2j$ , and let  $w_{n+j} = \bar{w}_j$ , for  $1 \leq j \leq n$ . Describe all binary strings of length  $2n$  that differ from each of  $w_1, \dots, w_{2n}$  in at most  $n$  bit positions.
- Given a clause  $(l_1 \vee l_2 \vee l_3)$  with strictly distinct literals  $l_1, l_2, l_3 \in \{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$ , let  $y$  be the string of length  $2n$  that is entirely zero except that it has

subsumed  
*waerden*  
 Dull  
 pigeonhole  
 BDD  
 lexicographic order  
 Quick  
 monkey wrench principle  
 Gardner  
 queen placement  
 Closest strings  
 binary strings  
 noisy data  
 Covering strings  
 NP-complete

1 in position  $2k - 1$  when some  $l_i$  is  $\bar{x}_k$ , and 1 in position  $2k$  when some  $l_i$  is  $x_k$ .  
In how many bit positions does a string that satisfies (a) differ from  $y$ ?

- c) Given a 3SAT problem  $F$  with  $m$  clauses and  $n$  variables, use (a) and (b) to construct strings  $s_1, \dots, s_{m+2n}$  of length  $2n$  such that  $F$  is satisfiable if and only if the closest string problem is satisfiable with  $r_j = n + [j > 2n]$ .
- d) Illustrate your construction in (c) by exhibiting the closest string problems that correspond to the simple 3SAT problems  $R$  and  $R'$  in (6) and (7).

**505.** [21] Experiment with making Algorithm L nondeterministic, by randomizing the initial order of VAR in step L1 just as HEAP is initialized randomly in step C1. How does the modified algorithm perform on, say, problems D3, K0, and W2 of Table 6?

**506.** [22] The *weighted variable interaction graph* of a family of clauses has one vertex for each variable and the weight  $\sum 2/(|c|(|c| - 1))$  between vertices  $u$  and  $v$ , where the sum is over all clauses  $c$  that contain both  $\pm u$  and  $\pm v$ . Figure 52 indicates these weights indirectly, by making the heavier edges darker.

- a) True or false: The sum of all edge weights is the total number of clauses.  
b) Explain why the graph for test case B2 has exactly 6 edges of weight 2. What are the weights of the other edges in that graph?

► **507.** [21] (Marijn Heule.) Explain why “windfalls” (see (72)) help Algorithm L to deal with miter problems such as D5.

**508.** [M20] According to Table 7, Algorithm C proved problem T3 to be unsatisfiable after learning about 323 thousand clauses. About how many times did it enter a purging phase in step C7?

**509.** [20] Several of the “training set” tasks used when tuning Algorithm C’s parameters were taken from the 100 test cases of Table 6. Why didn’t this lead to a problem of “overfitting” (namely, of choosing parameters that are too closely associated with the trainees)?

**510.** [18] When the data points A1, A2,  $\dots$ , X8 were plotted in Fig. 55, one by one, they sometimes covered parts of previously plotted points, because of overlaps. What test cases are partially hidden by (a) T2? (b) X6? (c) X7?

**511.** [22] Problem P4 in Table 6 is a strange set of clauses that lead to extreme behavior of Algorithm C in Figs. 54 and 55; and it causes Algorithm L to “time out” in Fig. 53.

- a) The preprocessing algorithm of the text needs about 1.5 megamems to convert those 2509 clauses in 400 variables into just 2414 clauses in 339 variables. Show empirically that Algorithm L makes short work of the resulting 2414 clauses.  
b) How efficient is Algorithm C on those preprocessed clauses?  
c) What is the behavior of WalkSAT on P4, with and without preprocessing?

**512.** [29] Find parameters for Algorithm C that will find an Erdős discrepancy pattern  $x_1 x_2 \dots x_n$  rapidly when  $n = 500$ . (This is problem E0 in Table 6.) Then compare the running times of nine random runs with your parameters versus nine random runs with (194), when  $n = 400, 500, 600, \dots, 1100, 1160, \text{ and } 1161$ .

**513.** [24] Find parameters for Algorithm L that tune it for  $rand(3, m, n, seed)$ .

**514.** [24] The timings quoted in the text for Algorithm W, for problems in Table 6, are based on the median of nine runs using the parameters  $p = .4$  and  $N = 50n$ , restarting from scratch if necessary until a solution is found. Those parameters worked fine in most cases, unless Algorithm W was unsuited to the task. But problem C9 was solved more quickly with  $p = .6$  and  $N = 2500n$  (943  $M\mu$  versus 9.1  $G\mu$ ).

Find values of  $p$  and  $N/n$  that give near-optimum performance for problem C9.

3SAT  
closest string  
Rivest’s clauses  
nondeterministic  
randomizing  
variable interaction graph  
Heule  
windfalls  
miter problems  
purging  
training set  
tuning  
overfitting  
preprocessing  
lookahead solver versus conflict driven  
WalkSAT  
Erdős discrepancy pattern  
discrepancy  
*rand*  
WalkSAT

- **515.** [23] (*Hard sudoku.*) Specify SAT clauses with which a designer of sudoku puzzles can meet the following specifications: (i) If cell  $(i, j)$  of the puzzle is blank, so is cell  $(10-i, 10-j)$ , for  $1 \leq i, j \leq 9$ . (ii) Every row, every column, and every box contains at least one blank. (Here “box” means one of sudoku’s nine special  $3 \times 3$  subarrays.) (iii) No box contains an all-blank row or an all-blank column. (iv) There are at least two ways to fill every blank cell, without conflicting with nonblank entries in the same row, column, or box. (v) If a row, column, or box doesn’t already contain  $k$ , there are at least two places to put  $k$  into that row, column, or box, without conflict. (vi) If the solution has a  $2 \times 2$  subarray of the form  $\begin{smallmatrix} k & l \\ l & k \end{smallmatrix}$ , those four cells must not all be blank.

Hard sudoku  
sudoku  
exact cover problem  
strong exponential time hypothesis  
 $k$ SAT  
one-per-clause  
NP-complete  
3SAT  
one-in-three  
ternary  
permanent  
gadget

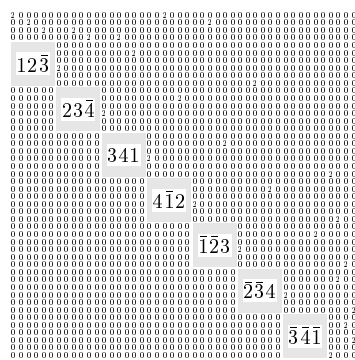
(Condition (i) is a feature of “classic” sudoku puzzles. Conditions (iv) and (v) ensure that the corresponding exact cover problem has no forced moves; see Section 7.2.2.1. Condition (vi) rules out common cases with non-unique solutions.)

- 516.** [M49] Prove or disprove the *strong exponential time hypothesis*: “If  $\tau < 2$ , there is an integer  $k$  such that no randomized algorithm can solve every  $k$ SAT problem in fewer than  $\tau^n$  steps, where  $n$  is the number of variables.”

- 517.** [25] Given clauses  $C_1, \dots, C_m$ , the *one-per-clause* satisfiability problem asks if there is a Boolean assignment  $x_1 \dots x_n$  such that every clause is satisfied by a *unique* literal. In other words, we want to solve the simultaneous equations  $\Sigma C_j = 1$  for  $1 \leq j \leq m$ , where  $\Sigma C$  is the sum of the literals of clause  $C$ .

- a) Prove that this problem is NP-complete, by reducing 3SAT to it.
- b) Prove that this problem, in turn, can be reduced to its special case “*one-in-three* satisfiability,” where every given clause is required to be ternary.

- 518.** [M32] Given a 3SAT problem with  $m$  clauses and  $n$  variables, we shall construct a  $(6m + n) \times (6m + n)$  matrix  $M$  of integers such that the *permanent*, per  $M$ , is zero if and only if the clauses are unsatisfiable. For example, the solvable problem (7) corresponds to the  $46 \times 46$  matrix indicated here; each shaded box stands for a fixed  $6 \times 6$  matrix  $A$  that corresponds to a clause.



Each  $A$  has three “inputs” in columns 1, 3, 5 and three “outputs” in rows 2, 4, 6. The first  $n$  rows and the last  $n$  columns correspond to variables. Outside of the  $A$ s, all entries are either 0 or 2; and the 2s link variables to clauses, according to a scheme much like the data structures in several of the algorithms in this section:

Let  $I_{ij}$  and  $O_{ij}$  denote the  $j$ th input and output of clause  $i$ , for  $1 \leq i \leq m$  and  $1 \leq j \leq 3$ . Then, if literal  $l$  appears in  $t \geq 0$  clauses  $i_1 < \dots < i_t$ , as element  $j_1, \dots, j_t$ , we put ‘2’ in column  $I_{i_k+1j_{k+1}}$  of row  $O_{i_k j_k}$  for  $0 \leq k \leq t$  ( $O_{i_0 j}$  is row  $|l|$ ,  $I_{i_t+1j}$  is column  $6m + |l|$ ).

- a) Find a  $6 \times 6$  matrix  $A = (a_{ij})$ , whose elements are either 0, 1, or  $-1$ , such that

$$\text{per} \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21}+2r & a_{22} & a_{23}+2s & a_{24} & a_{25}+2t & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41}+2u & a_{42} & a_{43}+2v & a_{44} & a_{45}+2w & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61}+2x & a_{62} & a_{63}+2y & a_{64} & a_{65}+2z & a_{66} \end{pmatrix} = 16 \left( \text{per} \begin{pmatrix} r+1 & s & t \\ u & v+1 & w \\ x & y & z+1 \end{pmatrix} - 1 \right).$$

*Hint:* There’s a solution with lots of symmetry.

- b) In which of the rows and columns of  $M$  does ‘2’ occur twice? once? not at all?
- c) Conclude that per  $M = 2^{4m+n} s$ , when the 3SAT problem has exactly  $s$  solutions.

**519.** [20] Table 7 shows inconclusive results in a race for factoring between *factor\_fifo* and *factor\_lifo*. What is the comparable performance of *factor\_rand*( $m, n, z, 314159$ )?

- **520.** [24] Every instance of SAT corresponds in a natural way to an *integer programming feasibility* problem: To find, if possible, integers  $x_1, \dots, x_n$  that satisfy the linear inequalities  $0 \leq x_j \leq 1$  for  $1 \leq j \leq n$  and

$$l_1 + l_2 + \dots + l_k \geq 1 \quad \text{for each clause } C = (l_1 \vee l_2 \vee \dots \vee l_k).$$

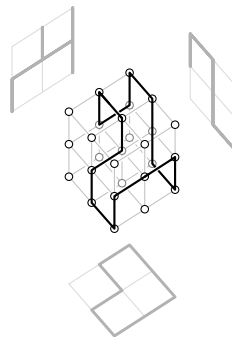
For example, the inequality that corresponds to the clause  $(x_1 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_7)$  is  $x_1 + (1-x_3) + (1-x_4) + x_7 \geq 1$ ; i.e.,  $x_1 - x_3 - x_4 + x_7 \geq -1$ .

Sophisticated “IP solvers” have been developed by numerous researchers for solving general systems of integer linear inequalities, based on techniques of “cutting planes” in high-dimensional geometry. Thus we can solve any satisfiability problem by using such general-purpose software, as an alternative to trying a SAT solver.

Study the performance of the best available IP solvers, with respect to the 100 sets of clauses in Table 6, and compare it to the performance of Algorithm C in Table 7.

**521.** [30] Experiment with the following idea, which is much simpler than the clause-purging method described in the text: “Forget a learned clause of length  $k$  with probability  $p_k$ ,” where  $p_1 \geq p_2 \geq p_3 \geq \dots$  is a tunable sequence of probabilities.

- **522.** [26] (*Loopless shadows.*) A cyclic path within the cube  $P_3 \square P_3 \square P_3$  is shown here, together with the three “shadows” that appear when it is projected onto each coordinate plane. Notice that the shadow at the bottom contains a loop, but the other two shadows do not. Does this cube contain a cycle whose three shadows are entirely *without* loops? Use SAT technology to find out.



**523.** [30] Prove that, for any  $m$  or  $n$ , no cycle of the graph  $P_m \square P_n \square P_2$  has loopless shadows.

- **524.** [22] Find all *Hamiltonian paths* of the cube  $P_3 \square P_3 \square P_3$  that have loopless shadows.
- **525.** [40] Find the most difficult 3SAT problem you can that has at most 100 variables.

**526.** [M25] (David S. Johnson, 1974.) If  $F$  has  $m$  clauses, all of size  $\geq k$ , prove that some assignment leaves at most  $m/2^k$  clauses unsatisfied.

**999.** [M00] this is a temporary exercise (for dummies)

*factor\_fifo*  
*factor\_lifo*  
*factor\_rand*  
 integer programming  
 linear inequalities  
 IP solvers  
 cutting planes  
 100 sets of clauses  
 purging  
 Loopless shadows  
 shadows  
 projected  
 Hamiltonian paths  
 difficult  
 3SAT  
 Johnson  
 MAXSAT lower bound

## SECTION 7.2.2.2

1. (a)  $\emptyset$  (no clauses). (b)  $\{\epsilon\}$  (one clause, which is empty).

2. Letting 1  $\leftrightarrow$  lazy, 2  $\leftrightarrow$  happy, 3  $\leftrightarrow$  unhealthy, 4  $\leftrightarrow$  dancer, we're given the respective clauses  $\{314, \bar{1}42, 3\bar{4}2, \bar{2}4\bar{3}, \bar{1}3\bar{2}, \bar{2}\bar{3}1, \bar{1}\bar{4}\bar{3}\}$ , matching  $R'$  in (7). So all known Pincusians dance happily, and none are lazy. But we know nothing about their health. [And we might wonder why travelers have bothered to describe so many empty sets.]

3.  $f(j-1, n) + f(k-1, n)$ , where  $f(p, n) = \sum_{d=1}^q (n - pd) = p\binom{q}{2} + q(n \bmod p) \approx n^2/(2p)$ , if we set  $q = \lfloor n/p \rfloor$ .

4. Those constraints are unsatisfiable if and only if we remove a subset of either  $\{357, 456, \bar{3}\bar{5}\bar{7}, \bar{4}\bar{5}\bar{6}\}$ ,  $\{246, 468, \bar{2}\bar{4}\bar{6}, \bar{4}\bar{6}\bar{8}\}$ ,  $\{246, 357, 468, \bar{4}\bar{5}\bar{6}\}$ , or  $\{456, \bar{2}\bar{4}\bar{6}, \bar{3}\bar{5}\bar{7}, \bar{4}\bar{6}\bar{8}\}$ .

5. No polynomial upper bound for  $W(3, k)$  is currently known. Clearly  $W(3, k)$  is less than  $\bar{W}(3, k)$ , the minimum  $n$  that guarantees either three equally spaced 0s or  $k$  consecutive 1s. An analysis by R. L. Graham in *Integers* **6** (2006), A29:1–A29:5, beefed up by a subsequent theorem of T. H. Bloom in *arXiv:1405.5800 [math.NT]* (2014), 22 pages, shows that  $\bar{W}(3, k) = \exp O(k(\log k)^4)$ .

6. Let each  $x_i$  be 0 with probability  $p = (2 \ln k)/k$ , and let  $n$  be at most  $k^2/(\ln k)^3$ . There are two kinds of “bad events”:  $A_i$ , a set of three equally spaced 0s, occurs with probability  $P = p^3$ ; and  $A'_j$ , a set of  $k$  equally spaced 1s, occurs with probability  $P' = (1-p)^k \leq \exp(-kp) = 1/k^2$ . In the lopsided dependency graph, which is bipartite, each  $A_i$  is adjacent to at most  $D = 3k^3/((k-1)(\ln k)^3)$  nodes  $A'_j$ ; each  $A'_j$  is adjacent to at most  $d = \frac{3}{2}k^3/(\ln k)^3$  nodes  $A_i$ . By Theorem J, we want to show that, for all sufficiently large values of  $k$ ,  $P \leq y(1-x)^D$  and  $P' \leq x(1-y)^d$ , for some  $x$  and  $y$ .

Choose  $x$  and  $y$  so that  $(1-x)^D = 1/2$  and  $y = 2P$ . Then  $x = \Theta((\log k)^3/k^2)$  and  $y = \Theta((\log k)^3/k^3)$ ; hence  $(1-y)^d = \exp(-yd + O(y^2d)) = O(1)$ . [See T. Brown, B. M. Landman, and A. Robertson, *J. Combinatorial Theory* **A115** (2008), 1304–1309.]

7. Yes, for all  $n$ , when  $x_1x_2x_3\dots = 001001001\dots$

8. For example, let  $x_{i,a}$  signify that  $x_i = a$ , for  $1 \leq i \leq n$  and  $0 \leq a < b$ . The relevant clauses are then  $x_{i,0} \vee \dots \vee x_{i,b-1}$  for  $1 \leq i \leq n$ ; and  $\bar{x}_{i,a} \vee \bar{x}_{i+d,a} \vee \dots \vee \bar{x}_{i+(k_a-1)d,a}$ , for  $1 \leq i \leq n - (k_a - 1)d$  and  $d \geq 1$ . Optionally include the clauses  $\bar{x}_{i,a} \vee \bar{x}_{i,a'}$  for  $0 \leq a < a' < b$ . (Whenever the relevant clauses are satisfiable, we can also satisfy the optional ones by falsifying some variables if necessary.)

[V. Chvátal found  $W(3, 3, 3) = 27$ . Kouril's paper shows that  $W(2, 4, 8) = 157$ ,  $W(2, 3, 14) = 202$ ,  $W(2, 5, 6) = 246$ ,  $W(4, 4, 4) = 293$ , and lists many smaller values.]

9.  $W(2, 2, k) = 3k - (2, 0, 2, 2, 1, 0)$  when  $k \bmod 6 = (0, 1, 2, 3, 4, 5)$ . The sequence  $2^{k-1}02^{k-1}12^{k-1}$  is maximal when  $k \perp 6$ ; also  $2^{k-1}02^{k-1}12^{k-3}$  when  $k \bmod 6 = 3$ ; also  $2^{k-1}02^{k-2}12^{k-1}$  when  $k \bmod 6 = 4$ ; otherwise  $2^{k-1}02^{k-2}12^{k-2}$ . [See B. Landman, A. Robertson, and C. Culver, *Integers* **5** (2005), A10:1–A10:11, where many other values of  $W(2, \dots, 2, k)$  are also established.]

10. If the original variables are  $\{1, \dots, n\}$ , let the new ones be  $\{1, \dots, n\} \cup \{1', \dots, n'\}$ . The new problem has positive clauses  $\{11', \dots, nn'\}$ . Its negative clauses are, for example,  $2'\bar{6}\bar{7}\bar{9}'$  if  $2\bar{6}\bar{7}\bar{9}$  was an original clause. The original problem is equivalent because it can be obtained from the new one by resolving away the primed variables.

[One can in fact construct an equivalent monotonic problem of size  $O(m+n)$  in which  $(x_1 \vee \dots \vee x_k)$  is a positive clause if and only if  $(\bar{x}_1 \vee \dots \vee \bar{x}_k)$  is a negative clause. Such a problem, “not-all-equal SAT,” is equivalent to 2-colorability of hypergraphs. See L. Lovász, *Congressus Numerantium* **8** (1973), 3–12; H. Kleine Büning and T. Lettmann, *Propositional Logic* (Cambridge Univ. Press, 1999), §3.2, Problems 4–8.]

$\emptyset$   
empty clause  
nullary clause  
empty sets  
unsatisfiable core  
Graham  
Bloom  
lopsided dependency graph  
Brown  
Landman  
Robertson  
Chvátal  
Kouril  
Landman  
Robertson  
Culver  
resolving  
not-all-equal SAT  
hypergraph 2-colorability  
2-colorability of hypergraphs  
Lovász  
Kleine Büning  
Lettmann

11. For each variable  $i$ , the only way to match vertices of the forms  $ij'$  and  $ij''$  is to choose all of its true triples or all of its false triples.

For each clause  $j$ , the vertex pairs  $\{j'2, j'3\}$ ,  $\{j'4, j'5\}$ ,  $\{j'6, j'7\}$  define three “slots”; hence two of the vertices  $\{wj, xj, yj, zj\}$  must be matched into the same slot. Furthermore we can't have two in one slot and two in another, because the remaining slot would then be unmatched. Thus two of the  $\bar{l}j$  vertices are matched in their slot, while the other is matched with  $j'1$ , whenever we have a perfect matching.

Conversely, if all clauses are satisfied, with  $l_k$  true in clause  $j$ , there always are exactly two ways to match  $\bar{l}_k j$  with  $j'1$  while matching  $wj, xj, yj, zj$ , and the other two  $\bar{l}j$  vertices with  $j'2, \dots, j'7$ . (It's a beautiful construction! Notice that no vertex appears in more than three triples.)

12. Equation (13) says  $S_1(y_1, \dots, y_p) = S_{\geq 1}(y_1, \dots, y_p) \wedge S_{\leq 1}(y_1, \dots, y_p)$ . If  $p \leq 4$ , use  $\bigwedge_{1 \leq j < k \leq p} (\bar{y}_j \vee \bar{y}_k)$  for  $S_{\leq 1}(y_1, \dots, y_p)$ ; otherwise  $S_{\leq 1}(y_1, \dots, y_p)$  can be encoded recursively via the clauses  $S_{\leq 1}(y_1, y_2, y_3, t) \wedge S_{\leq 1}(\bar{t}, y_4, \dots, y_p)$ , where  $t$  is a new variable. [This method saves half of the auxiliary variables in the answer to exercise 7.1.1–55(b).]

Note: Langford's problem involves primary columns only; in an exact cover problem with *nonprimary* columns, such columns only need the constraint  $S_{\leq 1}(y_1, \dots, y_p)$ .

13. (a)  $S_1(x_1, x_2, x_3, x_4, x_5, x_6) \wedge S_1(x_7, x_8, x_9, x_{10}, x_{11}) \wedge S_1(x_{12}, x_{13}) \wedge S_1(x_{14}, x_{15}, x_{16}) \wedge S_1(x_1, x_7, x_{12}, x_{14}) \wedge S_1(x_2, x_8, x_{13}, x_{15}) \wedge S_1(x_1, x_3, x_9, x_{16}) \wedge S_1(x_2, x_4, x_7, x_{10}) \wedge S_1(x_3, x_5, x_8, x_{11}, x_{12}) \wedge S_1(x_4, x_6, x_9, x_{13}, x_{14}) \wedge S_1(x_5, x_{10}, x_{15}) \wedge S_1(x_6, x_{11}, x_{16})$ .

(b) Duplicate clauses occur when rows intersect more than once. We avoid them if we simply generate clauses  $\bar{x}_i \vee \bar{x}_j$  for every pair  $(i, j)$  of intersecting rows.

(c) When *langford*(4) is generated in this way, it has 85 distinct clauses in 16 variables, namely  $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_6) \wedge (x_7 \vee x_8 \vee x_9 \vee x_{10} \vee x_{11}) \wedge \dots \wedge (x_6 \vee x_{11} \vee x_{16}) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge \dots \wedge (\bar{x}_{15} \vee \bar{x}_{16})$ .

But *langford'*(4) cannot use the trick of (b). It has 85 (nondistinct) clauses in 20 variables, beginning with 123456,  $\bar{1}\bar{2}$ ,  $\bar{1}\bar{3}$ ,  $\bar{1}\bar{1}'$ ,  $\bar{2}\bar{3}$ ,  $\bar{2}\bar{1}'$ ,  $\bar{3}\bar{1}'$ ,  $1'\bar{4}$ ,  $1'\bar{5}$ ,  $1'\bar{6}$ ,  $\bar{4}\bar{5}$ ,  $\bar{4}\bar{6}$ ,  $\bar{5}\bar{6}$ , ... , if we denote the auxiliary variables by  $1', 2', \dots$ . Two of those clauses ( $\bar{1}\bar{3}$  and  $\bar{4}\bar{6}$ ) are repeated. (Incidentally, *langford'*(12) has 1548 clauses, 417 variables, 3600 cells.)

14. (Answer by M. Heule.) Those clauses sometimes help to focus the search. For example, if we're trying to color the complete graph  $K_n$  with  $n$  colors (or pigeons), we don't want to waste time trying  $v_2 = 1$  when  $v_1$  is already 1.

On the other hand, other instances of SAT often run slower when redundant clauses are present, because more updates to the data structures are needed.

We might also take an opposite approach, and replace (17) by  $nd$  clauses that force every color class to be a *kernel*. (See exercise 21.) Such clauses sometimes speed up a proof of uncolorability.

15. There are  $N = n(n+1)$  vertices  $(j, k)$  for  $0 \leq j \leq n$  and  $0 \leq k < n$ . If  $(j, k) = (1, 0)$  we define  $(j, k) \text{ --- } (n, i)$  for  $x \leq i < n$ , where  $x = \lfloor n/2 \rfloor$ . Otherwise we define the following edges:  $(j, k) \text{ --- } (j+1, k+1)$  if  $j < n$  and  $k < n-1$ ;  $(j, k) \text{ --- } (j+1, k)$  if  $j < n$  and  $j \neq k$ ;  $(j, k) \text{ --- } (j, k+1)$  if  $k < n-1$  and  $j \neq k+1$ ;  $(j, k) \text{ --- } (n, n-1)$  if  $j = 0$ ;  $(j, k) \text{ --- } (n-j, 0)$  if  $k < n-1$  and  $j = k$ ;  $(j, k) \text{ --- } (n+1-j, 0)$  if  $j > 0$  and  $j = k$ ;  $(j, k) \text{ --- } (n-j, n-j-1)$  if  $k = n-1$  and  $0 < j < k$ ;  $(j, k) \text{ --- } (n+1-j, n-j)$  if  $k = n-1$  and  $0 < j < n$ . Finally,  $(0, 0) \text{ --- } (1, 0)$ , and  $(0, 0) \text{ --- } (n, i)$  for  $1 \leq i \leq x$ . That makes a grand total of  $3N - 6$  edges (as it should in a maximal planar graph, according to exercise 7–46).

16. There's a unique 4-clique when  $n \geq 5$ , namely  $\{(0, n-2), (0, n-1), (1, n-1), (n, n-1)\}$ . All other vertices, except  $(0, 0)$  and  $(1, 0)$ , are surrounded by neighbors

auxiliary variables  
Langford's problem  
exact cover problem  
*nonprimary* columns  
Heule  
complete graph  
pigeons  
kernel  
maximal planar graph

that form an induced cycle of length 4 or more (usually 6). [See J.-L. Laurier, *Artificial Intelligence* **14** (1978), 117.]

**17.** Let  $mcgregor(n)$  be the clauses (15) and (16) for the graph. Add clauses (19), for symmetric threshold functions to bound the number of variables  $v_i$  for color 1; the  $k$ th vertex  $x_k$  can be specified by the ordering in answer 20. Then if, for instance, we can satisfy those clauses together with the unit clause  $s_r^N$ , where  $N = n(n+1)$ , we have proved that  $f(n) < r$ . Similarly, if we can satisfy them together with  $\bar{s}_r^N$ , we have proved that  $g(n) \geq r$ . Additional unit clauses that specify the colors of the four clique vertices will speed up the computation: Four cases should be run, one with each clique vertex receiving color 1. If all four cases are unsatisfiable, we've proved that  $f(n) \geq r$  or  $g(n) < r$ , respectively. Binary search with different values of  $r$  will identify the optimum.

For speedier  $g(n)$ , first find a maximum independent set instead of a complete 4-coloring; then notice that the colorings for  $f(n)$  already achieve this maximum.

The results turn out to be  $f(n) = (2, 2, 3, 4, 5, 7, 7, 7, 8, 9, 10, 12, 12, 12)$  for  $n = (3, 4, \dots, 16)$ , and  $g(n) = (4, 6, 10, 13, 17, 23, 28, 35, 42, 50, 58, 68, 77, 88)$ .

**18.** Assuming that  $n \geq 4$ , first assign to vertex  $(j, k)$  the following "default color":  $1 + (j + k) \bmod 3$  if  $j \leq k$ ;  $1 + (j + k + 1 - n) \bmod 3$  if  $k < j/2$ ; otherwise  $1 + (j + k + 2 - n) \bmod 3$ . Then make the following changes to exceptional vertices: Vertex  $(1, 0)$  is colored 2 if  $n \bmod 6 = 0$  or 5, otherwise 3. Vertex  $(n, n - 1)$  is colored 4. For  $k \leftarrow 0$  up to  $n - 2$ , change the color of vertex  $(n, k)$  to 4, if its default color matches vertex  $(0, 0)$  when  $k \leq n/2$  or vertex  $(1, 0)$  when  $k > n/2$ . And make final touchups for  $1 \leq j < n/2$ , depending again on  $n \bmod 6$ :

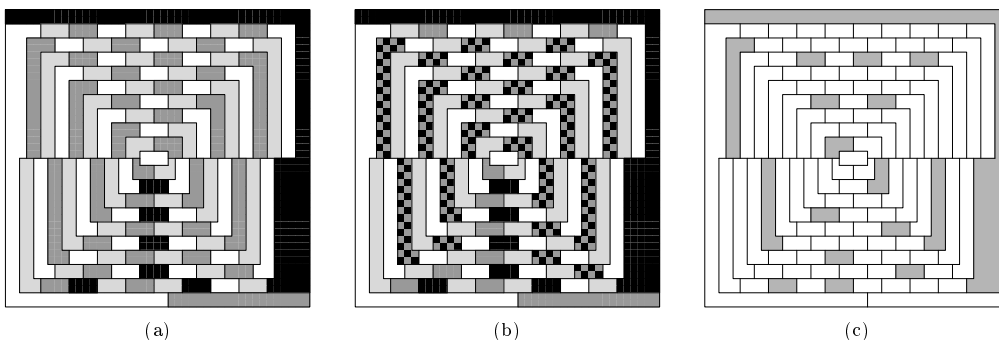
*Case 0:* Give color 4 to vertex  $(2j, j - 1)$  and color 1 to vertex  $(2j + 1, j)$ .

*Case 1:* Give color 4 to vertex  $(2j, j)$  and color 2 to vertex  $(2j + 1, j)$ .

*Case 2:* Give color 4 to vertex  $(2j, j)$  and color 1 to vertex  $(2j + 1, j)$ . Also give  $(n, n - 2)$  the color 1 and  $(n - 1, n - 3)$  the color 4.

*Cases 3, 4, 5:* Give color 4 to vertex  $(2j + 1, j)$ .

For example, the coloring for the case  $n = 10$  (found by Bryant) is shown in Fig. A-5(a).



**Fig. A-5.** Colorings and kernels of McGregor's graph.

The color distribution is  $(\lfloor n^2/3 \rfloor, \lfloor n^2/3 \rfloor, \lfloor n^2/3 \rfloor, 5k) + ((0, 1, k, -1), (1, k, 1, 0), (-1, k + 1, 1, 2), (0, k, 1, 2), (1, k + 1, 1, 2), (0, 2, k + 1, 3))$ , for  $n \bmod 6 = (0, 1, 2, 3, 4, 5)$ ,  $k = \lfloor n/6 \rfloor$ . Since this construction achieves all of the optimum values for  $f(n)$  and  $g(n)$ , when  $n \leq 16$ , it probably is optimum for all  $n$ . Moreover, the value of  $g(n)$  agrees with the size of the maximum independent set in all known cases. A further conjecture is that the maximum independent set is *unique*, whenever  $n \bmod 6 = 0$  and  $n > 6$ .

Laurier  
 $mcgregor(n)$   
 symmetric threshold functions  
 symmetry breaking  
 Binary search  
 maximum independent set  
 Bryant  
 maximum independent set





this tree-based method apparently needs one more variable and two more clauses when  $(n, r) = (7, 4)$ . But the next exercise shows that (18) and (19) don't really win!

binary recurrence relations  
recurrence relations  
Sinz

**24.** (a) The clause  $(\bar{b}_1^2 \vee \bar{b}_r^3)$  appears only if  $t_3 = r$ ; and  $t_3 \leq n/2$ .

(b) For example,  $t_3 = \min(r, 4) < r$  when  $n = 11$  and  $r = 5$ .

(c) In this case  $t_k$  is the number of leaves below node  $k$ , and the only auxiliary variables that survive pure literal elimination are  $b_{t_k}^k$ . We're left with just  $n-1$  surviving clauses, namely  $(\bar{b}_{t_{2k}}^{2k} \vee \bar{b}_{t_{2k+1}}^{2k+1} \vee b_{t_k}^k)$  for  $1 < k < n$ , plus  $(\bar{b}_{t_2}^2 \vee \bar{b}_{t_3}^3)$ .

(d) If  $2^k \leq n \leq 2^k + 2^{k-1}$  we have  $(n', n'') = (n - 2^{k-1}, 2^{k-1})$ ; on the other hand if  $2^k + 2^{k-1} \leq n \leq 2^{k+1}$  we have  $(n', n'') = (2^k, n - 2^k)$ . (Notice that  $n'' \leq n' \leq 2n''$ .)

(e) No pure literals are removed in this completely balanced case (which is the easiest to analyze). We find  $a(2^k, 2^{k-1}) = (k-1)2^k$  and  $c(2^k, 2^{k-1}) = (2^{k-2} + k - 1)2^k$ .

(f) One can show that  $a(n, r) = (r \leq n' ? b(n', r) + b(n'', r) : r \leq n' ? b(n', n'') + b(n'', n'')) : b(n', n-r) + b(n'', n-r)$ , where  $b(1, 1) = 0$  and  $b(n, r) = r + b(n', \min(r, n')) + b(n'', \min(r, n''))$  for  $n \geq 2$ . Similarly,  $c(n, r) = (r \leq n'' ? r + f(n', 0, r) + f(n'', 0, r) : r \leq n' ? n'' + f(n', r - n'', r) + f(n'', 0, n'')) : n - r + f(n', r - n'', n') + f(n'', r - n', n'')$ , where  $f(n, l, r) = \sum_{k=l+1}^r \min(k+1, n'+1, n+1-k) + (r \leq n'' ? r + f(n', 0, r) + f(n'', 0, r) : r \leq n' ? n'' + f(n', 0, r) + f(n'', 0, n'')) : r < n ? n - r + f(n', 0, n') + f(n'', 0, n'') : r - l < n'' ? f(n', n' - r + l, n') + f(n'', n'' - r + l, n'') : r - l < n' ? f(n', n' - r + l, n') + f(n'', 0, n'') : f(n', 0, n') + f(n'', 0, n''))$  for  $n \geq 2$  and  $f(1, 0, 1) = 0$ . The desired results follow by induction from these recurrence relations.

Incidentally, ternary branching can give further savings. We can, for example, handle the case  $n = 6, r = 3$  with 17 clauses in the 6 variables  $b_1^2, b_2^2, b_3^2, b_1^3, b_2^3, b_3^3$ .

**25.** From (18) and (19) we obtain  $5n - 12$  clauses in  $2n - 4$  variables, with a simple lattice-like structure. But (20) and (21) produce a more complex tree-like pattern, with  $2n - 4$  variables and with  $\lfloor n/2 \rfloor$  nodes covering just two leaves. So we get  $\lfloor n/2 \rfloor$  nodes with 3 clauses,  $n \bmod 2$  nodes with 5 clauses,  $\lceil n/2 \rceil$  nodes with 7 clauses, and 2 clauses from (21), totalling  $5n - 12$  as before (assuming that  $n > 3$ ). In fact, all but  $n - 2$  of the clauses are binary in both cases.

**26.** Imagine the boundary conditions  $s_j^0 = 1, s_j^{r+1} = 0, s_0^k = 0$ , for  $1 \leq j \leq n - r$  and  $1 \leq k \leq r$ . The clauses say that  $s_1^k \leq \dots \leq s_{n-r}^k$  and that  $x_{j+k} s_j^k \leq s_j^{k+1}$ ; so the hint follows by induction on  $j$  and  $k$ .

Setting  $j = n - r$  and  $k = r + 1$  shows that we cannot satisfy the new clauses when  $x_1 + \dots + x_n \geq r + 1$ . Conversely, if we can satisfy  $F$  with  $x_1 + \dots + x_n \leq r$  then we can satisfy (18) and (19) by setting  $s_j^k \leftarrow [x_1 + \dots + x_{j+k-1} \geq k]$ .

**27.** Argue as in the previous answer, but imagine that  $b_0^k = 1, b_{r+1}^1 = 0$ ; prove the hint by induction on  $j$  and  $n - k$  (beginning with  $k = n - 1$ , then  $k = n - 2$ , and so on).

**28.** For example, the clauses for  $\bar{x}_1 + \dots + \bar{x}_n \leq n - 1$  when  $n = 5$  are  $(x_1 \vee s_1^1), (x_2 \vee \bar{s}_1^1 \vee s_1^2), (x_3 \vee \bar{s}_1^2 \vee s_1^3), (x_4 \vee \bar{s}_1^3 \vee s_1^4), (x_5 \vee \bar{s}_1^4)$ . We may assume that  $n \geq 4$ ; then the first two clauses can be replaced by  $(x_1 \vee x_2 \vee s_1^2)$ , and the last two by  $(x_{n-1} \vee x_n \vee \bar{s}_1^{n-2})$ , yielding  $n - 2$  clauses of length 3 in  $n - 3$  auxiliary variables.

**29.** We can assume that  $1 \leq r_1 \leq \dots \leq r_n = r < n$ . Sinz's clauses (18) and (19) actually do the job nicely if we also assert that  $s_j^k$  is false whenever  $k = r_i + 1$  and  $j = i - r_i$ .

**30.** The clauses now are  $(\bar{s}_j^k \vee s_{j+1}^k), (\bar{x}_{j+k} \vee \bar{s}_j^k \vee s_j^{k+1}), (s_j^k \vee \bar{s}_j^{k+1}), (x_{j+k} \vee s_j^k \vee \bar{s}_{j+1}^k)$ , hence they define the quantities  $s_j^k = [x_1 + \dots + x_{j+k-1} \geq k]$ ; implicitly  $s_0^k = s_j^{r+1} = 0$  and  $s_j^0 = s_{n-r+1}^k = 1$ . The new clauses in answer 23 are  $\frac{1}{1} \frac{2}{1}, \frac{2}{1} \frac{3}{1}, \frac{3}{1} \frac{4}{1}, \frac{1}{2} \frac{2}{2}, \frac{2}{2} \frac{3}{2}, \frac{3}{2} \frac{4}{2}, \frac{1}{3} \frac{2}{3}, \frac{2}{3} \frac{3}{3}, \frac{3}{3} \frac{4}{3}, \frac{1}{4} \frac{2}{4}, \frac{2}{4} \frac{3}{4}, \frac{3}{4} \frac{4}{4}, \frac{1}{5} \frac{2}{5}, \frac{2}{5} \frac{3}{5}, \frac{3}{5} \frac{4}{5}, \frac{1}{6} \frac{2}{6}, \frac{2}{6} \frac{3}{6}, \frac{3}{6} \frac{4}{6}, \frac{1}{7} \frac{2}{7}, \frac{2}{7} \frac{3}{7}, \frac{3}{7} \frac{4}{7}, \frac{1}{8} \frac{2}{8}, \frac{2}{8} \frac{3}{8}, \frac{3}{8} \frac{4}{8}, \frac{1}{9} \frac{2}{9}, \frac{2}{9} \frac{3}{9}, \frac{3}{9} \frac{4}{9}, \frac{1}{10} \frac{2}{10}, \frac{2}{10} \frac{3}{10}, \frac{3}{10} \frac{4}{10}, \frac{1}{11} \frac{2}{11}, \frac{2}{11} \frac{3}{11}, \frac{3}{11} \frac{4}{11}, \frac{1}{12} \frac{2}{12}, \frac{2}{12} \frac{3}{12}, \frac{3}{12} \frac{4}{12}, \frac{1}{13} \frac{2}{13}, \frac{2}{13} \frac{3}{13}, \frac{3}{13} \frac{4}{13}, \frac{1}{14} \frac{2}{14}, \frac{2}{14} \frac{3}{14}, \frac{3}{14} \frac{4}{14}, \frac{1}{15} \frac{2}{15}, \frac{2}{15} \frac{3}{15}, \frac{3}{15} \frac{4}{15}, \frac{1}{16} \frac{2}{16}, \frac{2}{16} \frac{3}{16}, \frac{3}{16} \frac{4}{16}, \frac{1}{17} \frac{2}{17}, \frac{2}{17} \frac{3}{17}, \frac{3}{17} \frac{4}{17}, \frac{1}{18} \frac{2}{18}, \frac{2}{18} \frac{3}{18}, \frac{3}{18} \frac{4}{18}, \frac{1}{19} \frac{2}{19}, \frac{2}{19} \frac{3}{19}, \frac{3}{19} \frac{4}{19}, \frac{1}{20} \frac{2}{20}, \frac{2}{20} \frac{3}{20}, \frac{3}{20} \frac{4}{20}, \frac{1}{21} \frac{2}{21}, \frac{2}{21} \frac{3}{21}, \frac{3}{21} \frac{4}{21}, \frac{1}{22} \frac{2}{22}, \frac{2}{22} \frac{3}{22}, \frac{3}{22} \frac{4}{22}, \frac{1}{23} \frac{2}{23}, \frac{2}{23} \frac{3}{23}, \frac{3}{23} \frac{4}{23}, \frac{1}{24} \frac{2}{24}, \frac{2}{24} \frac{3}{24}, \frac{3}{24} \frac{4}{24}, \frac{1}{25} \frac{2}{25}, \frac{2}{25} \frac{3}{25}, \frac{3}{25} \frac{4}{25}, \frac{1}{26} \frac{2}{26}, \frac{2}{26} \frac{3}{26}, \frac{3}{26} \frac{4}{26}, \frac{1}{27} \frac{2}{27}, \frac{2}{27} \frac{3}{27}, \frac{3}{27} \frac{4}{27}, \frac{1}{28} \frac{2}{28}, \frac{2}{28} \frac{3}{28}, \frac{3}{28} \frac{4}{28}, \frac{1}{29} \frac{2}{29}, \frac{2}{29} \frac{3}{29}, \frac{3}{29} \frac{4}{29}, \frac{1}{30} \frac{2}{30}, \frac{2}{30} \frac{3}{30}, \frac{3}{30} \frac{4}{30}, \frac{1}{31} \frac{2}{31}, \frac{2}{31} \frac{3}{31}, \frac{3}{31} \frac{4}{31}, \frac{1}{32} \frac{2}{32}, \frac{2}{32} \frac{3}{32}, \frac{3}{32} \frac{4}{32}, \frac{1}{33} \frac{2}{33}, \frac{2}{33} \frac{3}{33}, \frac{3}{33} \frac{4}{33}, \frac{1}{34} \frac{2}{34}, \frac{2}{34} \frac{3}{34}, \frac{3}{34} \frac{4}{34}, \frac{1}{35} \frac{2}{35}, \frac{2}{35} \frac{3}{35}, \frac{3}{35} \frac{4}{35}, \frac{1}{36} \frac{2}{36}, \frac{2}{36} \frac{3}{36}, \frac{3}{36} \frac{4}{36}, \frac{1}{37} \frac{2}{37}, \frac{2}{37} \frac{3}{37}, \frac{3}{37} \frac{4}{37}, \frac{1}{38} \frac{2}{38}, \frac{2}{38} \frac{3}{38}, \frac{3}{38} \frac{4}{38}, \frac{1}{39} \frac{2}{39}, \frac{2}{39} \frac{3}{39}, \frac{3}{39} \frac{4}{39}, \frac{1}{40} \frac{2}{40}, \frac{2}{40} \frac{3}{40}, \frac{3}{40} \frac{4}{40}, \frac{1}{41} \frac{2}{41}, \frac{2}{41} \frac{3}{41}, \frac{3}{41} \frac{4}{41}, \frac{1}{42} \frac{2}{42}, \frac{2}{42} \frac{3}{42}, \frac{3}{42} \frac{4}{42}, \frac{1}{43} \frac{2}{43}, \frac{2}{43} \frac{3}{43}, \frac{3}{43} \frac{4}{43}, \frac{1}{44} \frac{2}{44}, \frac{2}{44} \frac{3}{44}, \frac{3}{44} \frac{4}{44}, \frac{1}{45} \frac{2}{45}, \frac{2}{45} \frac{3}{45}, \frac{3}{45} \frac{4}{45}, \frac{1}{46} \frac{2}{46}, \frac{2}{46} \frac{3}{46}, \frac{3}{46} \frac{4}{46}, \frac{1}{47} \frac{2}{47}, \frac{2}{47} \frac{3}{47}, \frac{3}{47} \frac{4}{47}, \frac{1}{48} \frac{2}{48}, \frac{2}{48} \frac{3}{48}, \frac{3}{48} \frac{4}{48}, \frac{1}{49} \frac{2}{49}, \frac{2}{49} \frac{3}{49}, \frac{3}{49} \frac{4}{49}, \frac{1}{50} \frac{2}{50}, \frac{2}{50} \frac{3}{50}, \frac{3}{50} \frac{4}{50}, \frac{1}{51} \frac{2}{51}, \frac{2}{51} \frac{3}{51}, \frac{3}{51} \frac{4}{51}, \frac{1}{52} \frac{2}{52}, \frac{2}{52} \frac{3}{52}, \frac{3}{52} \frac{4}{52}, \frac{1}{53} \frac{2}{53}, \frac{2}{53} \frac{3}{53}, \frac{3}{53} \frac{4}{53}, \frac{1}{54} \frac{2}{54}, \frac{2}{54} \frac{3}{54}, \frac{3}{54} \frac{4}{54}, \frac{1}{55} \frac{2}{55}, \frac{2}{55} \frac{3}{55}, \frac{3}{55} \frac{4}{55}, \frac{1}{56} \frac{2}{56}, \frac{2}{56} \frac{3}{56}, \frac{3}{56} \frac{4}{56}, \frac{1}{57} \frac{2}{57}, \frac{2}{57} \frac{3}{57}, \frac{3}{57} \frac{4}{57}, \frac{1}{58} \frac{2}{58}, \frac{2}{58} \frac{3}{58}, \frac{3}{58} \frac{4}{58}, \frac{1}{59} \frac{2}{59}, \frac{2}{59} \frac{3}{59}, \frac{3}{59} \frac{4}{59}, \frac{1}{60} \frac{2}{60}, \frac{2}{60} \frac{3}{60}, \frac{3}{60} \frac{4}{60}, \frac{1}{61} \frac{2}{61}, \frac{2}{61} \frac{3}{61}, \frac{3}{61} \frac{4}{61}, \frac{1}{62} \frac{2}{62}, \frac{2}{62} \frac{3}{62}, \frac{3}{62} \frac{4}{62}, \frac{1}{63} \frac{2}{63}, \frac{2}{63} \frac{3}{63}, \frac{3}{63} \frac{4}{63}, \frac{1}{64} \frac{2}{64}, \frac{2}{64} \frac{3}{64}, \frac{3}{64} \frac{4}{64}, \frac{1}{65} \frac{2}{65}, \frac{2}{65} \frac{3}{65}, \frac{3}{65} \frac{4}{65}, \frac{1}{66} \frac{2}{66}, \frac{2}{66} \frac{3}{66}, \frac{3}{66} \frac{4}{66}, \frac{1}{67} \frac{2}{67}, \frac{2}{67} \frac{3}{67}, \frac{3}{67} \frac{4}{67}, \frac{1}{68} \frac{2}{68}, \frac{2}{68} \frac{3}{68}, \frac{3}{68} \frac{4}{68}, \frac{1}{69} \frac{2}{69}, \frac{2}{69} \frac{3}{69}, \frac{3}{69} \frac{4}{69}, \frac{1}{70} \frac{2}{70}, \frac{2}{70} \frac{3}{70}, \frac{3}{70} \frac{4}{70}, \frac{1}{71} \frac{2}{71}, \frac{2}{71} \frac{3}{71}, \frac{3}{71} \frac{4}{71}, \frac{1}{72} \frac{2}{72}, \frac{2}{72} \frac{3}{72}, \frac{3}{72} \frac{4}{72}, \frac{1}{73} \frac{2}{73}, \frac{2}{73} \frac{3}{73}, \frac{3}{73} \frac{4}{73}, \frac{1}{74} \frac{2}{74}, \frac{2}{74} \frac{3}{74}, \frac{3}{74} \frac{4}{74}, \frac{1}{75} \frac{2}{75}, \frac{2}{75} \frac{3}{75}, \frac{3}{75} \frac{4}{75}, \frac{1}{76} \frac{2}{76}, \frac{2}{76} \frac{3}{76}, \frac{3}{76} \frac{4}{76}, \frac{1}{77} \frac{2}{77}, \frac{2}{77} \frac{3}{77}, \frac{3}{77} \frac{4}{77}, \frac{1}{78} \frac{2}{78}, \frac{2}{78} \frac{3}{78}, \frac{3}{78} \frac{4}{78}, \frac{1}{79} \frac{2}{79}, \frac{2}{79} \frac{3}{79}, \frac{3}{79} \frac{4}{79}, \frac{1}{80} \frac{2}{80}, \frac{2}{80} \frac{3}{80}, \frac{3}{80} \frac{4}{80}, \frac{1}{81} \frac{2}{81}, \frac{2}{81} \frac{3}{81}, \frac{3}{81} \frac{4}{81}, \frac{1}{82} \frac{2}{82}, \frac{2}{82} \frac{3}{82}, \frac{3}{82} \frac{4}{82}, \frac{1}{83} \frac{2}{83}, \frac{2}{83} \frac{3}{83}, \frac{3}{83} \frac{4}{83}, \frac{1}{84} \frac{2}{84}, \frac{2}{84} \frac{3}{84}, \frac{3}{84} \frac{4}{84}, \frac{1}{85} \frac{2}{85}, \frac{2}{85} \frac{3}{85}, \frac{3}{85} \frac{4}{85}, \frac{1}{86} \frac{2}{86}, \frac{2}{86} \frac{3}{86}, \frac{3}{86} \frac{4}{86}, \frac{1}{87} \frac{2}{87}, \frac{2}{87} \frac{3}{87}, \frac{3}{87} \frac{4}{87}, \frac{1}{88} \frac{2}{88}, \frac{2}{88} \frac{3}{88}, \frac{3}{88} \frac{4}{88}, \frac{1}{89} \frac{2}{89}, \frac{2}{89} \frac{3}{89}, \frac{3}{89} \frac{4}{89}, \frac{1}{90} \frac{2}{90}, \frac{2}{90} \frac{3}{90}, \frac{3}{90} \frac{4}{90}, \frac{1}{91} \frac{2}{91}, \frac{2}{91} \frac{3}{91}, \frac{3}{91} \frac{4}{91}, \frac{1}{92} \frac{2}{92}, \frac{2}{92} \frac{3}{92}, \frac{3}{92} \frac{4}{92}, \frac{1}{93} \frac{2}{93}, \frac{2}{93} \frac{3}{93}, \frac{3}{93} \frac{4}{93}, \frac{1}{94} \frac{2}{94}, \frac{2}{94} \frac{3}{94}, \frac{3}{94} \frac{4}{94}, \frac{1}{95} \frac{2}{95}, \frac{2}{95} \frac{3}{95}, \frac{3}{95} \frac{4}{95}, \frac{1}{96} \frac{2}{96}, \frac{2}{96} \frac{3}{96}, \frac{3}{96} \frac{4}{96}, \frac{1}{97} \frac{2}{97}, \frac{2}{97} \frac{3}{97}, \frac{3}{97} \frac{4}{97}, \frac{1}{98} \frac{2}{98}, \frac{2}{98} \frac{3}{98}, \frac{3}{98} \frac{4}{98}, \frac{1}{99} \frac{2}{99}, \frac{2}{99} \frac{3}{99}, \frac{3}{99} \frac{4}{99}, \frac{1}{100} \frac{2}{100}, \frac{2}{100} \frac{3}{100}, \frac{3}{100} \frac{4}{100}, \frac{1}{101} \frac{2}{101}, \frac{2}{101} \frac{3}{101}, \frac{3}{101} \frac{4}{101}, \frac{1}{102} \frac{2}{102}, \frac{2}{102} \frac{3}{102}, \frac{3}{102} \frac{4}{102}, \frac{1}{103} \frac{2}{103}, \frac{2}{103} \frac{3}{103}, \frac{3}{103} \frac{4}{103}, \frac{1}{104} \frac{2}{104}, \frac{2}{104} \frac{3}{104}, \frac{3}{104} \frac{4}{104}, \frac{1}{105} \frac{2}{105}, \frac{2}{105} \frac{3}{105}, \frac{3}{105} \frac{4}{105}, \frac{1}{106} \frac{2}{106}, \frac{2}{106} \frac{3}{106}, \frac{3}{106} \frac{4}{106}, \frac{1}{107} \frac{2}{107}, \frac{2}{107} \frac{3}{107}, \frac{3}{107} \frac{4}{107}, \frac{1}{108} \frac{2}{108}, \frac{2}{108} \frac{3}{108}, \frac{3}{108} \frac{4}{108}, \frac{1}{109} \frac{2}{109}, \frac{2}{109} \frac{3}{109}, \frac{3}{109} \frac{4}{109}, \frac{1}{110} \frac{2}{110}, \frac{2}{110} \frac{3}{110}, \frac{3}{110} \frac{4}{110}, \frac{1}{111} \frac{2}{111}, \frac{2}{111} \frac{3}{111}, \frac{3}{111} \frac{4}{111}, \frac{1}{112} \frac{2}{112}, \frac{2}{112} \frac{3}{112}, \frac{3}{112} \frac{4}{112}, \frac{1}{113} \frac{2}{113}, \frac{2}{113} \frac{3}{113}, \frac{3}{113} \frac{4}{113}, \frac{1}{114} \frac{2}{114}, \frac{2}{114} \frac{3}{114}, \frac{3}{114} \frac{4}{114}, \frac{1}{115} \frac{2}{115}, \frac{2}{115} \frac{3}{115}, \frac{3}{115} \frac{4}{115}, \frac{1}{116} \frac{2}{116}, \frac{2}{116} \frac{3}{116}, \frac{3}{116} \frac{4}{116}, \frac{1}{117} \frac{2}{117}, \frac{2}{117} \frac{3}{117}, \frac{3}{117} \frac{4}{117}, \frac{1}{118} \frac{2}{118}, \frac{2}{118} \frac{3}{118}, \frac{3}{118} \frac{4}{118}, \frac{1}{119} \frac{2}{119}, \frac{2}{119} \frac{3}{119}, \frac{3}{119} \frac{4}{119}, \frac{1}{120} \frac{2}{120}, \frac{2}{120} \frac{3}{120}, \frac{3}{120} \frac{4}{120}, \frac{1}{121} \frac{2}{121}, \frac{2}{121} \frac{3}{121}, \frac{3}{121} \frac{4}{121}, \frac{1}{122} \frac{2}{122}, \frac{2}{122} \frac{3}{122}, \frac{3}{122} \frac{4}{122}, \frac{1}{123} \frac{2}{123}, \frac{2}{123} \frac{3}{123}, \frac{3}{123} \frac{4}{123}, \frac{1}{124} \frac{2}{124}, \frac{2}{124} \frac{3}{124}, \frac{3}{124} \frac{4}{124}, \frac{1}{125} \frac{2}{125}, \frac{2}{125} \frac{3}{125}, \frac{3}{125} \frac{4}{125}, \frac{1}{126} \frac{2}{126}, \frac{2}{126} \frac{3}{126}, \frac{3}{126} \frac{4}{126}, \frac{1}{127} \frac{2}{127}, \frac{2}{127} \frac{3}{127}, \frac{3}{127} \frac{4}{127}, \frac{1}{128} \frac{2}{128}, \frac{2}{128} \frac{3}{128}, \frac{3}{128} \frac{4}{128}, \frac{1}{129} \frac{2}{129}, \frac{2}{129} \frac{3}{129}, \frac{3}{129} \frac{4}{129}, \frac{1}{130} \frac{2}{130}, \frac{2}{130} \frac{3}{130}, \frac{3}{130} \frac{4}{130}, \frac{1}{131} \frac{2}{131}, \frac{2}{131} \frac{3}{131}, \frac{3}{131} \frac{4}{131}, \frac{1}{132} \frac{2}{132}, \frac{2}{132} \frac{3}{132}, \frac{3}{132} \frac{4}{132}, \frac{1}{133} \frac{2}{133}, \frac{2}{133} \frac{3}{133}, \frac{3}{133} \frac{4}{133}, \frac{1}{134} \frac{2}{134}, \frac{2}{134} \frac{3}{134}, \frac{3}{134} \frac{4}{134}, \frac{1}{135} \frac{2}{135}, \frac{2}{135} \frac{3}{135}, \frac{3}{135} \frac{4}{135}, \frac{1}{136} \frac{2}{136}, \frac{2}{136} \frac{3}{136}, \frac{3}{136} \frac{4}{136}, \frac{1}{137} \frac{2}{137}, \frac{2}{137} \frac{3}{137}, \frac{3}{137} \frac{4}{137}, \frac{1}{138} \frac{2}{138}, \frac{2}{138} \frac{3}{138}, \frac{3}{138} \frac{4}{138}, \frac{1}{139} \frac{2}{139}, \frac{2}{139} \frac{3}{139}, \frac{3}{139} \frac{4}{139}, \frac{1}{140} \frac{2}{140}, \frac{2}{140} \frac{3}{140}, \frac{3}{140} \frac{4}{140}, \frac{1}{141} \frac{2}{141}, \frac{2}{141} \frac{3}{141}, \frac{3}{141} \frac{4}{141}, \frac{1}{142} \frac{2}{142}, \frac{2}{142} \frac{3}{142}, \frac{3}{142} \frac{4}{142}, \frac{1}{143} \frac{2}{143}, \frac{2}{143} \frac{3}{143}, \frac{3}{143} \frac{4}{143}, \frac{1}{144} \frac{2}{144}, \frac{2}{144} \frac{3}{144}, \frac{3}{144} \frac{4}{144}, \frac{1}{145} \frac{2}{145}, \frac{2}{145} \frac{3}{145}, \frac{3}{145} \frac{4}{145}, \frac{1}{146} \frac{2}{146}, \frac{2}{146} \frac{3}{146}, \frac{3}{146} \frac{4}{146}, \frac{1}{147} \frac{2}{147}, \frac{2}{147} \frac{3}{147}, \frac{3}{147} \frac{4}{147}, \frac{1}{148} \frac{2}{148}, \frac{2}{148} \frac{3}{148}, \frac{3}{148} \frac{4}{148}, \frac{1}{149} \frac{2}{149}, \frac{2}{149} \frac{3}{149}, \frac{3}{149} \frac{4}{149}, \frac{1}{150} \frac{2}{150}, \frac{2}{150} \frac{3}{150}, \frac{3}{150} \frac{4}{150}, \frac{1}{151} \frac{2}{151}, \frac{2}{151} \frac{3}{151}, \frac{3}{151} \frac{4}{151}, \frac{1}{152} \frac{2}{152}, \frac{2}{152} \frac{3}{152}, \frac{3}{152} \frac{4}{152}, \frac{1}{153} \frac{2}{153}, \frac{2}{153} \frac{3}{153}, \frac{3}{153} \frac{4}{153}, \frac{1}{154} \frac{2}{154}, \frac{2}{154} \frac{3}{154}, \frac{3}{154} \frac{4}{154}, \frac{1}{155} \frac{2}{155}, \frac{2}{155} \frac{3}{155}, \frac{3}{155} \frac{4}{155}, \frac{1}{156} \frac{2}{156}, \frac{2}{156} \frac{3}{156}, \frac{3}{156} \frac{4}{156}, \frac{1}{157} \frac{2}{157}, \frac{2}{157} \frac{3}{157}, \frac{3}{157} \frac{4}{157}, \frac{1}{158} \frac{2}{158}, \frac{2}{158} \frac{3}{158}, \frac{3}{158} \frac{4}{158}, \frac{1}{159} \frac{2}{159}, \frac{2}{159} \frac{3}{159}, \frac{3}{159} \frac{4}{159}, \frac{1}{160} \frac{2}{160}, \frac{2}{160} \frac{3}{160}, \frac{3}{160} \frac{4}{160}, \frac{1}{161} \frac{2}{161}, \frac{2}{161} \frac{3}{161}, \frac{3}{161} \frac{4}{161}, \frac{1}{162} \frac{2}{162}, \frac{2}{162} \frac{3}{162}, \frac{3}{162} \frac{4}{162}, \frac{1}{163} \frac{2}{163}, \frac{2}{163} \frac{3}{163}, \frac{3}{163} \frac{4}{163}, \frac{1}{164} \frac{2}{164}, \frac{2}{164} \frac{3}{164}, \frac{3}{164} \frac{4}{164}, \frac{1}{165} \frac{2}{165}, \frac{2}{165} \frac{3}{165}, \frac{3}{165} \frac{4}{165}, \frac{1}{166} \frac{2}{166}, \frac{2}{166} \frac{3}{166}, \frac{3}{166} \frac{4}{166}, \frac{1}{167} \frac{2}{167}, \frac{2}{167} \frac{3}{167}, \frac{3}{167} \frac{4}{167}, \frac{1}{168} \frac{2}{168}, \frac{2}{168} \frac{3}{168}, \frac{3}{168} \frac{4}{168}, \frac{1}{169} \frac{2}{169}, \frac{2}{169} \frac{3}{169}, \frac{3}{169} \frac{4}{169}, \frac{1}{170} \frac{2}{170}, \frac{2}{170} \frac{3}{170}, \frac{3}{170} \frac{4}{170}, \frac{1}{171} \frac{2}{171}, \frac{2}{171} \frac{3}{171}, \frac{3}{171} \frac{4}{171}, \frac{1}{172} \frac{2}{172}, \frac{2}{172} \frac{3}{172}, \frac{3}{172} \frac{4}{172}, \frac{1}{173} \frac{2}{173}, \frac{2}{173} \frac{3}{173}, \frac{3}{173} \frac{4}{173}, \frac{1}{174} \frac{2}{174}, \frac{2}{174} \frac{3}{174}, \frac{3}{174} \frac{4}{174}, \frac{1}{175} \frac{2}{175}, \frac{2}{175} \frac{3}{175}, \frac{3}{175} \frac{4}{175}, \frac{1}{176} \frac{2}{176}, \frac{2}{176} \frac{3}{176}, \frac{3}{176} \frac{4}{176}, \frac{1}{177} \frac{2}{177}, \frac{2}{177} \frac{3}{177}, \frac{3}{177} \frac{4}{177}, \frac{1}{178} \frac{2}{178}, \frac{2}{178} \frac{3}{178}, \frac{3}{178} \frac{4}{178}, \frac{1}{179} \frac{2}{179}, \frac{2}{179} \frac{3}{179}, \frac{3}{179} \frac{4}{179}, \frac{1}{180} \frac{2}{180}, \frac{2}{180} \frac{3}{180}, \frac{3}{180} \frac{4}{180}, \frac{1}{181} \frac{2}{181}, \frac{2}{181} \frac{3}{181}, \frac{3}{181} \frac{4}{181}, \frac{1}{182} \frac{2}{182}, \frac{2}{182} \frac{3}{182}, \frac{3}{182} \frac{4}{182}, \frac{1}{183} \frac{2}{183}, \frac{2}{183} \frac{3}{183}, \frac{3}{183} \frac{4}{183}, \frac{1}{184} \frac{2}{184}, \frac{2}{184} \frac{3}{184}, \frac{3}{184} \frac{4}{184}, \frac{1}{185} \frac{2}{185}, \frac{2}{185} \frac{3}{185}, \frac{3}{185} \frac{4}{185}, \frac{1}{186} \frac{2}{186}, \frac{2}{186} \frac{3}{186}, \frac{3}{186} \frac{4}{186}, \frac{1}{187} \frac{2}{187}, \frac{2}{187} \frac{3}{187}, \frac{3$

With (20) and (21) we can identify  $b_j^k$  with  $\bar{b}_{l_{k+1}-j}^k$ , when  $l_k > 1$  leaves are below node  $k$ . Then  $b_j^k$  is true if *and only if* the leaves below  $k$  have  $j$  or more 1s. For example, answer 23 gets the new clauses  $7_2^6, 6_2^6, 67_1^5, 5_2^5, 4_2^5, 45_1^5; 3_2^4, 2_2^4, 23_1^4; 1_3^3, 6_3^3, 1_2^3, 1_2^2, 1_1^1; 2_4^4, 2_4^4, 1_3^3, 2_2^3, 1_3^3, 1_2^2, 2_1^2, 1_1^1, 4_1^1, 3_2^2, 2_3^2$ .

Furthermore, (20) and (21) can be unified in the same way with the weaker constraints  $r' \leq x_1 + \dots + x_n \leq r$ . If we want, say,  $2 \leq x_1 + \dots + x_7 \leq 4$ , we can simply replace the final four clauses of the previous paragraph by  $\frac{4}{1} \frac{5}{1} \frac{2}{1}, \frac{2}{2} \frac{3}{1}, \frac{2}{1} \frac{3}{2}$ . Under the conventions of (18) and (19), by contrast, these weaker constraints would generate a comparable number of new clauses, namely  $\frac{1}{1} \frac{2}{1}, \frac{1}{2} \frac{2}{2}, \frac{1}{3} \frac{2}{3}, \frac{1}{4} \frac{2}{4}, \frac{1}{5} \frac{2}{5}$  and  $1_1^1, 2_1^2, 3_1^2, 3_2^2, 4_2^2, 4_3^2, 5_3^2, 5_4^2, 5_4^1, 6_4^2, 6_5^1, 7_5^2$ ; but those clauses involve the new variables  $\frac{1}{4}, \frac{1}{5}, \frac{2}{4}, \frac{2}{5}$ .

unary encoding  
 cardinality constraints, intervals  
 subinterval constraints  
 cardinality constraints, subintervals  
 symmetry  
 benchmark  
 backtrack  
 Theobald  
 Niborski  
 Erdős  
 Turán  
 Wagstaff  
 clique

**31.** We can use the constraints on the second line of (10), together with the constraints of exercise 30 that force  $x_1 + \dots + x_n = r$ . Then we seek  $n$  for which this problem is satisfiable, while the same problem with  $x_n = 0$  is not. The following small values can be used to check the calculations:

$$\begin{array}{l}
 r = 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16 \ 17 \ 18 \ 19 \ 20 \ 21 \ 22 \ 23 \ 24 \ 25 \ 26 \ 27 \\
 F_3(r) = 1 \ 2 \ 4 \ 5 \ 9 \ 11 \ 13 \ 14 \ 20 \ 24 \ 26 \ 30 \ 32 \ 36 \ 40 \ 41 \ 51 \ 54 \ 58 \ 63 \ 71 \ 74 \ 82 \ 84 \ 92 \ 95 \ 100 \\
 F_4(r) = 1 \ 2 \ 3 \ 5 \ 6 \ 8 \ 9 \ 10 \ 13 \ 15 \ 17 \ 19 \ 21 \ 23 \ 25 \ 27 \ 28 \ 30 \ 33 \ 34 \ 37 \ 40 \ 43 \ 45 \ 48 \ 50 \ 53 \\
 F_5(r) = 1 \ 2 \ 3 \ 4 \ 6 \ 7 \ 8 \ 9 \ 11 \ 12 \ 13 \ 14 \ 16 \ 17 \ 18 \ 19 \ 24 \ 25 \ 27 \ 28 \ 29 \ 31 \ 33 \ 34 \ 36 \ 37 \ 38 \\
 F_6(r) = 1 \ 2 \ 3 \ 4 \ 5 \ 7 \ 8 \ 9 \ 10 \ 12 \ 13 \ 14 \ 15 \ 17 \ 18 \ 19 \ 20 \ 22 \ 23 \ 24 \ 25 \ 26 \ 29 \ 32 \ 33 \ 35 \ 36
 \end{array}$$

Furthermore, significant speedup is possible if we also make use of previously computed values  $F_t(1), \dots, F_t(r-1)$ . For example, when  $t = 3$  and  $r \geq 5$  we must have  $x_{a+1} + \dots + x_{a+8} \leq 4$  for  $0 \leq a \leq n-8$ , because  $F_3(5) = 9$ . These additional subinterval constraints blend beautifully with those of exercise 30, because  $x_{a+1} + \dots + x_{a+p} \leq q$  for  $0 \leq a \leq n-p$  implies  $s_{b+p-q}^k \vee s_b^{k-q}$  for  $0 \leq b \leq n+1-p+q-r$  and  $q < k \leq r$ .

We can also take advantage of left-right symmetry by appending the unit clause  $\bar{s}_{\lceil (n-r)/2 \rceil}^{\lceil r/2 \rceil}$  when  $r$  is odd;  $s_{n/2-r/2+1}^{r/2}$  when  $n$  and  $r$  are both even.

Suitable benchmark examples arise when computing, say,  $F_3(27)$  or  $F_4(36)$ . But for large cases, general SAT-based methods do not seem to compete with the best special-purpose backtrack routines. For example, Gavin Theobald and Rodolfo Niborski have obtained the value  $F_3(41) = 194$ , which seems well beyond the reach of these ideas.

[See P. Erdős and P. Turán, *J. London Math. Soc.* (2) **11** (1936), 261–264; errata, **34** (1959), 480; S. S. Wagstaff, Jr., *Math. Comp.* **26** (1972), 767–771.]

**32.** Use (15) and (16), and optionally (17), but omit variable  $v_j$  unless  $j \in L(v)$ .

**33.** To double-color a graph with  $k$  colors, change (15) to the set of  $k$  clauses  $v_1 \vee \dots \vee v_{j-1} \vee v_{j+1} \vee v_k$ , for  $1 \leq j \leq k$ ; similarly,  $\binom{k}{2}$  clauses of length  $k-2$  will yield a triple coloring. Small examples reveal that  $C_{2l+1}$  for  $l \geq 2$  can be double-colored with five colors:  $\{1, 2\}(\{3, 4\}\{5, 1\})^{l-1}\{2, 3\}\{4, 5\}$ ; furthermore, seven colors suffice for triple coloring when  $l \geq 3$ :  $\{1, 2, 3\}(\{4, 5, 6\}\{7, 1, 2\})^{l-2}\{3, 4, 5\}\{6, 7, 1\}\{2, 3, 4\}\{5, 6, 7\}$ . The following exercise proves that those colorings are in fact optimum.

**34.** (a) We can obviously find a  $q$ -tuple coloring with  $q\chi(G)$  colors. And McGregor's graph has a four-clique, hence  $\chi^*(G) \geq 4$ .

(b) Any  $q$ -tuple coloring with  $p$  colors yields a solution to the fractional exact cover problem, if we let  $\lambda_j = \sum_{i=1}^p [S_j \text{ is the set of vertices colored } i] / q$ . Conversely, the theory of linear equalities tells us that there is always an optimum solution with rational  $\{\lambda_1, \dots, \lambda_N\}$ ; such a solution yields a  $q$ -tuple coloring when each  $q\lambda_j$  is an integer.

(c)  $\chi^*(C_n) = \chi(C_n) = 2$  when  $n$  is even; and  $\chi^*(C_{2l+1}) \leq 2 + 1/l = n/\alpha(C_{2l+1})$ , because there's an  $l$ -tuple coloring with  $n$  colors as in the previous exercise. Also  $\chi^*(G) \geq n/\alpha(G)$  in general:  $n = \sum_v \sum_j \lambda_j [v \in S_j] = \sum_j \lambda_j |S_j| \leq \alpha(G) \sum_j \lambda_j$ .

(d) For the hint, let  $S = \{v_1, \dots, v_l\}$  where vertices are sorted by their colors. Since vertex  $v_j$  belongs to  $C_i$  with  $|C_i| \geq |\{v_j, \dots, v_l\}|$ , we have  $t_{v_j} \leq 1/(l + 1 - j)$ .

So  $\chi(G) \leq k = \sum_v t_v = \sum_v t_v \sum_j \lambda_j [v \in S_j] = \sum_j \lambda_j \sum_v t_v [v \in S_j] \leq \sum_j \lambda_j H_{\alpha(G)}$ .

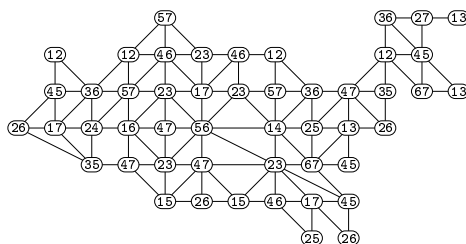
[See David S. Johnson, *J. Computer and System Sci.* **9** (1974), 264–269; L. Lovász, *Discrete Math.* **13** (1975), 383–390. The concept of fractional covering is due to A. J. W. Hilton, R. Rado, and S. H. Scott, *Bull. London Math. Soc.* **5** (1973), 302–306.]

Johnson  
Lovász  
Hilton  
Rado  
Scott  
wheel  
degree  
WalkSAT  
symmetry

**35.** (a) The double coloring below proves that  $\chi^*(G) \geq 7/2$ ; and it is optimum because  $NV$  and its neighbors induce the wheel  $W_6$ . (Notice that  $\chi^*(W_n) = 1 + \chi^*(C_{n-1})$ .)

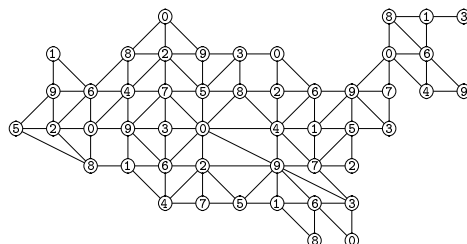
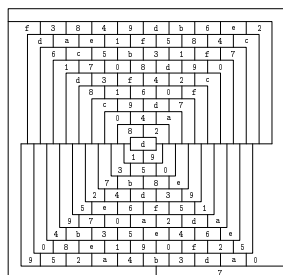
(b) By part (c) of the previous exercise,  $\chi^*(G) \geq 25/4$ . Furthermore there is a quadruple coloring with 25 colors:

AEUY ABUV BCWV CDWX DEXY  
AEFJ ABFG BCGH CDHI DEIJ  
FJKO FGKL GHLM HIMN IJNO  
KOPT KLPQ LMQR MNRS NOST  
PTUY PQUV QRVW RSWX STXY



[Is  $\overline{C_5} \boxtimes C_5$  the smallest graph for which  $\chi^*(G) < \chi(G) - 1$ ?]

**36.** A few more binary color constraints analogous to (16) yield the corresponding SAT problem. We can also assume that the upper right corner is colored 0, because that region touches  $n + 4 = 14$  others; at least  $n + 6$  colors are needed. The constraints elsewhere aren't very tight (see exercise 38(b)); thus we readily obtain an optimum radio coloring with  $n + 6$  colors for the McGregor graphs of all orders  $n > 4$ , such as the one below. An  $(n + 7)$ th color is necessary and sufficient when  $n = 3$  or 4.



**37.** The 10-coloring shown here is optimum, because Missouri (M0) has degree 8.

**38.** By looking at solutions for  $n = 10$ , say, which can be obtained quickly via Algorithm W (WalkSAT), it's easy to discover patterns that work in general: (a) Let  $(x, y)$  have color  $(2x + 4y) \bmod 7$ . (Seven colors are clearly necessary when  $n \geq 3$ .) (b) Let  $(x, y, z)$  have color  $(2x + 6y) \bmod 9$ . (Nine colors are clearly necessary when  $n \geq 4$ .)

**39.** Let  $f(n)$  denote the fewest consecutive colors. SAT solvers readily verify that  $f(n) = (1, 3, 5, 7, 8, 9)$  for  $n = (0, 1, 2, 3, 4, 5)$ . Furthermore we can exploit symmetry to show that  $f(6) > 10$ : One can assume that 000000 is colored 0, and that the colors of 000001,  $\dots$ , 100000 are increasing; that leaves only three possibilities for each of the

latter. Finally, we can verify that  $f(6) = 11$  by finding a solution that uses only the colors  $\{0, 1, 3, 4, 6, 7, 9, 10\}$ .

But  $f(7)$  is known only to be  $\geq 11$  and  $\leq 15$ .

$[L(2, 1)$  labelings were named by J. R. Griggs and R. K. Yeh, who initiated the theory in *SIAM J. Discrete Math.* **5** (1992), 586–595. The best known upper bounds, including the fact that  $f(2^k - k - 1) \leq 2^k$ , were obtained by M. A. Whittlesey, J. P. Georges, and D. W. Mauro, who also solved exercise 38(a); see *SIAM J. Discrete Math.* **8** (1995), 499–506.]

**40.** No; the satisfiable cases are  $z = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 21$ . [The statement would have been true if we'd also required  $(x_m \vee \dots \vee x_2) \wedge (y_n \vee \dots \vee y_2)$ .]

**41.** First there are  $mn$  ANDs to form  $x_i y_j$ . A bin that contains  $t$  bits initially will generate  $\lfloor t/2 \rfloor$  carries for the next bin, using  $(t-1)/2$  adders. (For example,  $t = 6$  will invoke 2 full adders and one half adder.) The respective values of  $t$  for  $\text{bin}[2]$ ,  $\text{bin}[3]$ ,  $\dots$ ,  $\text{bin}[m+n+1]$  are  $(1, 2, 4, 6, \dots, 2m-2, 2m-1, \dots, 2m-1, 2m-2, 2m-3, \dots, 5, 3, 1)$ , with  $n-m$  occurrences of  $2m-1$ . That makes a total  $mn - m - n$  full adders and  $m$  half adders; altogether we get  $mn + 2(mn - m - n) + m$  instances of AND,  $mn - m - n$  instances of OR, and  $2(mn - m - n) + m$  instances of XOR.

**42.** Ternary XOR requires quaternary clauses, but ternary clauses suffice for median:

$$\begin{array}{cccc}
 (t \vee u \vee v \vee \bar{x}) & (t \vee u \vee \bar{v} \vee x) & (t \vee u \vee \bar{y}) & (\bar{t} \vee \bar{u} \vee y) \\
 (t \vee \bar{u} \vee \bar{v} \vee \bar{x}) & (t \vee \bar{u} \vee v \vee x) & (t \vee v \vee \bar{y}) & (\bar{t} \vee \bar{v} \vee y) \\
 (\bar{t} \vee u \vee \bar{v} \vee \bar{x}) & (\bar{t} \vee u \vee v \vee x) & (u \vee v \vee \bar{y}) & (\bar{u} \vee \bar{v} \vee y) \\
 (\bar{t} \vee \bar{u} \vee v \vee \bar{x}) & (\bar{t} \vee \bar{u} \vee \bar{v} \vee x) & & 
 \end{array}$$

These clauses specify respectively that  $x \leq t \oplus u \oplus v$ ,  $x \geq t \oplus u \oplus v$ ,  $y \leq \langle tuv \rangle$ ,  $y \geq \langle tuv \rangle$ .

**43.**  $x = y = 3$  works when  $n = 2$ , but the cases  $3 \leq n \leq 7$  are unsatisfiable. We can use  $x = 3(2^{n-2} + 1)$ ,  $y = 7(2^{n-3} + 1)$  for all  $n \geq 8$ . (Such solutions seem to be quite rare. Another is  $x = 3227518467$ ,  $y = 3758194695$  when  $n = 32$ .)

**44.** First scout the territory quickly by looking at all  $\binom{N+1}{2} \approx 660$  billion cases with at most six zeros in  $x$  or  $y$ ; here  $N = \binom{32}{26} + \binom{32}{27} + \dots + \binom{32}{32}$ . This uncovers the remarkable pair  $x = 2^{32} - 2^{26} - 2^{22} - 2^{11} - 2^8 - 2^4 - 1$ ,  $y = 2^{32} - 2^{11} + 2^8 - 2^4 + 1$ , whose product is  $2^{64} - 2^{58} - 2^{54} - 2^{44} - 2^{33} - 2^8 - 1$ . Now a SAT solver finishes the job by showing that the clauses for  $32 \times 32$  bit multiplication are unsatisfiable in the presence of the further constraint  $\bar{x}_1 + \dots + \bar{x}_{32} + \bar{y}_1 + \dots + \bar{y}_{32} + \bar{z}_1 + \dots + \bar{z}_{64} \leq 15$ . (The LIFO version of the clauses worked much faster than FIFO in the author's experiments with Algorithm L. Symmetry was broken by separate runs with  $x_k \dots x_1 = 01^{k-1}$ ,  $y_k \dots y_1 = 1^k$ .)

**45.** Use the clauses for  $xy = z$  in the factorization problem, with  $m = \lfloor t/2 \rfloor$ ,  $n = \lceil t/2 \rceil$ , and  $x_j = y_j$  for  $1 \leq j \leq m$ ; append the unit clause  $(\bar{y}_n)$  if  $m < n$ .

**46.** The two largest,  $285000288617375^2$  and  $301429589329949^2$ , have 97 bits; the next square binary palindrome,  $1178448744881657^2$ , has 101. [This problem is *not* easy for SAT solvers; number theory does much better. Indeed, there's a nice way to find all  $n$ -bit examples by considering approximately  $2^{n/3}$  cases, because the rightmost  $n/3$  bits of an  $n/2$ -bit number  $x$  force the other  $n/6$  bits, if  $x^2$  is palindromic. The first eight square binary palindromes were found by G. J. Simmons, *J. Recreational Math.* **5** (1972), 11–19; all 31 solutions up to  $2^{95}$  were found by J. Schoenfeld in 2009.]

**47.** Each wire has a “top” and a “bottom.” There are  $n + g + 2h$  tops of wires, and  $m + 2g + h$  bottoms of wires. Hence the total number of wires is  $n + g + 2h = m + 2g + h$ , and we must have  $n + h = m + g$ .

Griggs  
Yeh  
Whittlesey  
Georges  
Mauro  
carries  
full adders  
half adder  
Tseytin encoding, half  
Knuth  
Symmetry was broken  
factorization  
unit clause  
SAT solvers  
number theory  
Simmons  
Schoenfeld

**48.** The wires compute  $q^1 \leftarrow q$ ,  $q^2 \leftarrow q$ ,  $x \leftarrow p \oplus q^1$ ,  $y \leftarrow q^2 \oplus r$ ,  $z \leftarrow x \oplus y$ . Let  $p$  denote “ $p$  stuck at 1” while  $\bar{p}$  denotes “ $p$  stuck at 0”. The pattern  $pqr = 000$  detects  $p$ ,  $q^1$ ,  $q^2$ ,  $r$ ,  $x$ ,  $y$ ,  $z$ ; 001 detects  $p$ ,  $q^1$ ,  $q^2$ ,  $\bar{r}$ ,  $x$ ,  $\bar{y}$ ,  $\bar{z}$ ; 010 detects  $p$ ,  $\bar{q}^1$ ,  $\bar{q}^2$ ,  $r$ ,  $\bar{x}$ ,  $\bar{y}$ ,  $z$ ; 011 detects  $p$ ,  $\bar{q}^1$ ,  $\bar{q}^2$ ,  $\bar{r}$ ,  $\bar{x}$ ,  $y$ ,  $\bar{z}$ ; 100 detects  $\bar{p}$ ,  $q^1$ ,  $q^2$ ,  $r$ ,  $\bar{x}$ ,  $y$ ,  $\bar{z}$ ; 101 detects  $\bar{p}$ ,  $q^1$ ,  $q^2$ ,  $\bar{r}$ ,  $\bar{x}$ ,  $\bar{y}$ ,  $z$ ; 110 detects  $\bar{p}$ ,  $\bar{q}^1$ ,  $\bar{q}^2$ ,  $r$ ,  $x$ ,  $\bar{y}$ ,  $\bar{z}$ ; 111 detects  $\bar{p}$ ,  $\bar{q}^1$ ,  $\bar{q}^2$ ,  $\bar{r}$ ,  $x$ ,  $y$ ,  $z$ . Notice that the stuck-at faults for  $q$  aren’t detectable (because  $z = (p \oplus q) \oplus (q \oplus r) = p \oplus r$ ); but we can detect faults on its clones  $q^1$ ,  $q^2$ . (In Fig. 34 the opposite happens.)

Three patterns such as  $\{100, 010, 001\}$  suffice for all of the detectable faults.

**49.** One finds, for example, that the faults  $b_3^2$ ,  $\bar{c}_1^2$ ,  $s^2$ , and  $\bar{q}$  are detected *only* by the pattern  $y_3y_2y_1x_2x_1 = 01111$ ;  $\bar{a}_2^2$ ,  $\bar{a}_3^2$ ,  $\bar{b}_3^2$ ,  $\bar{p}$ ,  $\bar{c}_2^2$ ,  $\bar{z}_5$  are detected only by 11011 or 11111.

All covering sets can be found by setting up a CNF with 99 positive clauses, one for each detectable fault; for example, the clause for  $\bar{z}_5$  is  $x_{27} \vee x_{31}$ , while the clause for  $x_2^2$  is  $x_4 \vee x_5 \vee x_{12} \vee x_{13} \vee x_{20} \vee x_{21} \vee x_{28} \vee x_{29}$ . We can find minimum covers from a BDD for these clauses, or by using a SAT solver with additional clauses such as (20) and (21) to limit the number of positive literals. Exactly fourteen sets of five patterns suffice, the most memorable being  $\{01111, 10111, 11011, 11101, 11110\}$ . (Indeed, every minimum set includes at least three of these five patterns.)

**50.** Primed variables for tarnished wires are  $x'_2, b'_2, b'_3, s', p', q', z'_3, c'_2, z'_4, z'_5$ . Those wires also have sharpened variables  $x_2^\sharp, b_2^\sharp, \dots, z_5^\sharp$ ; and we need sharpened variables  $x_2^{1\sharp}, x_2^{3\sharp}, x_2^{4\sharp}, b_2^{1\sharp}, b_2^{2\sharp}, b_3^{1\sharp}, b_3^{2\sharp}, s^{1\sharp}, s^{2\sharp}, c_2^{1\sharp}, c_2^{2\sharp}$  for fanout wires. The primed variables are defined by clauses such as  $(\bar{p}' \vee a_3) \wedge (\bar{p}' \vee b'_2) \wedge (p' \vee \bar{a}_3 \vee \bar{b}'_2)$ , which corresponds to  $p' \leftarrow a_3 \wedge b'_2$ . Those clauses are appended to the 49 clauses listed after (23) in the text. Then there are two clauses (25) for nine of the ten primed-and-sharpened variables; however, in the case of  $x_2$  we use the unit clauses  $(x'_2) \wedge (\bar{x}_2)$  instead, because the variable  $x_2^\sharp$  doesn’t exist. There are five fanout clauses (26), namely  $(\bar{x}_2^{1\sharp} \vee x_2^{3\sharp} \vee x_2^{4\sharp}) \wedge (\bar{b}_2^\sharp \vee b_2^{1\sharp} \vee b_2^{2\sharp}) \wedge \dots \wedge (c_2^{1\sharp} \vee c_2^{2\sharp} \vee c_2^{3\sharp})$ . There are eleven clauses  $(\bar{x}_2^{3\sharp} \vee b_2^\sharp) \wedge (\bar{x}_2^{4\sharp} \vee b_3^\sharp) \wedge (\bar{b}_2^{1\sharp} \vee s^\sharp) \wedge \dots \wedge (\bar{b}_3^{2\sharp} \vee z_5^\sharp) \wedge (c_2^{2\sharp} \vee z_5^\sharp)$  for tarnished inputs to gates. And finally there’s  $(x_2^{1\sharp}) \wedge (z_3^\sharp \vee z_4^\sharp \vee z_5^\sharp)$ .

**51.** (The complete set of 196 patterns found by the author in 2013 included the inputs  $(x, y) = (2^{32} - 1, 2^{31} + 1)$  and  $(\lceil 2^{63/2} \rceil, \lceil 2^{63/2} \rceil)$  as well as the two number-theoretic patterns mentioned in the text. Long runs of carries are needed in the products.)

**52.**  $(z_{1,2} \vee z_{2,2} \vee \dots \vee z_{M,2}) \wedge (\bar{z}_{i,2} \vee \bar{q}_{i,1}) \wedge (\bar{z}_{i,2} \vee \bar{p}_{i,2}) \wedge (\bar{z}_{i,2} \vee \bar{q}_{i,3}) \wedge (\bar{z}_{i,2} \vee \bar{p}_{i,4}) \wedge \dots \wedge (\bar{z}_{i,2} \vee \bar{q}_{i,20})$ , for  $1 \leq i \leq M$ . The second subscript of  $z$  is  $k$  in the  $k$ th case,  $1 \leq k \leq P$ .

**53.** On the left is the binary expansion of  $\pi$ , and on the right is the binary expansion of  $e$ , 20 bits at a time (see Appendix A).

One way to define  $f(x)$  for all 20-bit  $x$  is to write  $\pi/4 = \sum_{k=1}^{\infty} u_k/2^{20k}$  and  $e/4 = \sum_{l=1}^{\infty} v_l/2^{20l}$ , where each  $u_k$  and  $v_l$  is a 20-bit number. Let  $k$  and  $l$  be smallest such that  $x = u_k$  and  $x = v_l$ . Then  $f(x) = [k \leq l]$ .

Equation (27) has actually been contrived to *sustain* an illusion of magic: Many simple Boolean functions are consistent with the data in Table 2, even if we require four-term DNFs of three literals each. But only two of them, like (27), have the additional property that they actually agree with the definition of  $f(x)$  in the previous paragraph for ten more cases, using  $u_k$  up to  $k = 22$  and  $v_l$  up to  $l = 20!$  One might almost begin to suspect that a SAT solver has discovered a deep new connection between  $\pi$  and  $e$ .

**54.** (a) The function  $\bar{x}_1x_9x_{11}\bar{x}_{18} \vee \bar{x}_6\bar{x}_{10}\bar{x}_{12} \vee \bar{x}_4x_{10}\bar{x}_{12}$  matches all 16 rows of Table 2; but adding the 17th row makes a 3-term DNF impossible.

(b) 21 rows are impossible, but (27) satisfies 20 rows.

CNF  
minimum covers  
BDD  
cardinality constraints  
tarnished wires  
Knuth  
carries  
pi, as source  
e, as source  
magic

(c)  $\bar{x}_1\bar{x}_5\bar{x}_{12}x_{17}\vee\bar{x}_4x_8\bar{x}_{13}\bar{x}_{15}\vee\bar{x}_6\bar{x}_9\bar{x}_{12}x_{16}\vee\bar{x}_6\bar{x}_{13}\bar{x}_{16}x_{20}\vee x_{13}x_{14}\bar{x}_{16}$  does 28, which is max. (Incidentally, this problem makes no sense for sufficiently large  $M$ , because the equation  $f(x) = 1$  probably does not have exactly  $2^{19}$  solutions.)

cardinality constr  
covering problem  
BDD  
generating function  
don't-cares  
evaluation of Boolean functions  
truth tables

**55.** Using (28)–(31) with  $p_{i,j} = 0$  for all  $i$  and  $j$ , and also introducing clauses like (20) and (21) to ensure that  $q_{i,1} + \dots + q_{i,20} \leq 3$ , leads to solutions such as

$$f(x_1, \dots, x_{20}) = \bar{x}_1\bar{x}_7\bar{x}_8 \vee \bar{x}_2\bar{x}_3\bar{x}_4 \vee \bar{x}_4\bar{x}_{13}\bar{x}_{14} \vee \bar{x}_6\bar{x}_{10}\bar{x}_{12}.$$

(There are no monotone *increasing* solutions with  $\leq 4$  terms of *any* length.)

**56.** We can define  $f$  consistently from only a subset of the variables if and only if no entry on the left agrees with any entry on the right, when restricted to those coordinate positions. For example, the first 10 coordinates do not suffice, because the top entry on the left begins with the same 10 bits as the 14th entry on the right. The first 11 coordinates do suffice (although two entries on the right actually agree in their first 12 bits).

Let the vectors on the left be  $u_k$  and  $v_l$  as in answer 53, and form the  $256 \times 20$  matrix whose rows are  $u_k \oplus v_l$  for  $1 \leq k, l \leq 16$ . We can solve the stated problem if and only if we can find five columns for which that matrix isn't 00000 in any row. This is the classical *covering problem* (but with rows and columns interchanged): We want to find five columns that cover every row.

In general, such an  $m \times n$  covering problem corresponds to an instance of SAT with  $m$  clauses and  $n$  variables  $x_j$ , where  $x_j$  means “select column  $j$ .” The clause for a particular row is the OR of the  $x_j$  for each column  $j$  in which that row contains 1. For example, in Table 2 we have  $u_1 \oplus v_1 = 0110010011110111000$ , so the first clause is  $x_2 \vee x_3 \vee x_6 \vee \dots \vee x_{17}$ . To cover with at most five columns, we add suitable clauses according to (20) and (21); this gives 396 clauses of total length 2894, in 75 variables.

(Of course  $\binom{20}{5}$  is only 15504; we don't need a SAT solver for this simple task! Yet Algorithm D needs only 578 kilomems, and Algorithm C finds an answer in 353 K $\mu$ .)

There are 12 solutions: We can restrict to coordinates  $x_j$  for  $j$  in  $\{1, 4, 15, 17, 20\}$ ,  $\{1, 10, 15, 17, 20\}$ ,  $\{1, 15, 17, 18, 20\}$ ,  $\{4, 6, 7, 10, 12\}$ ,  $\{4, 6, 9, 10, 12\}$ ,  $\{4, 6, 10, 12, 19\}$ ,  $\{4, 10, 12, 15, 19\}$ ,  $\{5, 7, 11, 12, 15\}$ ,  $\{6, 7, 8, 10, 12\}$ ,  $\{6, 8, 9, 10, 12\}$ ,  $\{7, 10, 12, 15, 20\}$ , or  $\{8, 15, 17, 18, 20\}$ . (Incidentally, BDD methods show that the number of solutions to the covering problem has the generating function  $12z^5 + 994z^6 + 13503z^7 + \dots + 20z^{19} + z^{20}$ , counting by the size of the covering set.)

**57.** Table 2 specifies a partially defined function of 20 Boolean variables, having  $2^{20} - 32$  “don't-cares.” Exercise 56 shows how to embed it in a partially defined function of only 5 Boolean variables, in twelve different ways. So we have twelve different truth tables:

|          |          |          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|----------|----------|
| 11110110 | 0*1*010* | 10000111 | 10*0*1*0 | 00100101 | 11110*0* | 1011**** | **0**00* |
| 011*011* | 1*110100 | 10*001*1 | 1000**10 | 100*1**0 | 11*00010 | 1100**0* | *0**0101 |
| 011*1*11 | 010*100* | 10*0*000 | *101*011 | **1*1000 | 1*101100 | 1*100*10 | 0*****1* |
| 10101110 | 0*100*1* | 1*001*00 | 1**00*** | 1*1*1*10 | 10001100 | 0*101*1* | **1*0*10 |
| 10101110 | 0*1*0*10 | 1*0*1*00 | 0**01*** | 1*01*00* | 1101*0*0 | 0011*11* | 1*100*0* |
| 1*01110* | 00**110* | 11**0*00 | 10*****0 | 001*1001 | *1**1*1* | 11*0*010 | 01011001 |

And the tenth of these yields  $f(x) = ((x_8 \oplus (x_9 \vee x_{10})) \vee ((x_6 \vee x_{12}) \oplus \bar{x}_{10})) \oplus x_{12}$ .

**58.** These clauses are satisfiable whenever the other clauses are satisfiable (except in the trivial case when  $f(x) = 0$  for all  $x$ ), because we don't need to include both  $x_j$  and  $\bar{x}_j$  in the same term. Furthermore they reduce the space of possibilities by a factor of  $(3/4)^N$ . So they seem worthwhile. (On the other hand, their effect on the running time appears to be negligible, at least with respect to Algorithm C in small-scale trials.)

**59.**  $f(x) \oplus \hat{f}(x) = x_2 \bar{x}_3 \bar{x}_6 \bar{x}_{10} \bar{x}_{12} (\bar{x}_8 \vee x_8 (x_{13} \vee x_{15}))$  is a function of eight variables that has 7 solutions. Thus the probability is  $7/256 = .02734375$ .

**60.** A typical example with 32 given values of  $f(x)$ , chosen randomly, yielded

$$\hat{f}(x_1, \dots, x_{20}) = x_4 \bar{x}_7 \bar{x}_{12} \vee \bar{x}_6 x_8 \bar{x}_{11} x_{14} x_{20} \vee \bar{x}_9 \bar{x}_{12} x_{18} \bar{x}_{19} \vee \bar{x}_{13} \bar{x}_{16} \bar{x}_{17} x_{19},$$

which of course is way off; it differs from  $f(x)$  with probability  $102752/2^{18} \approx .39$ . With 64 training values, however,

$$\hat{f}(x_1, \dots, x_{20}) = x_2 \bar{x}_{13} \bar{x}_{15} x_{19} \vee \bar{x}_3 \bar{x}_9 \bar{x}_{19} \bar{x}_{20} \vee \bar{x}_6 \bar{x}_{10} \bar{x}_{12} \vee \bar{x}_8 x_{10} \bar{x}_{12}$$

comes closer, disagreeing only with probability  $404/2^{11} \approx .197$ .

**61.** We can add 24 clauses  $(p_{a,1} \vee q_{a,1} \vee p_{a,2} \vee \bar{q}_{a,2} \vee p_{a,3} \vee \bar{q}_{a,3} \vee \dots \vee p_{b,1} \vee q_{b,1} \vee \dots \vee p_{c,1} \vee q_{c,1} \vee \dots \vee p_{d,1} \vee q_{d,1} \vee \dots \vee \bar{p}_{d,10} \vee q_{d,10} \vee \dots \vee p_{d,20} \vee q_{d,20})$ , one for each permutation  $abcd$  of  $\{1, 2, 3, 4\}$ ; the resulting clauses are satisfiable only by other functions  $f(x)$ .

But the situation is more complicated in larger examples, because a function can have many equivalent representations as a short DNF. A general scheme, to decide whether the function described by a particular setting  $p'_{i,j}$  and  $q'_{i,j}$  of the  $p$ s and  $q$ s is unique, would be to add more complicated clauses, which state that  $p_{i,j}$  and  $q_{i,j}$  give a different solution. Those clauses can be generated by the Tseytin encoding of

$$\bigvee_{i=1}^M \bigwedge_{j=1}^N ((\bar{p}_{i,j} \wedge \bar{x}_j) \vee (\bar{q}_{i,j} \wedge x_j)) \oplus \bigvee_{i=1}^M \bigwedge_{j=1}^N ((\bar{p}'_{i,j} \wedge \bar{x}_j) \vee (\bar{q}'_{i,j} \wedge x_j)).$$

**62.** Preliminary experiments by the author, with  $N = 20$  and  $p = 1/8$ , seem to indicate that more data points are needed to get convergence by this method, but the SAT solver tends to run about 10 times faster. Thus, locally biased data points appear to be preferable unless the cost of observing the hidden function is relatively large.

Incidentally, the chance that  $x^{(k)} = x^{(k-1)}$  was relatively high in these experiments ( $(7/8)^{20} \approx .069$ ); so cases with  $y^{(k)} = 0$  were bypassed.

**63.** With Tseytin encoding (24), it's easy to construct  $6r + 2n - 1$  clauses in  $2r + 2n - 1$  variables that are satisfiable if and only if  $\alpha$  fails to sort the binary sequence  $x_1 \dots x_n$ . For example, the clauses when  $\alpha = [1:2][3:4][1:3][2:4][2:3]$  are  $(x_1 \vee \bar{l}_1) \wedge (x_2 \vee \bar{l}_1) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee l_1) \wedge (\bar{x}_1 \vee h_1) \wedge (\bar{x}_2 \vee h_1) \wedge (x_1 \vee x_2 \vee \bar{h}_1) \wedge \dots \wedge (l_4 \vee \bar{l}_5) \wedge (h_3 \vee \bar{l}_5) \wedge (\bar{l}_4 \vee \bar{h}_3 \vee l_5) \wedge (\bar{l}_4 \vee h_5) \wedge (\bar{h}_3 \vee h_5) \wedge (l_4 \vee h_3 \vee \bar{h}_5) \wedge (g_1 \vee g_2 \vee g_3) \wedge (\bar{g}_1 \vee l_3) \wedge (\bar{g}_1 \vee \bar{l}_5) \wedge (\bar{g}_2 \vee l_5) \wedge (\bar{g}_2 \vee \bar{h}_5) \wedge (\bar{g}_3 \vee h_5) \wedge (\bar{g}_3 \vee \bar{h}_4)$ . They're unsatisfiable, so  $\alpha$  always sorts properly.

**64.** Here we reverse the policy of the previous answer, and construct clauses that are *satisfiable* when they describe a sorting network: Let the variable  $C_{i,j}^t$  stand for the existence of comparator  $[i:j]$  at time  $t$ , for  $1 \leq i < j \leq n$  and  $1 \leq t \leq T$ . Also, adapting (20) and (21), let variables  $B_{j,k}^t$  be defined for  $1 \leq j \leq n-2$  and  $1 \leq k \leq n$ , with clauses

$$(\bar{B}_{2j,k}^t \vee \bar{B}_{2j+1,k}^t) \wedge (\bar{B}_{2j,k}^t \vee B_{j,k}^t) \wedge (\bar{B}_{2j+1,k}^t \vee B_{j,k}^t) \wedge (B_{2j,k}^t \vee B_{2j+1,k}^t \vee \bar{B}_{j,k}^t); \quad (*)$$

in this formula we substitute  $\{C_{1,k}^t, \dots, C_{k-1,k}^t, C_{k,k+1}^t, \dots, C_{k,n}^t\}$  for the  $n-1$  "leaf nodes"  $\{B_{n-1,k}^t, \dots, B_{2n-3,k}^t\}$ . These clauses prohibit comparators from clashing at time  $t$ , and they make  $B_{1,k}^t$  false if and only if line  $k$  remains unused.

If  $x = x_1 \dots x_n$  is any binary vector, let  $y_1 \dots y_n$  be the result of sorting  $x$  (so that  $(y_1 \dots y_n)_2 = 2^{\nu x} - 1$ ). The following clauses  $F(x)$  encode the fact that comparators  $C_{i,j}^t$  transform  $x \mapsto y$ :  $(\bar{C}_{i,j}^t \vee \bar{V}_{x,i}^t \vee V_{x,i}^{t-1}) \wedge (\bar{C}_{i,j}^t \vee \bar{V}_{x,i}^t \vee V_{x,j}^{t-1}) \wedge (\bar{C}_{i,j}^t \vee V_{x,i}^t \vee \bar{V}_{x,i}^{t-1} \vee \bar{V}_{x,j}^{t-1}) \wedge (\bar{C}_{i,j}^t \vee \bar{V}_{x,j}^t \vee V_{x,i}^{t-1} \vee V_{x,j}^{t-1}) \wedge (\bar{C}_{i,j}^t \vee V_{x,j}^t \vee \bar{V}_{x,i}^{t-1} \vee \bar{V}_{x,j}^{t-1}) \wedge (B_{1,i}^t \vee \bar{V}_{x,i}^t \vee V_{x,i}^{t-1}) \wedge$

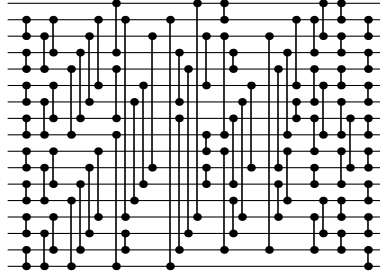


$(B_{1,i}^t \vee V_{x,i}^t \vee \overline{V}_{x,i}^{t-1})$ , for  $1 \leq i < j \leq n$  and  $1 \leq t \leq T$ ; here we substitute  $x_j$  for  $V_{x,j}^0$  and also substitute  $y_j$  for  $V_{x,j}^T$ , thereby simplifying the boundary conditions.

Furthermore, we can remove all variables  $V_{x,i}^t$  when  $x$  has  $i$  leading 0s and  $V_{x,j}^t$  when  $x$  has  $j$  trailing 1s, replacing them by 0 and 1 respectively and simplifying further.

Finally, given any sequence  $\alpha = [i_1 : j_1] \dots [i_r : j_r]$  of initial comparators,  $T$  further parallel stages will yield a sorting network if and only if the clauses  $(*)$ , together with  $\bigwedge_x F(x)$  over all  $x$  producible by  $\alpha$ , are simultaneously satisfiable.

Setting  $n = 9$ ,  $\alpha = [1:6][2:7][3:8][4:9]$ , and  $T = 5$ , we obtain 85768 clauses in 5175 variables, if we leave out the ten vectors  $x$  that are already sorted. Algorithm C finds them unsatisfiable after spending roughly 200 megamems; therefore  $\hat{T}(9) > 6$ . (Algorithm L fails spectacularly on these clauses, however.) Setting  $T \leftarrow 6$  quickly yields  $\hat{T}(9) \leq 7$ . D. Bundala and J. Závodný [LNCS 8370 (2014), 236–247] used this approach to prove in fact that  $\hat{T}(11) = 8$  and  $\hat{T}(13) = 9$ . Then T. Ehlers and M. Müller extended it [arXiv:1410.2736 [cs.DS] (2014), 10 pages], to prove that  $\hat{T}(17) = 10$ , with the surprising optimum network shown here.



Bundala  
Závodný  
Ehlers  
Müller  
CNF  
cardinality constraints  
elegant

**65.** (a) The goal is to express the transition equation in CNF. There are  $\binom{8}{4}$  clauses like  $(\bar{x}' \vee \bar{x}_a \vee \bar{x}_b \vee \bar{x}_c \vee \bar{x}_d)$ , one for each choice of four neighbors  $\{a, b, c, d\} \subseteq \{\text{NW}, \text{N}, \dots, \text{SE}\}$ . Also  $\binom{8}{7}$  clauses like  $(\bar{x}' \vee x_a \vee \dots \vee x_g)$ , one for each choice of seven. Also  $\binom{8}{6}$  like  $(\bar{x}' \vee x_a \vee \dots \vee x_f)$ , for each choice of six. Also  $\binom{8}{3}$  like  $(x' \vee \bar{x}_a \vee \bar{x}_b \vee \bar{x}_c \vee x_d \vee \dots \vee x_h)$ , complementing just three. And finally  $\binom{8}{2}$  like  $(x' \vee \bar{x} \vee \bar{x}_a \vee \bar{x}_b \vee x_c \vee \dots \vee x_g)$ , complementing just two and omitting any one of the others. Altogether  $70 + 8 + 28 + 56 + 28 = 190$  clauses of average length  $(70 \cdot 5 + 8 \cdot 8 + 28 \cdot 8 + 56 \cdot 9 + 28 \cdot 9) / 190 \approx 7.34$ .

(b) Here we let  $x = x_{ij}$ ,  $x_{\text{NW}} = x_{(i-1)(j-1)}, \dots, x_{\text{SE}} = x_{(i+1)(j+1)}$ ,  $x' = x'_{ij}$ . There are seven classes of auxiliary variables  $a_k^{ij}, \dots, g_k^{ij}$ , each of which has two children; the meaning is that the sum of the descendants is  $\geq k$ . We have  $k \in \{2, 3, 4\}$  for the  $a$  variables,  $k \in \{1, 2, 3, 4\}$  for the  $b$  and  $c$  variables, and  $k \in \{1, 2\}$  for  $d, e, f, g$ .

The children of  $a^{ij}$  are  $b^{(i+1)j}$  and  $c^{ij}$ . The children of  $b^{ij}$  are  $d^{i(j-(j\&2))}$  and  $e^{i(j+(j\&2))}$ . The children of  $c^{ij}$  are  $f^{i'j'}$  and  $g^{ij}$ , where  $i' = i+2$  and  $j' = (j-1) \mid 1$  if  $i$  is odd, otherwise  $i' = i$  and  $j' = j - (j \& 1)$ . The children of  $d^{ij}$  are  $x_{(i-1)(j+1)}$  and  $x_{i(j+1)}$ . The children of  $e^{ij}$  are  $x_{(i-1)(j-1)}$  and  $x_{i(j-1)}$ . The children of  $f^{ij}$  are  $x_{(i-1)j}$  and  $x_{(i-1)(j+1)}$ . Finally, the children of  $g^{ij}$  are  $x_{i'j}$  and  $x_{i''j''}$ , where  $i' = i+1 - ((i \& 1) \ll 1)$ ; and  $(i'', j'') = (i+1, j \oplus 1)$  if  $i$  is odd, otherwise  $(i'', j'') = (i-1, j-1 + ((j \& 1) \ll 1))$ . (OK — this isn't elegant. But hey, it works!)

If the children of  $p$  are  $q$  and  $r$ , the clauses that define  $p_k$  are  $(p_k \vee \bar{q}_{k'} \vee \bar{r}_{k''})$  for  $k' + k'' = k$  and  $(\bar{p}_k \vee q_{k'} \vee r_{k''})$  for  $k' + k'' = k + 1$ . In these clauses we omit  $\bar{q}_0$  or  $\bar{r}_0$ ; we also omit  $q_m$  or  $r_m$  when  $q$  or  $r$  has fewer than  $m$  descendants.

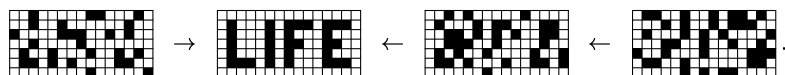
For example, these rules define  $d_1^{35}$  and  $d_2^{35}$  by the following six clauses:

$$(d_1^{35} \vee \bar{x}_{26}), (d_1^{35} \vee \bar{x}_{36}), (d_2^{35} \vee \bar{x}_{26} \vee \bar{x}_{36}), (\bar{d}_1^{35} \vee x_{26} \vee x_{36}), (\bar{d}_2^{35} \vee x_{26}), (\bar{d}_2^{35} \vee x_{36}).$$

The variables  $b_k^{ij}$  are defined only when  $i$  is odd;  $d_k^{ij}$  and  $e_k^{ij}$  only when  $i$  is odd and  $j \bmod 4 < 2$ ;  $f_k^{ij}$  only when  $i+j$  is even. Thus the total number of auxiliary variables per cell  $(i, j)$ , ignoring small corrections at boundary points, is  $3 + 4/2 + 4 + 2/4 + 2/4 + 2/2 + 2 = 13$  of types  $a$  through  $g$ , not 19, because of the sharing; and the total number of clauses per cell to define them is  $21 + 16/2 + 16 + 6/4 + 6/4 + 6/2 + 6 = 57$ , not 77.

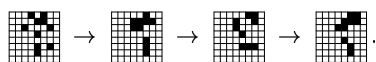
Finally we define  $x'_{ij}$  from  $a_2^{ij}, a_3^{ij}, a_4^{ij}$ , by means of six clauses  $(\bar{x}'_{ij} \vee \bar{a}_4^{ij}), (\bar{x}'_{ij} \vee a_2^{ij}), (\bar{x}'_{ij} \vee x_{ij} \vee a_3^{ij}), (x'_{ij} \vee a_4^{ij} \vee \bar{a}_3^{ij}), (x'_{ij} \vee \bar{x}_{ij} \vee \bar{y}_{ij}), (y_{ij} \vee a_4^{ij} \vee \bar{a}_2^{ij})$ , where  $y_{ij}$  is another auxiliary variable (introduced only to avoid clauses of size 4).

**66.** All solutions to (a) can be characterized by a BDD of 8852 nodes, from which we can obtain the generating function  $38z^{28} + 550z^{29} + \dots + 150z^{41}$  that enumerates them (with a total computation time of only 150 megamems or so). Part (b), however, is best suited to SAT, and  $X_0$  must have at least 38 live cells. Typical answers are

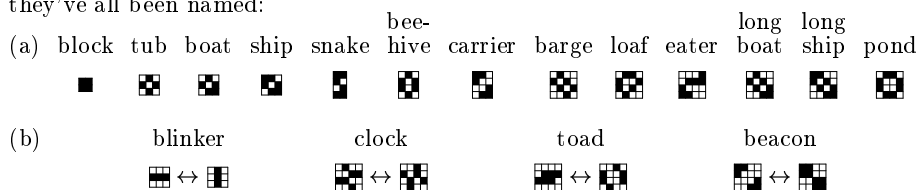


**67.** Either or at lower left will produce the  $X_0$  of (37) at time 1. But length 22 is impossible: With  $r = 4$  we can verify that all the live cells in  $X_4$  lie in some  $3 \times 3$  subarray. Then with  $r = 22$  we need to rule out only  $\binom{9}{3} + \binom{9}{4} + \binom{9}{5} \times 6 = 2016$  possibilities, one for each viable  $X_4$  within each essentially different  $3 \times 3$  subarray.

**68.** The author believes that  $r = 12$  is impossible, but his SAT solvers have not yet been able to verify this conjecture. Certainly  $r = 11$  is achievable, because we can continue with the text's fifth example after prepending



**69.** Since only 8548 essentially different  $4 \times 4$  bitmaps are possible (see Section 7.2.3), an exhaustive enumeration is no sweat. The small stable patterns arise frequently, so they've all been named:



(A glider is also considered to be stable, although it's not an oscillator.)

**70.** (a) A cell with three live neighbors in the stator will stay alive.  
 (b) A  $4 \times n$  board doesn't work; Fig. A-6 shows the  $5 \times 8$  examples.  
 (c) Again, the smallest-weight solutions with smallest rectangles are shown in Fig. A-6. Oscillators with these rotors are plentiful on larger boards; the first examples of each kind were found respectively by Richard Schroepel (1970), David Buckingham (1972), Robert Wainwright (1985).

**71.** Let the variables  $X_t = x_{ijt}$  characterize the configuration at time  $t$ , and suppose we require  $X_r = X_0$ . There are  $q = 8r$  automorphisms  $\sigma$  that take  $X_t \mapsto X_{(t+p) \bmod r \tau}$ , where  $0 \leq p < r$  and  $\tau$  is one of the eight symmetries of a square grid.

Any global permutation of the  $N = n^2 r$  variables leads via Theorem E to a canonical form, where we require the solution to be lexicographically less than or equal to the  $q - 1$  solutions that are equivalent to it under automorphisms.

Such lexicographic tests can be enforced by introducing  $(q-1)(3N-2)$  new clauses of length  $\leq 3$ , as in (169) — and often greatly simplified using Corollary E.

These additional clauses can significantly speed up a proof of unsatisfiability. On the other hand they can also slow down the search, if a problem has abundant solutions.

BDD  
 author  
 stable  
 Schroepel  
 Buckingham  
 Wainwright  
 automorphisms

In practice it's usually better to insist only on solutions that are *partially* canonical, by using only some of the automorphisms and by requiring lexicographic order only on some of the variables.

Wainwright  
Flammenkamp  
eightfold symmetry  
encoding  
Gosper  
phoenix  
quilt patterns

72. (a) The two  $7 \times 7$ s, shown in Fig. A-6, were found by R. Wainwright (trice tongs, 1972) and A. Flammenkamp (jam, 1988).

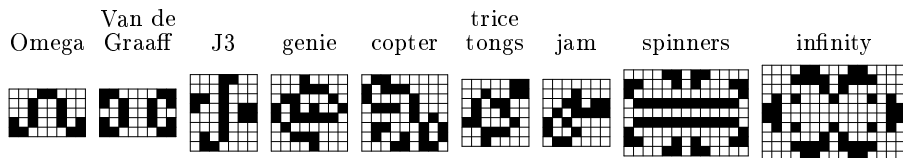
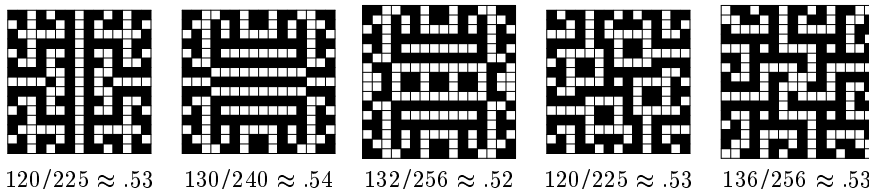


Fig. A-6. Noteworthy minimal oscillators of periods 2 and 3.

(b) Here the smallest examples are  $9 \times 13$  and  $10 \times 15$ ; the former has four L-rotors surrounding long stable lines. Readers will also enjoy discovering  $10 \times 10$  and  $13 \times 13$  instances that have full eightfold symmetry. (When encoding such symmetrical problems by using exercise 65(b), we need only compute the transitions between variables  $x_{ij}$  for  $1 \leq i \leq \lceil m/2 \rceil$  and  $1 \leq j \leq \lceil n/2 \rceil$ ; every other variable is identical to one of these. However, the auxiliary variables  $a^{ij}, \dots, g^{ij}$  shouldn't be coalesced in this way.)

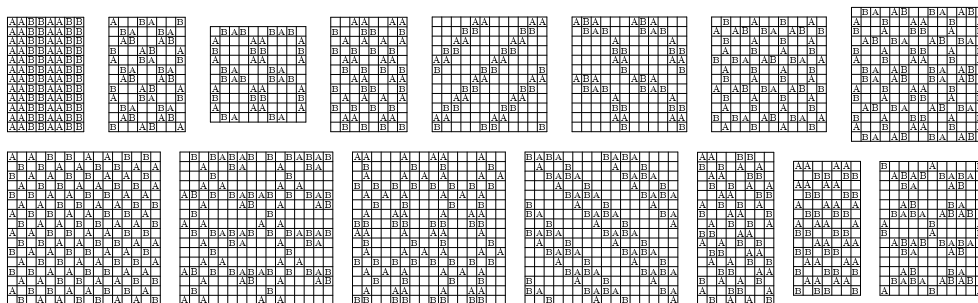
(c,d) Champion heavyweights have small rotors. What a cool four-way snake dance!



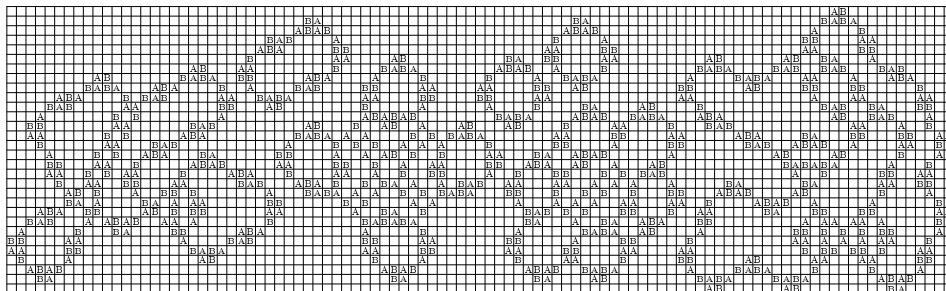
73. (a) They don't have three A neighbors; and they don't have three B neighbors.

(b) Two examples appear in Fig. A-7, where they are packed as snugly as possible into a  $12 \times 15$  box. This pattern, found by R. W. Gosper about 1971, is called the *phoenix*, since its living cells repeatedly die and rise again. It is the smallest mobile flipflop; the same idea yields the next smallest (also seen in Fig. A-7), which is  $10 \times 12$ .

(c) The nonblank one comes from a  $1 \times 4$  torus; the checkerboard from an  $8 \times 8$ . Here are some amazing  $m \times n$  ways to satisfy the constraints for small  $m$  and  $n$ :



Notice that infinite one-dimensional examples are implied by several of these motifs; the checkerboard, in fact, can be fabricated by placing  $\begin{matrix} A & B \\ B & A \end{matrix}$  diagonals together.



**Fig. A-7.** Mobile flipflops: An ideal way to tile the floor of a workspace for hackers.

**74.** Call a cell *tainted* if it is A with more than one A neighbor or B with more than one B neighbor. Consider the topmost row with a tainted cell, and the leftmost tainted cell in that row. We can assume that this cell is an A, and that its neighbors are S, T, U, V, W, X, Y, Z in the pattern  $\begin{array}{c} \text{S} \\ \text{T} \\ \text{U} \\ \text{V} \\ \text{W} \\ \text{X} \\ \text{Y} \\ \text{Z} \end{array}$ . Three of those eight neighbors are type B, and at least four are type A; several cases need to be considered.

*Case 1:*  $W = X = Y = Z = A$ . Then we must have  $S = U = V = B$  and  $T = 0$  (blank), because S, T, U, V aren't tainted. The three left neighbors of V can't be type A, since V already has three A neighbors; nor can they be type B, since V isn't tainted. Hence the tainted X, which must have two B neighbors in the three cells below it, cannot also have two or more A neighbors there.

*Case 2:*  $T = A$  or  $V = A$ . If, say,  $T = A$  then  $X = Y = Z = A$ , and neither V nor W can be type B.

*Case 3:*  $S \neq A$ ,  $U = A$ . Then W can't be type B, and S must be tainted.

*Case 4:*  $S = A$ ,  $U \neq A$ . At least one of W, X, Y, Z is B; at least three are A; so exactly three are A. The B can't be Y, which has four A neighbors. Nor can it be W or Z: That would force V to be blank, hence  $T = U = B$ ; consequently  $W = A$ ,  $Z = B$ . Since W is tainted, at least two of its right neighbors must be A, contradicting  $Z = B$ .

Thus  $X = B$  in Case 4. Either T or V is also B, while the other is blank; say T is blank. The three left neighbors of V cannot be A. So they must either all be B (tainting the cell left of S) or all blank. In the latter case the upper neighbors of T must be BBA in that order, since T is blank. But that taints the B above T. A symmetric argument applies if V is blank.

*Case 5:*  $S = U = A$ . Then  $W \neq A$ , and at least two of  $\{X, Y, Z\}$  are A. Now  $Y = Z = A$  forces  $T = V = X = B$  and W blank, tarnishing V.

Similarly,  $X = Y = A$  forces  $T = W = Z = B$  and V blank; this case is more difficult. The three lower neighbors of Y must be AAB, in that order, lest a B be surrounded by four A's. But then the left neighbors of X are BBB; hence so are the left neighbors of V, tarnishing the middle one.

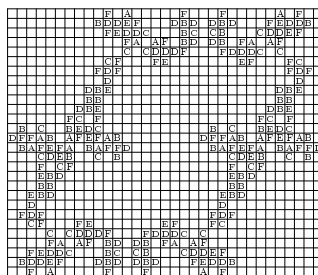
Finally, therefore, Case 5 implies that  $X = Z = A$ . Either T, V, W, or Y is blank; the other three are B. The blank can't be T, since T's upper three neighbors can't be A. It can't be W or Y, since V and T aren't tainted. So  $T = W = Y = B$  and V is blank. The left neighbors of S cannot be A, because S isn't tainted. So the cell left of X must be A. Therefore X must have at least four A neighbors; but that's impossible, because Y already has three.

Diagonally adjacent A's are rare. (In fact, they cannot occur in rectangular grids of size less than  $16 \times 18$ .) But diligent readers will be able to spot them in Fig. A-7, which exhibits an astonishing variety of different motifs that are possible in large grids.

**75.** Let the cells alive at times  $p - 2, p - 1, p$  be of types  $X, Y, Z$ , and consider the topmost row in which a live cell ever appears. Without loss of generality, the leftmost cell in that row is type  $Z$ . The cell below that  $Z$  can't be of type  $Y$ , because that  $Y$  would have three  $X$  neighbors and four  $Y$  neighbors besides  $Z$  and the blank to its left.

Thus the picture must look like  $\begin{matrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{matrix}$ , where the three predecessors of  $Z$  and the topmost  $Y$  are filled in. But there's no room for the three predecessors of the topmost  $X$ .

**76.** The smallest known example, a  $28 \times 33$  pattern found by Jason Summers in 2012, is illustrated here using the letters  $\{F, A, B\}, \{B, C, D\}, \{D, E, F\}$  for cells that are alive when  $t \bmod 3 = 0, 1, 2$ . His ingenious construction leads in particular to an infinite solution based on a  $7 \times 24$  torus. An amazing infinite  $7 \times 7$  toroidal pattern also exists, but little else is yet known.



Summers  
torus  
Rokicki  
still life  
symeater  
blocks  
carriers  
Buckingham  
Silver  
pixels  
glider's symmetry

**77.** If the first four cells in row 4 of  $X_0$  (and of  $X_5$ ) contain  $a, b, c, d$ , we need  $a + b \neq 1, a + b + c \neq 1, b + c + d \neq 2$ . In clause form this becomes  $\bar{a} \vee \bar{b}, a \vee \bar{b}, b \vee \bar{c}, \bar{c} \vee d, \bar{b} \vee c \vee \bar{d}$ .

Similarly, let the last four elements of column 5 be  $(f, g, h, i)$ ; then we want  $f + g + h \neq 2, g + h + i \neq 2, h + i \neq 2$ . These conditions simplify to  $\bar{f} \vee \bar{g}, \bar{f} \vee \bar{h}, \bar{g} \vee \bar{i}, \bar{h} \vee \bar{i}$ .

**78.** The "9<sup>2</sup> phage" in Fig. A-8 is a minimal example.

**79.** (Solution by T. G. Rokicki.) A tremendous battle flares up, raging wildly on all fronts. When the dust finally settles at time 1900, 11 gliders are escaping the scene (1 going in the original NE direction, 3 going NW, 5 going SW, and 2 going SE), leaving behind 16 blocks, 1 tub, 2 loaves, 3 boats, 4 ships, 8 beehives, 1 pond, 15 blinkers, and 1 toad. (One should really watch this with a suitable applet.)

**80.** Paydirt is hit on  $10 \times 10$  and  $11 \times 11$  boards, with  $X_8 = X_9$ ; see Fig. A-8. The minimal example, "symeater19," has a close relative, "symeater20," which consists simply of two blocks and two carriers, strategically placed. (The first of these, also called "eater 2," was discovered by D. Buckingham in the early 1970s; the other by S. Silver in 1998.) They both have the additional ability to eat the glider if it is moved one or two cells to the right of the position shown, or one cell to the left.

It is important to realize that the diagonal track of a glider does *not* pass through the corners, bisecting them; the axis of a glider's symmetry actually passes through the *midpoints* of pixel edges, thereby cutting off small triangles whose area is  $1/8$  of a full pixel. Consequently, any eater that is symmetric about a diagonal will eat gliders in two adjacent tracks. The two in Fig. A-8 are exceptional because they're quadruply effective. Furthermore symeater20 will eat from the opposite direction; and either of its carriers can be swapped to another position next to the blocks.

**81.** Two eaters make "ssymeater14" (Fig. A-8); and "ssymeater22" is narrower.

**82.** (a) If  $X \rightarrow X'$ , then  $x'_{ij} = 1$  only if we have  $\sum_{i'=i-1}^{i+1} \sum_{j'=j-1}^{j+1} x_{i'j'} \geq 3$ .  
 (b) Use the same inequality, and induction on  $j$ .



Such an approach is out of the question for  $10 \times 10$  bitmaps, because  $2^{100} \gg 2^{36}$ . But we *can* find all  $10 \times 10$  Gardens of Eden for which there is  $90^\circ$ -rotational symmetry, by trying only about  $2^{25}/2$  patterns, again using Gray code. Aha: Eight such patterns have no predecessor, and four of them correspond to the given orphan.

[See C. Hartman, M. J. H. Heule, K. Kwekkeboom, and A. Noels, *Electronic J. Combinatorics* **20**, 3 (August 2013), #P16. The existence of Gardens of Eden with respect to many kinds of cellular automata was first proved nonconstructively by E. F. Moore, *Proc. Symp. Applied Math.* **14** (1962), 17–33.]

**86.** The 80 cells outside the inner  $8 \times 8$  can be chosen in  $N = 11,984,516,506,952,898$  ways. (A BDD of size 53464 proves this.) So the answer is  $N/2^{100-64} \approx 174,398$ .

**87.** Instead of using subscripts  $t$  and  $t + 1$ , we can write the transition clauses for  $X \rightarrow X'$  in the form  $(@ \vee \bar{A}0 \vee A0')$ , etc. Let Alice's states be  $\{\alpha_1, \dots, \alpha_p\}$  and let Bob's be  $\{\beta_1, \dots, \beta_q\}$ . The clauses  $(@ \vee \bar{\alpha}_i \vee \alpha'_i)$  and  $(\bar{@} \vee \bar{\beta}_i \vee \beta'_i)$  say that your state doesn't change unless you are bumped. If state  $\alpha$  corresponds to the command 'Maybe go to  $s$ ', the clause  $(\bar{@} \vee \bar{\alpha} \vee \alpha' \vee s')$  defines the next possible states after bumping. The analogous clause for 'Critical, go to  $s$ ' or 'Set  $v \leftarrow b$ , go to  $s$ ' is simply  $(\bar{@} \vee \bar{\alpha} \vee s')$ ; and the latter also generates the clause  $(\bar{@} \vee \bar{\alpha} \vee v')$  if  $b = 1$ ,  $(\bar{@} \vee \bar{\alpha} \vee \bar{v}')$  if  $b = 0$ . The command 'If  $v$  go to  $s_1$ , else to  $s_0$ ' generates  $(\bar{@} \vee \bar{\alpha} \vee \bar{v} \vee s'_1) \wedge (\bar{@} \vee \bar{\alpha} \vee v \vee s'_0)$ . And for each variable  $v$ , if the states whose commands set  $v$  are  $\alpha_{i_1}, \dots, \alpha_{i_h}$ , the clauses

$$(\bar{@} \vee v \vee \alpha_{i_1} \vee \dots \vee \alpha_{i_h} \vee \bar{v}') \wedge (\bar{@} \vee \bar{v} \vee \alpha_{i_1} \vee \dots \vee \alpha_{i_h} \vee v')$$

encode the fact that  $v$  isn't changed by other commands.

Bob's program generates similar clauses — but they use @, not  $\bar{@}$ , and  $\beta$ , not  $\alpha$ .

Incidentally, when other protocols are considered in place of (40), the initial state  $X_0$  analogous to (41) is constructed by putting Alice and Bob into their smallest possible states, and by setting all shared variables to 0.

**88.** For example, let all variables be false except  $A0_0, B0_0, @_0, A1_1, B0_1, A1_2, B1_2, A1_3, B2_3, @_3, A2_4, B2_4, @_4, A3_5, B2_5, l_5, A3_6, B3_6, l_6$ .

**89.** No; we can find a counterexample to the corresponding clauses as in the previous exercise:  $A0_0, B0_0, A0_1, B1_1, A0_2, B2_2, b_2, @_2, A1_3, B2_3, b_3, A1_4, B3_4, b_4, A1_5, B4_5, b_5, @_5, A2_6, B4_6, a_6, b_6, @_6, A5_7, B4_7, a_7, b_7, A5_8, B2_8, a_8, b_8, l_8, A5_9, B5_9, a_9, b_9, l_9$ .

(This protocol was the author's original introduction to the fascinating problem of mutual exclusion [see *CACM* **9** (1966), 321–322, 878], about which Dijkstra had said "Quite a collection of trial solutions have been shown to be incorrect.")

**90.** Alice starves in (43) with  $p = 1$  and  $r = 3$  in (47), if she moves to A1 and then Bob remains in B0 whenever he is bumped. The  $A2 \wedge B2$  deadlock mentioned in the text for (45) corresponds to (47) with  $p = 4$  and  $r = 6$ . And in (46), successive moves to  $B1, (B2, A1, A2, B3, B1, A4, A5, A0)^\infty$  will starve poor Bob.

**91.** A cycle (47) with no maybe/critical states for Alice can certainly starve her. Conversely, given (i), (ii), (iii), suppose Alice is in no maybe/critical state when  $t \geq t_0$ ; and let  $t_0 < t_1 < t_2 < \dots$  be times with  $@_{t_i} = 1$  but with  $@_t = 0$  for at least one  $t$  between  $t_i$  and  $t_{i+1}$ . Then we must have  $X_{t_i} = X_{t_j}$  for some  $i < j$ , because the number of states is finite. Hence there's a starvation cycle with  $p = t_i$  and  $r = t_j$ .

**92.** For  $0 \leq i < j \leq r$  we want clauses that encode the condition  $X_i \neq X_j$ . Introduce new variables  $\sigma_{ij}$  for each state  $\sigma$  of Alice or Bob, and  $v_{ij}$  for each shared variable  $v$ . Assert that at least one of these new variables is true. (For the protocol (40) this clause

$90^\circ$ -rotational symmetry  
Hartman  
Heule  
Kwekkeboom  
Noels  
cellular automata  
nonconstructively  
Moore  
BDD  
transition clauses  
initial state  
Knuth  
Dijkstra  
encode

would be  $(A0_{ij} \vee \dots \vee A4_{ij} \vee B0_{ij} \vee \dots \vee B4_{ij} \vee l_{ij})$ . Also assert the binary clauses  $(\bar{\sigma}_{ij} \vee \sigma_i) \wedge (\bar{\sigma}_{ij} \vee \bar{\sigma}_j)$  for each  $\sigma$ , and the ternary clauses  $(\bar{v}_{ij} \vee v_i \vee v_j) \wedge (\bar{v}_{ij} \vee \bar{v}_i \vee \bar{v}_j)$  for each  $v$ .

The transition clauses can also be streamlined, because we needn't allow cases where  $X_{t+1} = X_t$ . Thus, for example, we can omit  $B0_{t+1}$  from the clause  $(@_t \vee \bar{B}0_t \vee B0_{t+1} \vee B1_{t+1})$  of (42); and we can omit the clause  $(@_t \vee \bar{B}1_t \vee \bar{l}_t \vee B1_{t+1})$  entirely.

[If  $r$  is large, encodings with  $O(r(\log r)^2)$  clauses are possible via sorting networks, as suggested by D. Kroening and O. Strichman, *LNCS 2575* (2003), 298–309. The most practical scheme, however, seems to be to add the  $ij$  constraints one by one as needed; see N. Eén and N. Sörensson, *Electronic Notes in Theoretical Computer Science 89* (2003), 543–560.]

**93.** For the  $\Phi$  in (50), for example, we can use  $(x_1 \vee x_2 \vee \dots \vee x_{16}) \wedge (\bar{x}_1 \vee \bar{A}0') \wedge \dots \wedge (\bar{x}_1 \vee \bar{A}6') \wedge (\bar{x}_2 \vee \bar{B}0') \wedge \dots \wedge (\bar{x}_2 \vee \bar{B}6') \wedge (\bar{x}_3 \vee \bar{A}0') \wedge (\bar{x}_3 \vee \bar{a}') \wedge \dots \wedge (\bar{x}_{16} \vee \bar{B}6') \wedge (\bar{x}_{16} \vee \bar{b}')$ .

**94.**  $(X \rightarrow X' \rightarrow \dots \rightarrow X^{(r)}) \wedge \Phi(X) \wedge \Phi(X') \wedge \dots \wedge \Phi(X^{(r-1)}) \wedge \neg\Phi(X^{(r)})$ . [This important technique is called “ $k$ -induction”; see Mary Sheeran, Satnam Singh, and Gunnar Stålmarck, *LNCS 1954* (2000), 108–125. One can, for example, add the clause  $(\bar{A}5 \vee \bar{B}5)$  to (50) and prove the resulting formula  $\Phi$  by 3-induction.]

**95.** The critical steps have  $a = b = 1$ , by the invariants, so they have no predecessor.

**96.** The only predecessor of  $A5_2 \wedge B5_2 \wedge a_2 \wedge b_2 \wedge \bar{l}_2$  is  $A5_1 \wedge B4_1 \wedge a_1 \wedge b_1 \wedge \bar{l}_1$ ; and the only predecessor of *that* is  $A5_0 \wedge B3_0 \wedge a_0 \wedge b_0 \wedge \bar{l}_0$ . The case  $l_2$  is similar.

But *without* the invariants, we could find arbitrarily long paths to  $A5_r \wedge B5_r$ . In fact the longest such *simple* path has  $r = 33$ : Starting with  $A2_0 \wedge B2_0 \wedge \bar{a}_0 \wedge \bar{b}_0 \wedge l_0$ , we could successively bump Alice and Bob into states  $A3, A5, A6, A0, A1, A2, A3, B3, B4, A5, B3, A6, B4, A0, B3, A1, A2, A3, A5, A6, A0, A1, A2, B4, A3, A5, A6, A0, B5, A1, A2, A3, A5$ , never repeating a previous state. (Of course all of these states are unreachable from the real  $X_0$ , because none of them satisfy  $\Phi$ .)

**97.** No. Removing each person's final step in a path to  $A6 \wedge B6$  gives a path to  $A5 \wedge B5$ .

**98.** (a) Suppose  $X_0 \rightarrow \dots \rightarrow X_r = X_0$  is impure and  $X_i = X_j$  for some  $0 \leq i < j < r$ . We may assume that  $i = 0$ . If either of the two cycles  $X_0 \rightarrow \dots \rightarrow X_j = X_0$  or  $X_j \rightarrow \dots \rightarrow X_r = X_j$  is impure, it is shorter.

(b) In those states she would have had to be previously in  $A0$  or  $A5$ .

(c) Generate clauses  $(\bar{g}_0), (\bar{g}_t \vee g_{t-1} \vee @_{t-1}), (\bar{h}_0), (\bar{h}_t \vee h_{t-1} \vee @_{t-1}), (\bar{f}_t \vee g_t), (\bar{f}_t \vee h_t), (\bar{f}_t \vee \alpha_0 \vee \bar{\alpha}_t), (\bar{f}_t \vee \bar{\alpha}_0 \vee \alpha_t), (\bar{f}_t \vee v_0 \vee \bar{v}_t), (\bar{f}_t \vee \bar{v}_0 \vee v_t)$ , for  $1 \leq t \leq r$ ; and  $(f_1 \vee f_2 \vee \dots \vee f_r)$ . Here  $v$  runs through all shared variables, and  $\alpha$  runs through all states that can occur in a starvation cycle. (For example, Alice's states with respect to protocol (49) would be restricted to  $A3$  and  $A4$ , but Bob's are unrestricted.)

(d) With exercise 92 we can determine that the longest simple path, using only states that can occur in a starvation cycle for (49), is 15. And the clauses of (c) are unsatisfiable when  $r = 15$  and invariant (50) is used. Thus the only possible starvation cycle is made from two simple pure cycles; and those are easy to rule out.

**99.** Invariant assertions define the values of  $a$  and  $b$  at each state. Hence mutual exclusion follows as in exercise 95. For starvation-freedom, we can exclude states  $A0, A6, A7, A8$  from any cycle that starves Alice. But we need also to show that the state  $A5_t \wedge B0_t \wedge l_t$  is impossible; otherwise she could starve while Bob is maybe-ing. For that purpose we can add  $\neg((A6 \vee A7 \vee A8) \wedge (B6 \vee B7 \vee B8)) \wedge \neg(A8 \wedge \bar{l}) \wedge \neg(B8 \wedge l) \wedge \neg((A3 \vee A4 \vee A5) \wedge B0 \wedge l) \wedge \neg(A0 \wedge (B3 \vee B4 \vee B5) \wedge \bar{l})$  to the invariant  $\Phi(X)$ . The longest simple path through allowable states has length 42; and the clauses of exercise

sorting networks  
Kroening  
Strichman  
Eén  
Sörensson  
 $k$ -induction  
induction  
Sheeran  
Singh  
Stålmarck  
longest simple path



98(c) are unsatisfiable when  $r = 42$ . Notice that Alice and Bob never compete when setting the common variable  $l$ , because states A7 and B7 cannot occur together.

(See Dijkstra's *Cooperating Sequential Processes*, cited in the text.)

100. Bob is starved by the moves B1, (A1, A2, A3, B2, A4, B3, A0, B4, B1)<sup>∞</sup>. But an argument similar to the previous answer shows that Alice cannot be.

[The protocol obviously provides mutual exclusion as in exercise 95. It was devised independently in the late 1970s by J. E. Burns and L. Lamport, as a special case of an  $N$ -player protocol using only  $N$  shared bits; see *JACM* **33** (1986), 337–339.]

101. The following solution is based on G. L. Peterson's elegant protocol for  $N$  processes in *ACM Transactions on Programming Languages and Systems* **5** (1983), 56–65:

- |  |  |
|--|--|
| A0. Maybe go to A1.                      | B0. Maybe go to B1.                      |
| A1. Set $a_1 \leftarrow 1$ , go to A2.   | B1. Set $b_1 \leftarrow 1$ , go to B2.   |
| A2. If $b_2$ go to A2, else to A3.       | B2. If $a_1$ go to B2, else to B3.       |
| A3. Set $a_2 \leftarrow 1$ , go to A4.   | B3. Set $b_2 \leftarrow 1$ , go to B4.   |
| A4. Set $a_1 \leftarrow 0$ , go to A5.   | B4. Set $b_1 \leftarrow 0$ , go to B5.   |
| A5. If $b_1$ go to A5, else to A6.       | B5. If $a_2$ go to B5, else to B6.       |
| A6. Set $a_1 \leftarrow 1$ , go to A7.   | B6. Set $b_1 \leftarrow 1$ , go to B7.   |
| A7. If $b_1$ go to A8, else to A9.       | B7. If $a_1$ go to B8, else to B12.      |
| A8. If $b_2$ go to A7, else to A9.       | B8. If $a_2$ go to B9, else to B12.      |
| A9. Critical, go to A10.                 | B9. Set $b_1 \leftarrow 0$ , go to B10.  |
| A10. Set $a_1 \leftarrow 0$ , go to A11. | B10. If $a_1$ go to B11, else to B6.     |
| A11. Set $a_2 \leftarrow 0$ , go to A0.  | B11. If $a_2$ go to B10, else to B6.     |
|  | B12. Critical, go to B13.                |
|  | B13. Set $b_1 \leftarrow 0$ , go to B14. |
|  | B14. Set $b_2 \leftarrow 0$ , go to B0.  |

(Alice and Bob might need an app to help them deal with this.)

102. The clauses for, say, 'B5. If  $a$  go to B6, else to B7.' should be  $(@ \vee \overline{B5} \vee \bar{a} \vee \alpha_1 \vee \dots \vee \alpha_p \vee B6') \wedge (@ \vee \overline{B5} \vee a \vee \alpha_1 \vee \dots \vee \alpha_p \vee B7') \wedge (@ \vee \overline{B5} \vee B6' \vee B7')$ , where  $\alpha_1, \dots, \alpha_p$  are the states in which Alice sets  $a$ .

103. See, for example, any front cover of *SICOMP*, or of *SIAM Review* since 1970.

104. Assume that  $m \leq n$ . The case  $m = n$  is clearly impossible, because all four corners must be occupied. When  $m$  is odd and  $n = m + k + 1$ , put  $m$  bishops in the first and last columns, then  $k$  in the middle columns of the middle row. When  $m$  is even and  $n = m + 2k + 1$ , put  $m$  in the first and last columns, and two in the middle rows of columns  $m/2 + 2j$  for  $1 \leq j \leq k$ . There's no solution when  $m$  and  $n$  are both even, because the maximum number of independent bishops of each color is  $(m + n - 2)/2$ . [R. Berghammer, *LNCS* **6663** (2011), 103–106.]

105. (a) We must have  $(x_{ij}, x'_{ij}) = (1, 0)$  for  $t$  pairs  $ij$ , and  $(0, 1)$  for  $t$  other pairs; otherwise  $x_{ij} = x'_{ij}$ . Hence there are  $2^{m+n-2t}$  solutions.

(b) Use  $2mn$  variables  $y_{ij}, y'_{ij}$  for  $1 \leq i \leq m$  and  $1 \leq j \leq n$ , with binary clauses  $(\bar{y}_{ij} \vee \bar{y}'_{ij})$ , together with  $m + n + 2(m + n - 1)$  sets of cardinality constraints such as (20) and (21) to enforce the balance condition  $\sum \{y_{ij} + y'_{ij} \mid ij \in L\} = |L|$  for each row, column, and diagonal line  $L$ .

(c)  $T(m, n) = 1$  when  $\min(m, n) < 4$ , because only the zero matrix qualifies in such cases. Other values can be enumerated by backtracking, if they are small enough. (The asymptotic behavior is unknown.)

|             |         |      |       |         |          |
|-------------|---------|------|-------|---------|----------|
|             | $n = 4$ | $5$  | $6$   | $7$     | $8$      |
| $T(4, n) =$ | 3       | 7    | 17    | 35      | 77       |
| $T(5, n) =$ | 7       | 31   | 109   | 365     | 1367     |
| $T(6, n) =$ | 17      | 109  | 877   | 6315    | 47607    |
| $T(7, n) =$ | 35      | 365  | 6315  | 107637  | 1703883  |
| $T(8, n) =$ | 77      | 1367 | 47607 | 1703883 | 66291089 |

(d) Suppose  $m \leq n$ . Any solution with nonzero top row, bottom row, left column, and right column has all entries zero except

Dijkstra  
Burns  
Lamport  
Peterson  
SIAM  
Berghammer  
cardinality constraints  
backtracking

that  $y_{1t} = -y_{t1} = y_{(m+1-t)1} = -y_{mt} = y_{m(n+1-t)} = -y_{(m+1-t)n} = y_{tn} = -y_{1(n+1-t)}$ , for some  $t$  with  $1 < t \leq m/2$ . So the answer is  $2 \sum_{k=3}^m \lfloor (k-1)/2 \rfloor (m-k)(n-k)$ , which simplifies to  $q(q-1)(4q(n-q) - 5n + 2q + 3 + (m \bmod 2)(6n - 8q - 5))/3$  when  $q = \lfloor m/2 \rfloor$ .

Gerdes  
unit clauses

[The answer in the case  $(m, n) = (25, 30)$  is 36080; hence a random  $25 \times 30$  image will have an average of  $36080/256 \approx 140.9$  tomographically equivalent “neighbors” that differ from it in exactly eight pixel positions. Figure 36 has five such neighbors, one of which is shown in answer 111 below.]

(e) We can make all entries nonzero except on the main diagonals (see below). This is optimum, because the diagonal lines for  $a_1, a_3, \dots, a_{4n-1}, b_1, b_3, \dots, b_{4n-1}$  must each contain a different 0. So the answer is  $2n(n-1)$ . (But the maximum for odd sized boards is unknown; for  $n = (5, 7, 9)$  it turns out to be  $(6, 18, 33)$ .)

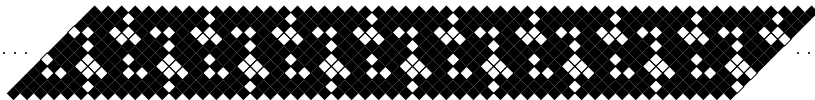
|          |            |         |         |
|----------|------------|---------|---------|
| 0++++--0 | 0+++0---0  |         |         |
| -0++--0+ | -----++0+  | 0+---00 | 0+---00 |
| --0+-0++ | 0-+-----+  | --+000  | --+000  |
| ---00+++ | +---+0---  | 0+--00+ | 0+-0-0+ |
| +++00--- | -+----0++  | -00+0-+ | -0-+00+ |
| ++0-+0-- | +0-0+0+--  | +0-0+0- | +000+-- |
| +0--++0- | ---+---++0 | +0+0-0- | +0+-00- |
| 0----++0 | +++++0---  | 0--0++0 | 0--0++0 |
|          | 0-0-----0  |         |         |

(f) The smallest counterexamples are  $7 \times 7$  (see above).

106. In an  $m \times n$  problem we must have  $0 \leq r_i \leq n, 0 \leq c_j \leq m$ , and  $0 \leq a_d, b_d \leq \min\{d, m, n, m+n-d\}$ . So the total number  $B$  of possibilities, assuming that  $m \leq n$ , is  $(n+1)^m (m+1)^n ((m+1)! (m+1)^{n-m} m!)^2$ , which is  $\approx 3 \cdot 10^{197}$  when  $(m, n) = (25, 30)$ . Since  $2^{750}/B \approx 2 \cdot 10^{28}$ , we conclude that a “random”  $25 \times 30$  digital tomography problem usually has more than  $10^{28}$  solutions. (Of course there are other constraints too; for example, the fact that  $\sum r_i = \sum c_j = \sum a_d = \sum b_d$  reduces  $B$  by at least a factor of  $(n+1)(m+1)^2$ .)

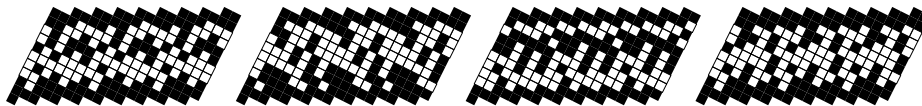
107. (a)  $(r_1, \dots, r_6) = (11, 11, 11, 9, 9, 10)$ ;  $(c_1, \dots, c_{13}) = (6, 5, 6, 2, 4, 4, 6, 5, 4, 2, 6, 5, 6)$ ;  $(a_1, \dots, a_6) = (11, 10, 9, 9, 11, 11)$ ;  $(b_1, \dots, b_{12}) = (6, 1, 6, 5, 7, 5, 6, 2, 6, 5, 7, 5)$ .

(b) There are two others, namely the following one and its left-right reversal:



[Reference: P. Gerdes, *Sipatsi* (Maputo: U. Pedagógica, 2009), page 62, pattern #122.]

108. Here are four of the many possibilities:



109. **F1.** [Initialize.] Find one solution  $y_1 \dots y_n$ , or terminate if the problem is unsatisfiable. Then set  $y_{n+1} \leftarrow 1$  and  $d \leftarrow 0$ .

**F2.** [Advance  $d$ .] Set  $d$  to the smallest  $j > d$  such that  $y_j = 1$ .

**F3.** [Done?] If  $d > n$ , terminate with  $y_1 \dots y_n$  as the answer.

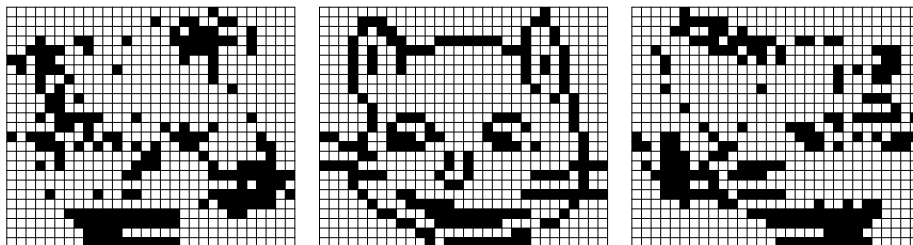
**F4.** [Try for smaller.] Try to find a solution with additional unit clauses to force  $x_j = y_j$  for  $1 \leq j < d$  and  $x_d = 0$ . If successful, set  $y_1 \dots y_n \leftarrow x_1 \dots x_n$ . Return to F2. ■

Even better is to incorporate a similar procedure into the solver itself; see exercise 275.

110. Algorithm B actually gives these directly:

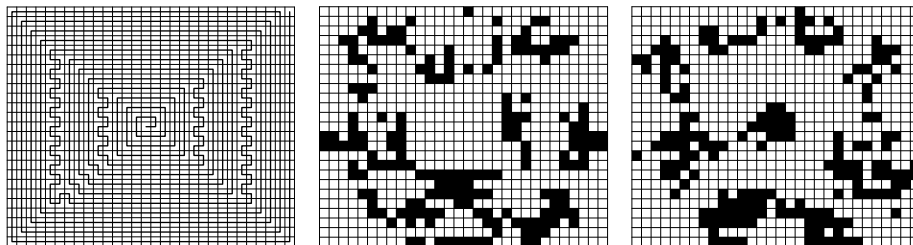
```
0011111110111011111001011111011110111111101110110111111110010111101111101111110011101111101111110111
1111111101111111001100111110011110111111101011111110111101111110011001111011011110111111
```

111. This family of problems appears to provide an excellent (though sometimes formidable) series of benchmark tests for SAT solvers. The suggested example has solutions



(a) colexicographically first; (b) minimally different; (c) colexicographically last;

and several of the entries in (a) were by no means easy. An even more difficult case arises if we base lexicographic order on a rook path that spirals out from the center (thus favoring solutions that are mostly 0 or mostly 1 in the middle):



(a) spiral rook path; (b) “spirographically” first; (c) “spirographically” last.

Here many of the entries have never yet been solved by a SAT solver, as of 2013, although again IP solvers have no great difficulty. In fact, the “lexicographic pure cutting plane” procedure of E. Balas, M. Fischetti, and A. Zanette [*Math. Programming* **A130** (2011), 153–176; **A135** (2012), 509–514] turns out to be particularly effective on such problems.

112. Reasonably tight upper and lower bounds would also be interesting.

113. Given an  $N \times N \times N$  contingency problem with binary constraints  $C_{JK} = X_{*JK}$ ,  $R_{IK} = X_{I*K}$ ,  $P_{IJ} = X_{IJ*}$ , we can construct an equivalent  $n \times n$  digital tomography problem with  $n = N^2 + N^3 + N^4$  as follows: First construct a four-dimensional tensor  $Y_{IJKL} = X_{(I \oplus L)JK}$ , where  $I \oplus L = 1 + (I + L - 1) \bmod N$ , and notice that  $Y_{*JKL} = Y_{IJK*} = X_{*JK}$ ,  $Y_{I*KL} = X_{(I \oplus L)*K}$ ,  $Y_{IJ*L} = X_{(I \oplus L)J*}$ . Then define  $x_{ij}$  for  $1 \leq i, j \leq n$  by the rule  $x_{ij} = Y_{IJKL}$  when  $i = I - N^2K + N^3L$ ,  $j = NJ + N^2K + N^3L$ , otherwise  $x_{ij} = 0$ . This rule makes sense; for if  $1 \leq I, I', J, J', K, K', L, L' \leq N$  and  $I - N^2K + N^3L = I' - N^2K' + N^3L'$  and  $NJ + N^2K + N^3L = NJ' + N^2K' + N^3L'$ , we have  $I \equiv I'$  (modulo  $N$ ); hence  $I = I'$  and  $K \equiv K'$ ; hence  $K = K'$ ,  $L = L'$ ,  $J = J'$ .

Under this correspondence the marginal sums are  $r_i = Y_{I*KL}$  when  $i = I - N^2K + N^3L$ ,  $c_j = Y_{*JKL}$  when  $j = NJ + N^2K + N^3L$ ,  $a_d = Y_{IJ*L}$  when  $d+1 = I + NJ + 2N^3L$ ,  $b_d = Y_{IJK*}$  when  $d - n = I - NJ - 2N^2K$ , otherwise zero. [See S. Brunetti, A. Del Lungo, P. Gritzmann, and S. de Vries, *Theoretical Comp. Sci.* **406** (2008), 63–71.]

benchmark tests  
rook path  
spiral  
cutting plane  
Balas  
Fischetti  
Zanette  
Brunetti  
Del Lungo  
Gritzmann  
de Vries

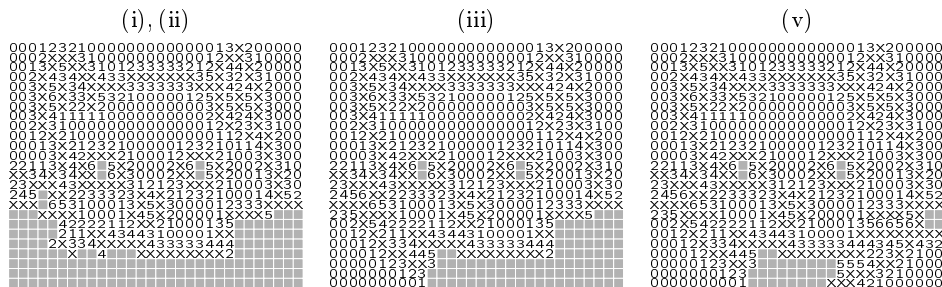
**114.** (a) From  $x_{7,23} + x_{7,24} = x_{7,23} + x_{7,24} + x_{7,25} = x_{7,24} + x_{7,25} = 1$  we deduce  $x_{7,23} = x_{7,25} = 0$  and  $x_{7,24} = 1$ , revealing  $n_{7,23} = n_{7,25} = 5$ . Now  $x_{6,23} + x_{6,24} = x_{6,24} + x_{6,25} = x_{4,24} + x_{5,24} + x_{6,24} + x_{6,25} = 1$ ; hence  $x_{4,24} = x_{5,24} = 0$ , revealing  $n_{4,24} = n_{5,24} = 2$ . So  $x_{6,23} = x_{6,25} = 0$ , and the rest is easy.  
 (b) Let  $y_{i,j}$  mean “cell  $(i, j)$  has been probed safely, revealing  $n_{i,j}$ .” Consider the clauses  $C$  by appending  $\bar{y}_{i,j}$  to each clause of the symmetric function  $[\sum_{i'=i-1}^{i+1} \sum_{j'=j-1}^{j+1} x_{i',j'} = n_{i,j}]$ , for all  $i, j$  with  $x_{i,j} = 0$ . Also include  $(\bar{x}_{i,j} \vee \bar{y}_{i,j})$ , as well as clauses for the symmetric function  $S_N(x)$  if we're told the total number  $N$  of mines.

symmetric function  
 polarities  
 smile  
 NP-complete  
 coNP-complete  
 Kaye  
 Scott  
 Stege  
 van Rooij  
 phoenix

Given any subset  $F$  of mine-free cells, the clauses  $C_F = C \wedge \bigwedge \{y_{i,j} \mid (i, j) \in F\}$  are satisfiable precisely by the configurations of mines that are consistent with the data  $\{n_{i,j} \mid (i, j) \in F\}$ . Therefore cell  $(i, j)$  is safe if and only if  $C_F \wedge x_{i,j}$  is unsatisfiable.

A simple modification of Algorithm C can be used to “grow”  $F$  until no further safe cells can be added: Given a solution to  $C_F$  for which neither  $x_{i,j}$  nor  $\bar{x}_{i,j}$  was obtained at root level (level 0), we can try to find a “flipped” solution by using the complemented value as the decision at level 1. Such a solution will be found if and only if the flipped value is consistent; otherwise the unflipped value will have been forced at level 0. By changing default polarities we can favor solutions that flip many variables at once. Whenever a literal  $\bar{x}_{i,j}$  is newly deduced at root level, we can force  $y_{i,j}$  to be true, thus adding  $(i, j)$  to  $F$ . We reach an impasse when a set of solutions has been obtained for  $C_F$  that covers both settings of every unforced  $x_{i,j}$ .

For problem (i) we start with  $F = \{(1, 1)\}$ , etc. Case (iv) by itself uncovers only 56 cells in the lower right corner. The other results, each obtained in  $< 6 G\mu$ , are:

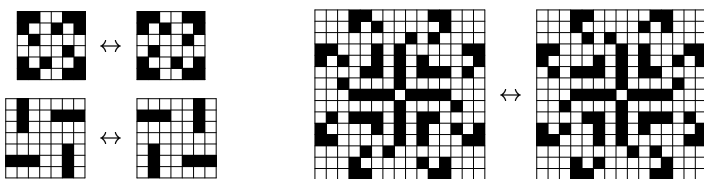


Notice that the Cheshire cat's famous smile defies logic and requires much guesswork!

[For aspects of Minesweeper that are NP-complete and coNP-complete, see Kaye, Scott, Stege, and van Rooij, *Math. Intelligencer* **22**, 2 (2000), 9–15; **33**, 4 (2011), 5–17.]

**115.** Several thousand runs of the algorithm in the previous exercise, given that the total number of mines is 10, indicate success probabilities  $.490 \pm .007$ ,  $.414 \pm .004$ ,  $.279 \pm .003$ , when the first guess is respectively in a corner, in the center of an edge, or in the center.

**116.** The smallest is the “clock” in answer 69(b). Other noteworthy possibilities are



as well as the “phoenix” in Fig. A-7.

**117.** (a) Set  $x_0 = x_{n+1} = 0$ , and let  $(a, b, c)$  be respectively the number of occurrences of  $(01, 10, 11)$  as a substring of  $x_0 x_1 \dots x_{n+1}$ . Then  $a + c = b + c = \nu x$  and  $c = \nu^{(2)} x$ ; hence  $a = b = \nu x - \nu^{(2)} x$  is the number of runs.

partial backtracking  
links dance  
Pure literals

(b) In this case the complete binary tree will have only  $n-1$  leaves, corresponding to  $\{x_1 x_2, \dots, x_{n-1} x_n\}$ ; therefore we want to replace  $n$  by  $n-1$  in (20) and (21).

The clauses of (20) remain unchanged unless  $t_k \leq 3$ . When  $t_k = 2$  they become  $(\bar{x}_{2k-n+1} \vee \bar{x}_{2k-n+2} \vee b_1^k) \wedge (\bar{x}_{2k-n+2} \vee \bar{x}_{2k-n+3} \vee b_1^k) \wedge (\bar{x}_{2k-n+1} \vee \bar{x}_{2k-n+2} \vee \bar{x}_{2k-n+3} \vee b_2^k)$ . When  $t_k = 3$  we have  $2k = n-1$ , and they become  $(\bar{b}_1^{2k} \vee b_1^k) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee b_1^k) \wedge (\bar{b}_2^{2k} \vee b_2^k) \wedge (\bar{b}_1^{2k} \vee \bar{x}_1 \vee \bar{x}_2 \vee b_2^k) \wedge (\bar{b}_2^{2k} \vee \bar{x}_1 \vee \bar{x}_2 \vee b_3^k)$ .

The clauses of (21) remain unchanged except in simple cases when  $n \leq 3$ .

(c) Now the leaves represent  $\bar{x}_i \bar{x}_{i+1} = \bar{x}_i \vee \bar{x}_{i+1}$ . So we change (20), when  $t_k = 2$ , to  $(x_{2k-n+1} \vee b_1^k) \wedge (x_{2k-n+2} \vee b_1^k) \wedge (x_{2k-n+3} \vee b_1^k) \wedge (x_{2k-n+2} \vee b_2^k) \wedge (x_{2k-n+1} \vee x_{2k-n+3} \vee b_2^k)$ . And there are eight clauses when  $t_k = 3$ :  $(\bar{b}_1^{2k} \vee b_1^k) \wedge (x_1 \vee b_1^k) \wedge (x_2 \vee b_1^k) \wedge (\bar{b}_2^{2k} \vee b_2^k) \wedge (\bar{b}_1^{2k} \vee x_1 \vee b_2^k) \wedge (\bar{b}_1^{2k} \vee x_2 \vee b_2^k) \wedge (\bar{b}_2^{2k} \vee x_1 \vee b_3^k) \wedge (\bar{b}_2^{2k} \vee x_2 \vee b_3^k)$ .

**118.** Let  $p_{i,j} = [\text{the pixel in row } i \text{ and column } j \text{ should be covered}]$ , and introduce variables  $h_{i,j}$  when  $p_{i,j} = p_{i,j+1} = 1$ ,  $v_{i,j}$  when  $p_{i,j} = p_{i+1,j} = 1$ . The clauses are (i)  $(h_{i,j} \vee h_{i,j-1} \vee v_{i,j} \vee v_{i-1,j})$ , whenever  $p_{i,j} = 1$ , omitting variables that don't exist; (ii)  $(\bar{h}_{i,j} \vee \bar{h}_{i,j-1})$ ,  $(\bar{h}_{i,j} \vee \bar{v}_{i,j})$ ,  $(\bar{h}_{i,j} \vee \bar{v}_{i-1,j})$ ,  $(\bar{h}_{i,j-1} \vee \bar{v}_{i,j})$ ,  $(\bar{h}_{i,j-1} \vee \bar{v}_{i-1,j})$ ,  $(\bar{v}_{i,j} \vee \bar{v}_{i-1,j})$ , whenever  $p_{i,j} = 1$ , omitting clauses whose variables don't both exist; and (iii)  $(h_{i,j} \vee h_{i+1,j} \vee v_{i,j} \vee v_{i,j+1})$ , whenever  $p_{i,j} + p_{i,j+1} + p_{i+1,j} + p_{i+1,j+1} \geq 3$ , omitting variables that don't exist. (The example has 10527 clauses in 2874 variables, but it's quickly solved.)

**119.** There's symmetry between  $l$  and  $\bar{l}$ , also between  $l$  and  $10-l$ ; so we need consider only  $l = (1, 2, 3, 4, 5)$ , with respectively  $(4, 4, 6, 6, 8)$  occurrences. The smallest result is  $F|5 = \{123, 234, 678, 789, 246, 468, 147, 369, \bar{1}\bar{2}\bar{3}, \bar{2}\bar{3}\bar{4}, \bar{3}\bar{4}, \bar{4}\bar{6}, \bar{6}\bar{7}, \bar{6}\bar{7}\bar{8}, \bar{7}\bar{8}\bar{9}, \bar{1}\bar{3}, \bar{2}\bar{4}\bar{6}, \bar{3}\bar{7}, \bar{4}\bar{6}\bar{8}, \bar{7}\bar{9}, \bar{1}\bar{4}\bar{7}, \bar{2}\bar{8}, \bar{3}\bar{6}\bar{9}, \bar{1}\bar{9}\}$ .

**120.** True.

**121.** The main point of interest is that an empty clause is typically discovered in the *midst* of step A3; partial backtracking must be done when taking back the changes that were made before this interruption.

**A3.** [Remove  $\bar{l}$ .] Set  $p \leftarrow F(\bar{l})$  (which is  $F(l \oplus 1)$ , see (57)). While  $p \geq 2n+2$ , set  $j \leftarrow C(p)$ ,  $i \leftarrow \text{SIZE}(j)$ , and if  $i > 1$  set  $\text{SIZE}(j) \leftarrow i-1$ ,  $p \leftarrow F(p)$ . But if  $i = 1$ , interrupt that loop and set  $p \leftarrow B(p)$ ; then while  $p \geq 2n+2$ , set  $j \leftarrow C(p)$ ,  $i \leftarrow \text{SIZE}(j)$ ,  $\text{SIZE}(j) \leftarrow i+1$ ,  $p \leftarrow B(p)$ ; and finally go to A5.

**A4.** [Deactivate  $l$ 's clauses.] Set  $p \leftarrow F(l)$ . While  $p \geq 2n+2$ , set  $j \leftarrow C(p)$ ,  $i \leftarrow \text{START}(j)$ ,  $p \leftarrow F(p)$ , and for  $i \leq s < i + \text{SIZE}(j) - 1$  set  $q \leftarrow F(s)$ ,  $r \leftarrow B(s)$ ,  $B(q) \leftarrow r$ ,  $F(r) \leftarrow q$ , and  $C(L(s)) \leftarrow C(L(s)) - 1$ . Then set  $a \leftarrow a - C(l)$ ,  $d \leftarrow d + 1$ , and return to A2.

**A7.** [Reactivate  $l$ 's clauses.] Set  $a \leftarrow a + C(l)$  and  $p \leftarrow B(l)$ . While  $p \geq 2n+2$ , set  $j \leftarrow C(p)$ ,  $i \leftarrow \text{START}(j)$ ,  $p \leftarrow B(p)$ , and for  $i \leq s < i + \text{SIZE}(j) - 1$  set  $q \leftarrow F(s)$ ,  $r \leftarrow B(s)$ ,  $B(q) \leftarrow F(r) \leftarrow s$ , and  $C(L(s)) \leftarrow C(L(s)) + 1$ . (The links dance a little here.)

**A8.** [Unremove  $\bar{l}$ .] Set  $p \leftarrow F(\bar{l})$ . While  $p \geq 2n+2$ , set  $j \leftarrow C(p)$ ,  $i \leftarrow \text{SIZE}(j)$ ,  $\text{SIZE}(j) \leftarrow i+1$ ,  $p \leftarrow F(p)$ . Then go to A5. ■

**122.** Pure literals are problematic when we want all solutions, so we don't take advantage of them here. Indeed, things get simpler; only the move codes 1 and 2 are needed.

**A1\*.** [Initialize.] Set  $d \leftarrow 1$ .

- A2\***. [Visit or choose.] If  $d > n$ , visit the solution defined by  $m_1 \dots m_n$  and go to A6\*. Otherwise set  $l \leftarrow 2d + 1$  and  $m_d \leftarrow 1$ .
- A3\***. [Remove  $\bar{l}$ .] Delete  $\bar{l}$  from all active clauses; but go to A5\* if that would make a clause empty.
- A4\***. [Deactivate  $l$ 's clauses.] Suppress all clauses that contain  $l$ . Then set  $d \leftarrow d + 1$  and return to A2\*.
- A5\***. [Try again.] If  $m_d = 1$ , set  $m_d \leftarrow 2$ ,  $l \leftarrow 2d$ , and go to A3\*.
- A6\***. [Backtrack.] Terminate if  $d = 1$ . Otherwise set  $d \leftarrow d - 1$  and  $l \leftarrow 2d + (m_d \& 1)$ .
- A7\***. [Reactivate  $l$ 's clauses.] Unsuppress all clauses that contain  $l$ .
- A8\***. [Unremove  $\bar{l}$ .] Reinstate  $\bar{l}$  in all the active clauses that contain it. Then go back to A5\*. ■

It's no longer necessary to update the values  $\mathbf{C}(k)$  for  $k < 2n + 2$  in steps A4\* and A7\*.

**123.** For example, we might have

$$\begin{array}{l} p = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16 \ 17 \ 18 \ 19 \ 20 \\ L(p) = 3 \ 9 \ 7 \ 8 \ 7 \ 5 \ 6 \ 5 \ 3 \ 4 \ 3 \ 8 \ 2 \ 8 \ 6 \ 9 \ 6 \ 4 \ 7 \ 4 \ 2 \end{array}$$

and  $\text{START}(j) = 21 - 3j$  for  $0 \leq j \leq 7$ ;  $W_2 = 3$ ,  $W_3 = 7$ ,  $W_4 = 4$ ,  $W_5 = 0$ ,  $W_6 = 5$ ,  $W_7 = 1$ ,  $W_8 = 6$ ,  $W_9 = 2$ . Also  $\text{LINK}(j) = 0$  for  $1 \leq j \leq 7$  in this case.

**124.** Set  $j \leftarrow W_{\bar{l}}$ . While  $j \neq 0$ , a literal other than  $\bar{l}$  should be watched in clause  $j$ , so we do the following: Set  $i \leftarrow \text{START}(j)$ ,  $i' \leftarrow \text{START}(j - 1)$ ,  $j' \leftarrow \text{LINK}(j)$ ,  $k \leftarrow i + 1$ . While  $k < i'$ , set  $l' \leftarrow L(k)$ ; if  $l'$  isn't false (that is, if  $|l'| > d$  or  $l' + m_{|l'|}$  is even, see (57)), set  $L(i) \leftarrow l'$ ,  $L(k) \leftarrow \bar{l}$ ,  $\text{LINK}(j) \leftarrow W_{l'}$ ,  $W_{l'} \leftarrow j$ ,  $j \leftarrow j'$ , and exit the loop on  $k$ ; otherwise set  $k \leftarrow k + 1$  and continue that loop. If  $k$  reaches  $i'$ , however, we cannot stop watching  $\bar{l}$ ; so we set  $W_{\bar{l}} \leftarrow j$ , exit the loop on  $j$ , and go on to step B5.

**125.** Change steps B2 and B4 to be like A2\* and A4\* in answer 122.

**126.** Starting with active ring (6978), the unit clause 9 will be found (because 9 appears before 8); the clause  $\bar{9}\bar{3}\bar{6}$  will become  $\bar{6}\bar{3}\bar{9}$ ; the active ring will become (786).

**127.** Before: 11414545; after: 1142. (And then 11425, etc.)

| 128. Active ring | $x_1 x_2 x_3 x_4$ | Units        | Choice    | Changed clauses                          |
|------------------|-------------------|--------------|-----------|--|
| (1234)           | - - - -           |              | $\bar{1}$ | 413                                      |
| (234)            | 0 - - -           |              | $\bar{2}$ | $\bar{1}24$                              |
| (34)             | 0 0 - -           | $\bar{3}$    | $\bar{3}$ |  |
| (4)              | 0 0 0 -           | $4, \bar{4}$ | Backtrack |  |
| (34)             | 0 - - -           |              | 2         | $3\bar{2}\bar{1}, \bar{4}\bar{2}\bar{1}$ |
| (34)             | 0 1 - -           | $\bar{4}$    | $\bar{4}$ | $314, \bar{3}42$                         |
| (3)              | 0 1 - 0           | $3, \bar{3}$ | Backtrack |  |
| (43)             | - - - -           |              | 1         | $2\bar{1}4, \bar{4}\bar{1}\bar{3}$       |
| (243)            | 1 - - -           |              | 2         |  |
| (43)             | 1 1 - -           | 3            | 3         | $4\bar{3}\bar{2}, \bar{2}\bar{3}\bar{1}$ |
| (4)              | 1 1 1 -           | $4, \bar{4}$ | Backtrack |  |
| (34)             | 1 - - -           |              | $\bar{2}$ | $\bar{3}21, 4\bar{1}2$                   |
| (34)             | 1 0 - -           | 4            | 4         | $\bar{3}\bar{1}4, 1\bar{2}4, 3\bar{4}2$  |
| (3)              | 1 0 - 1           | $3, \bar{3}$ | Backtrack |  |

**129.** Set  $j \leftarrow W_l$ , then do the following steps while  $j \neq 0$ : (i) Set  $p \leftarrow \text{START}(j) + 1$ ; (ii) if  $p = \text{START}(j - 1)$ , return 1; (iii) if  $L(p)$  is false (that is, if  $x_{|L(p)|} = L(p) \ \& \ 1$ ), set  $p \leftarrow p + 1$  and repeat (ii); (iv) set  $j \leftarrow \text{LINK}(j)$ . If  $j$  becomes zero, return 0.

**130.** Set  $l \leftarrow 2k + b$ ,  $j \leftarrow W_l$ ,  $W_l \leftarrow 0$ , and do the following steps while  $j \neq 0$ : (i) Set  $j' \leftarrow \text{LINK}(j)$ ,  $i \leftarrow \text{START}(j)$ ,  $p \leftarrow i + 1$ ; (ii) while  $L(p)$  is false, set  $p \leftarrow p + 1$  (see answer 129; this loop will end before  $p = \text{START}(j - 1)$ ); (iii) set  $l' \leftarrow L(p)$ ,  $L(p) \leftarrow l$ ,  $L(i) \leftarrow l'$ ; (iv) set  $p \leftarrow W_{l'}$  and  $q \leftarrow W_{\bar{l}'}$ , and go to (vi) if  $p \neq 0$  or  $q \neq 0$  or  $x_{|l'|} \geq 0$ ; (v) if  $t = 0$ , set  $t \leftarrow h \leftarrow |l'|$  and  $\text{NEXT}(t) \leftarrow h$ , otherwise set  $\text{NEXT}(|l'|) \leftarrow h$ ,  $h \leftarrow |l'|$ ,  $\text{NEXT}(t) \leftarrow h$  (thus inserting  $|l'| = l' \gg 1$  into the ring as its new head); (vi) set  $\text{LINK}(j) \leftarrow p$ ,  $W_{l'} \leftarrow j$  (thus inserting  $j$  into the watch list of  $l'$ ); (vii) set  $j \leftarrow j'$ .

[The tricky part here is to remember that  $t$  can be zero in step (v).]

**131.** For example, the author tried selecting a variable  $x_k$  for which  $s_{2k} \cdot s_{2k+1}$  is maximum, where  $s_l$  is the length of  $l$ 's watch list plus  $\varepsilon$ , and the parameter  $\varepsilon$  was 0.1. This reduced the runtime for *waerden*(3, 10; 97) to 139.8 gigamems, with 8.6 meganodes. Less dramatic effects occurred with *langford*(13): 56.2 gigamems, with 10.8 meganodes, versus 99.0 gigamems if the *minimum*  $s_{2k} \cdot s_{2k+1}$  was chosen instead.

**132.** The unsatisfiable clauses  $(\bar{x}_1 \vee x_2)$ ,  $(x_1 \vee \bar{x}_2)$ ,  $(\bar{x}_3 \vee x_4)$ ,  $(x_3 \vee \bar{x}_4)$ ,  $\dots$ ,  $(\bar{x}_{2n-1} \vee x_{2n})$ ,  $(x_{2n-1} \vee \bar{x}_{2n})$ ,  $(\bar{x}_{2n-1} \vee \bar{x}_{2n})$ ,  $(x_{2n-1} \vee x_{2n})$  cause it to investigate all  $2^n$  settings of  $x_1, x_3, \dots, x_{2n-1}$  before encountering a contradiction and repeatedly backtracking.

(Incidentally, the successive move codes make a pretty pattern. If the stated clauses are ordered randomly, the algorithm runs significantly faster, but it still apparently needs nonpolynomial time. What is the growth rate?)

**133.** (a) Optimum backtrack trees for  $n$ -variable SAT problems can be calculated with  $\Theta(n3^n)$  time and  $\Theta(3^n)$  space by considering all  $3^n$  partial assignments, “bottom up.” In this 9-variable problem we obtain a tree with 67 nodes (the minimum) if we branch first on  $x_3$  and  $x_5$ , then on  $x_6$  if  $x_3 \neq x_5$ ; unit clauses arise at all other nodes.

(b) Similarly, the worst tree turns out to have 471 nodes. But if we require the algorithm to branch on a unit clause whenever possible, the worst size is 187. (Branch first on  $x_1$ , then  $x_4$ , then  $x_7$ ; avoid opportunities for unit clauses.)

**134.** Let each BIMP list be accessed by ADDR, BSIZE, CAP, and K fields, where ADDR is the starting address in MEM of a block that's able to store CAP items, and  $\text{CAP} = 2^k$ ; ADDR is a multiple of CAP, and BSIZE is the number of items currently in use. Initially  $\text{CAP} = 4$ ,  $K = 2$ ,  $\text{BSIZE} = 0$ , and ADDR is a convenient multiple of 4. The  $2n$  BIMP tables therefore occupy  $8n$  slots initially. If MEM has room for  $2^M$  items, those tables can be allocated so that the doubly linked lists AVAIL[k] initially contain  $a_k = (0 \text{ or } 1)$  available blocks of size  $2^k$  for each  $k$ , where  $2^M - 8n = (a_{M-1} \dots a_1 a_0)_2$ .

Resizing is necessary when  $\text{BSIZE} = \text{CAP}$  and we need to increase BSIZE. Set  $a \leftarrow \text{ADDR}$ ,  $k \leftarrow K$ ,  $\text{CAP} \leftarrow 2^{k+1}$ , and let  $b \leftarrow a \oplus 2^k$  be the address of  $a$ 's buddy. If  $b$  is a free block of size  $2^k$ , we're in luck: We remove  $b$  from AVAIL[k]; then if  $a \ \& \ 2^k = 0$ , nothing needs to be done, otherwise we copy BSIZE items from  $a$  to  $b$  and set  $\text{ADDR} \leftarrow b$ .

In the unlucky case when  $b$  is either reserved or free of size  $< 2^k$ , we set  $p$  to the address of the first block in AVAIL[k'], where AVAIL[t] is empty for  $k < t < k'$  (or we panic if MEM's capacity is exceeded). After removing  $p$  from AVAIL[k'], we split off new free blocks of sizes  $2^{k+1}, \dots, 2^{k'-1}$  if  $k' > k + 1$ . Finally we copy BSIZE items from block  $a$  to block  $p$ , set  $\text{ADDR} \leftarrow p$ , and put  $a$  into AVAIL[k]. (We needn't try to “collapse”  $a$  with its buddy, since the buddy isn't free.)

**135.** They're the complements of the literals in BIMP( $\bar{I}$ ).

empty list  
Knuth  
move codes  
randomly  
asymptotics  
unit clause  
Resizing

**136.** Before,  $\{(1, 2), (4, 2), (4, 5), (5, 1), (5, 7), (6, 9)\}$ ; after,  $\{(1, 2), (4, 2), (6, 9)\}$ .

**137.** If  $p$  in a TIMP list points to the pair  $(u, v)$ , let's write  $u = U(p)$  and  $v = V(p)$ .

(a) Set  $N \leftarrow n - G$ ,  $x \leftarrow \text{VAR}[N]$ ,  $j \leftarrow \text{INX}[X]$ ,  $\text{VAR}[j] \leftarrow x$ ,  $\text{INX}[x] \leftarrow j$ ,  $\text{VAR}[N] \leftarrow X$ ,  $\text{INX}[X] \leftarrow N$ . Then do the following for  $l = 2X$  and  $l = 2X + 1$ , and for all  $p$  in  $\text{TIMP}(l)$ :  $u \leftarrow U(p)$ ,  $v \leftarrow V(p)$ ,  $p' \leftarrow \text{LINK}(p)$ ,  $p'' \leftarrow \text{LINK}(p')$ ;  $s \leftarrow \text{TSIZE}(\bar{u}) - 1$ ,  $\text{TSIZE}(\bar{u}) \leftarrow s$ ,  $t \leftarrow$  pair  $s$  of  $\text{TIMP}(\bar{u})$ ; if  $p' \neq t$ , swap pairs by setting  $u' \leftarrow U(t)$ ,  $v' \leftarrow V(t)$ ,  $q \leftarrow \text{LINK}(t)$ ,  $q' \leftarrow \text{LINK}(q)$ ,  $\text{LINK}(q') \leftarrow p'$ ,  $\text{LINK}(p) \leftarrow t$ ,  $U(p') \leftarrow u'$ ,  $V(p') \leftarrow v'$ ,  $\text{LINK}(p') \leftarrow q$ ,  $U(t) \leftarrow v$ ,  $V(t) \leftarrow \bar{l}$ ,  $\text{LINK}(t) \leftarrow p''$ . Then set  $s \leftarrow \text{TSIZE}(\bar{v}) - 1$ ,  $\text{TSIZE}(\bar{v}) \leftarrow s$ ,  $t \leftarrow$  pair  $s$  of  $\text{TIMP}(\bar{v})$ ; if  $p'' \neq t$ , swap pairs by setting  $u' \leftarrow U(t)$ ,  $v' \leftarrow V(t)$ ,  $q \leftarrow \text{LINK}(t)$ ,  $q' \leftarrow \text{LINK}(q)$ ,  $\text{LINK}(q') \leftarrow p''$ ,  $\text{LINK}(p') \leftarrow t$ ,  $U(p'') \leftarrow u'$ ,  $V(p'') \leftarrow v'$ ,  $\text{LINK}(p'') \leftarrow q$ ,  $U(t) \leftarrow \bar{l}$ ,  $V(t) \leftarrow u$ ,  $\text{LINK}(t) \leftarrow p$ .

Notice that we do *not* make the current pairs of  $\text{TIMP}(l)$  inactive. They won't be accessed by the algorithm until it needs to undo the swaps just made.

(b) In  $\text{VAR}$  and in each  $\text{TIMP}$  list, the active entries appear first. The inactive entries follow, in the same order as they were swapped out, because inactive entries never participate in swaps. Therefore we can reactivate the most-recently-swapped-out entry by simply increasing the count of active entries. We must, however, be careful to do this "virtual unswapping" in precisely the reverse order from which we did the swapping.

Thus, for  $l = 2X + 1$  and  $l = 2X$ , and for all  $p$  in  $\text{TIMP}(l)$ , proceeding in the reverse order from (a), we set  $u \leftarrow U(p)$ ,  $v \leftarrow V(p)$ ,  $\text{TSIZE}(\bar{v}) \leftarrow \text{TSIZE}(\bar{v}) + 1$ , and  $\text{TSIZE}(\bar{u}) \leftarrow \text{TSIZE}(\bar{u}) + 1$ .

(The number  $N$  of free variables increases implicitly, because  $N + E = n$  in step L12. Thus nothing needs to be done to  $\text{VAR}$  or  $\text{INX}$ .)

**138.** Because  $\bar{v} \in \text{BIMP}(\bar{u})$ , (62) will be used to make  $u$  nearly true. That loop will *also* make  $v$  nearly true, because  $v \in \text{BIMP}(u)$  is equivalent to  $\bar{u} \in \text{BIMP}(\bar{v})$ .

**139.** Introduce a new variable  $\text{BSTAMP}$  analogous to  $\text{ISTAMP}$ , and a new field  $\text{BST}(l)$  analogous to  $\text{IST}(l)$  in the data for each literal  $l$ . At the beginning of step L9, set  $\text{BSTAMP} \leftarrow \text{BSTAMP} + 1$ , then set  $\text{BST}(l) \leftarrow \text{BSTAMP}$  for  $l = \bar{u}$  and all  $l \in \text{BIMP}(\bar{u})$ . Now, if  $\text{BST}(\bar{v}) \neq \text{BSTAMP}$  and  $\text{BST}(v) \neq \text{BSTAMP}$ , do the following for all  $w \in \text{BIMP}(v)$ : If  $w$  is fixed in context  $\text{NT}$  (it must be fixed true, since  $\bar{w}$  implies  $\bar{v}$ ), do nothing. Otherwise if  $\text{BST}(\bar{w}) = \text{BSTAMP}$ , perform (62) with  $l \leftarrow u$  and exit the loop on  $w$  (because  $\bar{u}$  implies both  $w$  and  $\bar{w}$ ). Otherwise, if  $\text{BST}(w) \neq \text{BSTAMP}$ , append  $w$  to  $\text{BIMP}(\bar{u})$  and  $u$  to  $\text{BIMP}(\bar{w})$ . (Of course (63) must be invoked when needed.)

Then increase  $\text{BSTAMP}$  again, and do the same thing with  $u$  and  $v$  reversed.

**140.** Unfortunately, no: We might have  $\Omega(n)$  changes to  $\text{BSIZE}$  on each of  $\Omega(n)$  levels of the search tree. However, the  $\text{ISTACK}$  will never have more entries than the total number of cells in all  $\text{BIMP}$  tables (namely  $2^M$  in answer 134).

**141.** Suppose  $\text{ISTAMP} \leftarrow (\text{ISTAMP} + 1) \bmod 2^e$  in step L5. If  $\text{ISTAMP} = 0$  after that operation, we can safely set  $\text{ISTAMP} \leftarrow 1$  and  $\text{IST}(l) \leftarrow 0$  for  $2 \leq l \leq 2n + 1$ . (A similar remark applies to  $\text{BSTAMP}$  and  $\text{BST}(l)$  in answer 139.)

**142.** (The following operations, performed after  $\text{BRANCH}[d]$  is set in step L2, will also output ' | ' to mark levels of the search where no decision was made.) Set  $\text{BACKL}[d] \leftarrow F$ ,  $r \leftarrow k \leftarrow 0$ , and do the following while  $k < d$ : While  $r < \text{BACKF}[k]$ , output '6 + ( $R_r$  & 1) ' and set  $r \leftarrow r + 1$ . If  $\text{BRANCH}[k] < 0$ , output ' | ' ; otherwise output '2BRANCH[k] + ( $R_r$  & 1) ' and set  $r \leftarrow r + 1$ . While  $r < \text{BACKL}[k + 1]$ , output '4 + ( $R_r$  & 1) ' and set  $r \leftarrow r + 1$ . Then set  $k \leftarrow k + 1$ .

swap  
virtual unswapping  
BSTAMP  
BST( $l$ )  
stamping



**143.** The following solution treats KINX and KSIZE as the unmodified algorithm treats TIMP and TSIZE. It deals in a somewhat more subtle way with CINX and CSIZE: If clause  $c$  originally had size  $k$ , and if  $j$  of its literals have become false while none have yet become true, CSIZE( $c$ ) will be  $k - j$ , but the nonfalse literals will not necessarily appear at the beginning of list CINX( $c$ ). As soon as  $j$  reaches  $k - 2$ , or one of the literals becomes true, clause  $c$  becomes inactive and it disappears from the KINX tables of all free literals. The algorithm won't look at CINX( $c$ ) or CSIZE( $c$ ) again until it unfixes the literal that deactivated  $c$ . Thus a big clause is inactive if and only if it has been satisfied (contains a true literal) or has become binary (has at most two nonfalse literals).

Knuth  
undoing

We need to modify only the three steps that involve TIMP. The modified step L1, call it L1', inputs the big clauses in a straightforward way.

Step L7' removes the formerly free variable  $X$  from the data structures by first deactivating all of the active big clauses that contain  $L$ : For each of the KSIZE( $L$ ) numbers  $c$  in KINX( $L$ ), and for each of the CSIZE( $c$ ) free literals  $u$  in CINX( $c$ ), we swap  $c$  out of  $u$ 's clause list as follows: Set  $s \leftarrow \text{KSIZE}(u) - 1$ ,  $\text{KSIZE}(u) \leftarrow s$ ; find  $t \leq s$  with  $\text{KINX}(u)[t] = c$ ; if  $t \neq s$  set  $\text{KINX}(u)[t] \leftarrow \text{KINX}(u)[s]$  and  $\text{KINX}(u)[s] \leftarrow c$ . [*Heuristic:* If the number of free literals remaining in  $c$  is small compared to  $c$ 's original size, for example if say 15 or 20 original literals have become false, the remaining nonfalse literals can usefully be swapped into the first CSIZE( $c$ ) positions of CINX( $c$ ) when  $c$  is being deactivated. The author's experimental implementation does this when CSIZE( $c$ ) is at most  $\theta$  times the original size, where the parameter  $\theta$  is normally 25/64.]

Then step L7' updates clauses for which  $L$  has become false: For each of the KSIZE( $\bar{L}$ ) numbers  $c$  in KINX( $\bar{L}$ ), set  $s \leftarrow \text{CSIZE}(c) - 1$  and  $\text{CSIZE}(c) \leftarrow s$ ; if  $s = 2$ , find the two free literals  $(u, v)$  in CINX( $c$ ), swap them into the first positions of that list, put them on a temporary stack, and swap  $c$  out of the clause lists of  $u$  and  $v$  as above.

Finally, step L7' does step L8' = L8 for all  $(u, v)$  on the temporary stack. [The maximum size of that stack will be the maximum of KSIZE( $l$ ) over all  $l$ , after step L1'; so we allocate memory for that stack as part of step L1'.]

In step L12' we set  $L \leftarrow R_E$ ,  $X \leftarrow |L|$ , and reactivate the clauses that involve  $X$  as follows: For each of the KSIZE( $\bar{L}$ ) numbers  $c$  in KINX( $\bar{L}$ ), proceeding in reverse order from the order used in L7', set  $s \leftarrow \text{CSIZE}(c)$ ,  $\text{CSIZE}(c) \leftarrow s + 1$ ; if  $s = 2$ , swap  $c$  back into the clause lists of  $v$  and  $u$ , where  $u = \text{CINX}(c)[0]$  and  $v = \text{CINX}(c)[1]$ . For each of the KSIZE( $L$ ) numbers  $c$  in KINX( $L$ ), and for each of the CSIZE( $c$ ) free literals  $u$  in CINX( $c$ ), again proceeding in reverse order from the order used in L7', swap  $c$  back into the clause list of  $u$ . The latter operation simply increases KSIZE( $u$ ) by 1.

**144.** False;  $h'(l) = 0.1$  if and only if the *complement*,  $\bar{l}$ , doesn't appear in any clause.

**145.** By symmetry we know that  $h(l) = h(\bar{l}) = h(10 - l)$  for  $1 \leq l \leq 9$  at depth 0, and the BIMP tables are empty. The first five rounds of refinement respectively give  $(h(1), \dots, h(5)) = (4.10, 4.10, 6.10, 6.10, 8.10)$ ,  $(5.01, 4.59, 6.84, 6.84, 7.98)$ ,  $(4.80, 4.58, 6.57, 6.57, 8.32)$ ,  $(4.88, 4.54, 6.72, 6.67, 8.06)$ , and  $(4.85, 4.56, 6.63, 6.62, 8.23)$ , slowly converging to the limiting values

$$(4.85810213, 4.55160111, 6.66761920, 6.63699698, 8.16778057).$$

When  $d = 1$ , however, the successively refined values of  $(h(1), h(\bar{1}), \dots, h(4), h(\bar{4}))$  are erratic and divergent:  $(2.10, 8.70, 3.10, 6.40, 3.10, 13.00, 3.10, 10.70)$ ,  $(5.53, 3.33, 9.05, 2.58, 5.40, 5.57, 8.24, 4.83)$ ,  $(1.43, 9.60, 2.32, 10.06, 1.30, 16.96, 1.97, 15.54)$ ,  $(8.04, 1.42, 12.31, 1.29, 7.45, 2.39, 11.91, 1.81)$ ,  $(0.32, 14.19, 0.42, 15.63, 0.30, 25.67, 0.43, 24.17)$ .

They eventually oscillate between limits that favor either positive or negative literals:

$$(0.1012, 17.4178, 0.1019, 19.7351, 0.1015, 31.6345, 0.1021, 30.4902) \quad \text{and} \\ (10.3331, 0.1538, 15.8485, 0.1272, 9.6098, 0.1809, 15.4207, 0.1542).$$

[Equations (64) and (65), which were inspired by survey propagation, first appeared in unpublished work of S. Mijnders, B. de Wilde, and M. J. H. Heule in 2010. The calculations above indicate that we needn't take  $h(l)$  too seriously, although it does seem to yield good results in practice. The author's implementation also sets  $h'(l) \leftarrow \Theta$  if the right-hand side of (65) exceeds a threshold parameter  $\Theta$ , which is 20.0 by default.]

**146.** Good results have been obtained with the simple formula  $h(l) = \varepsilon + \text{KSIZE}(\bar{l}) + \sum_{u \in \text{BIMP}(l), u \text{ free}} \text{KSIZE}(\bar{u})$ , which estimates the potential number of big-clause reductions that occur when  $l$  becomes true. The parameter  $\varepsilon$  is typically set to 0.001.

**147.**  $\infty, 600, 60, 30, 30$ .

**148.** If a problem is easy, we don't care if we solve it in 2 seconds or in .000002 seconds. On the other hand if a problem is so difficult that it can be solved only by looking ahead more than we can accomplish in a reasonable time, we might as well face the fact that we won't solve it anyway. There's no point in looking ahead at 60 variables when  $d = 60$ , because we won't be able to deal with more than  $2^{50}$  or so nodes in any reasonable search tree.

**149.** The idea is to maintain a binary string  $\text{SIG}(x)$  for each variable  $x$ , representing the highest node of the search tree in which  $x$  has participated. Let  $b_j = [\text{BRANCH}[j] = 1]$ , and set  $\sigma \leftarrow b_0 \dots b_{d-1}$  at the beginning of step L2,  $\sigma \leftarrow b_0 \dots b_d$  at the beginning of step L4. Then  $x$  will be a participant in step X3 if and only if  $\text{SIG}(x)$  is a prefix of  $\sigma$ .

We update  $\text{SIG}(x)$  when  $x = |u|$  or  $x = |v|$  in step L9, by setting  $\text{SIG}(x) \leftarrow \sigma$  unless  $\text{SIG}(x)$  is a prefix of  $\sigma$ . The initial value of  $\text{SIG}(x)$  is chosen so that it is never a prefix of any possible  $\sigma$ .

(Notice that  $\text{SIG}(x)$  needn't change when backtracking. In practice we can safely maintain only the first 32 bits of  $\sigma$  and of each string  $\text{SIG}(x)$ , together with their exact lengths, because lookahead computations need not be precise. In answer 143, updates occur not in step L9 but in step L7'; they are done for all literals  $u \neq \bar{L}$  that appear in any big clause containing  $\bar{L}$  that is being shortened for the first time.)

**150.** Asserting 7 at level 22 will also 22fix  $\bar{1}$ , because of the clause 147. Then  $\bar{1}$  will 22fix 3 and 9, which will 22fix  $\bar{2}$  and  $\bar{6}$ , then  $\bar{8}$ ; and clause 258 becomes false. Therefore 7 becomes proto true; and (62) makes 3, 6, 9 all proto true, contradicting 369.

**151.** For example, one such arrangement is

$$l: 2 \quad \bar{8} \quad 9 \quad 3 \quad \bar{1} \quad 6 \quad \bar{7} \quad \bar{4} \quad 4 \quad 7 \quad \bar{6} \quad 1 \quad \bar{3} \quad \bar{9} \quad 8 \quad \bar{2} \\ o(l): 4 \quad 2 \quad 10 \quad 14 \quad 6 \quad 16 \quad 8 \quad 12 \quad 22 \quad 26 \quad 18 \quad 28 \quad 20 \quad 24 \quad 32 \quad 30$$

[Digraphs that are obtainable in this way are called "partial orderings of dimension  $\leq 2$ ," or *permutation posets*. We've actually seen them in exercise 5.1.1–11, where the set of arcs was represented as a set of *inversions*. Permutation posets have many nice properties, which we shall study in Section 7.4.2. For example, if we reverse the order of the list and complement the offsets, we reverse the directions on the arrows. All but two of the 238 connected partially ordered sets on six elements are permutation posets. Unfortunately, however, permutation posets don't work well with lookahead when they aren't also forests. For example, after 10fixing '9' and its consequences, we would want to remove those literals from the  $R$  stack when 14fixing '3'; see (71). But then we'd want them back when 6fixing ' $\bar{1}$ '.]

survey propagation  
Mijnders  
de Wilde  
Heule  
threshold parameter  
partial orderings of dimension  $\leq 2$   
permutation posets  
inversions  
partially ordered sets

**152.** A single clause such as ‘12’ or ‘123’ would be an example, except that the autarky test in step X9 would solve the problem before we ever get to step X3. The clauses  $\{1\bar{2}\bar{3}, \bar{1}2\bar{3}, \bar{1}\bar{2}3, \bar{1}\bar{2}\bar{3}, 245, 3\bar{4}\bar{5}\}$  do, however, work: Level 0 branches on  $x_1$ , and level 1 discovers an autarky with  $b$  and  $c$  both true but returns  $l = 0$ . Then level 2 finds all clauses satisfied, although both of the free variables  $x_4$  and  $x_5$  are newbies.

[Indeed, the absence of free participants means that the fixed-true literals form an autarky. If  $\text{TSIZE}(l)$  is nonzero for any free literal  $l$ , some clause is unsatisfied. Otherwise all clauses are satisfied unless some free  $l$  has an unfixed literal  $l' \in \text{BIMP}(l)$ .]

**153.** Make the CAND array into a heap, with an element  $x$  of least rating  $r(x)$  at the top (see Section 5.2.3). Then, while  $C > C_{\max}$ , delete the top of the heap (namely  $\text{CAND}[0]$ ).

**154.** The child  $\rightarrow$  parent relations in the subforest will be  $d \rightarrow c \rightarrow a$ ,  $b \rightarrow a$ ,  $\bar{c} \rightarrow \bar{d}$ , and either  $\bar{a} \rightarrow \bar{b}$  or  $\bar{a} \rightarrow \bar{c}$ . Here’s one suitable sequence, using the latter:

```
preorder   $\bar{b}$   a  b  c  d   $\bar{d}$    $\bar{c}$    $\bar{a}$ 
2·postorder 2 10 4 8 6 16 14 12
```

**155.** First construct the dependency graph on the  $2C$  candidate literals, by extracting a subset of arcs from the BIMP tables. (This computation needn’t be exact, because we’re only calculating heuristics; an upper bound can be placed on the number of arcs considered, so that we don’t spend too much time here. However, it is important to have the arc  $u \rightarrow v$  if and only if  $\bar{v} \rightarrow \bar{u}$  is also present.)

Then apply Tarjan’s algorithm [see Section 7.4.1, or SGB pages 512–519]. If a strong component contains both  $l$  and  $\bar{l}$  for some  $l$ , terminate with a contradiction. Otherwise, if a strong component contains more than one literal, choose a representative  $l$  with maximum  $h(l)$ ; the other literals of that component regard  $l$  as their parent. Be careful to ensure that  $l$  is a representative if and only if  $\bar{l}$  is also a representative.

The result will be a sequence of candidate literals  $l_1 l_2 \dots l_S$  in topological order, with  $l_i \rightarrow l_j$  only if  $i > j$ . Compute the “height” of each  $l_j$ , namely the length of the longest path from  $l_j$  to a sink. Then every literal of height  $h > 0$  has a predecessor of height  $h - 1$ , and we let one such predecessor be its parent in the subforest. Every literal of height 0 (a sink) has a null parent. Traversal of this subforest in double order (exercise 2.3.1–18) now makes it easy to build the LL table in preorder while filling the L0 table in postorder.

**156.** If  $\bar{l}$  doesn’t appear in any clause of  $F$ , then  $A = \{l\}$  is clearly an autarky.

**157.** Well, *any* satisfying assignment is an autarky. But more to the point is the autarky  $\{1, 2\}$  for  $F = \{1\bar{2}\bar{3}, \bar{1}24, \bar{3}\bar{4}\}$ .

**158.**  $\text{BIMP}(l)$  and  $\text{TIMP}(l)$  will be empty, so  $w$  will be zero when Algorithm X looks ahead on  $l$ . Thus  $l$  will be forced true, at depth  $d = 0$ . (But pure literals that arise in subproblems for  $d > 0$  won’t be detected unless they’re among the preselected candidates.)

**159.** (a) False (consider  $A = \{1\}$ ,  $F = \{1, 2, \bar{1}2\}$ ); but true if we assume that  $F|A$  is computed as a multiset (so that  $F|A$  would be  $\{2, 2\} \not\subseteq F$  in that example).

(b) True: Suppose  $A = A' \cup A''$ ,  $A' \cap A'' = \emptyset$ , and  $A''$  or  $\bar{A}''$  touches  $C \in F|A'$ . Then  $C \cap A' = \emptyset$  and  $C \cup C' \in F$ , where  $C' \subseteq \bar{A}'$ . Since  $A$  or  $\bar{A}$  touches  $C \cup C'$ , some  $a \in C \cup C'$  is in  $A$ ; hence  $a \in A''$ .

**160.** (a) If the gray clauses are satisfiable, let all black literals be true. [Notice, incidentally, that the suggested example coloring works like a charm in (7).]

autarky test  
autarky  
heap  
heuristics  
Tarjan  
SGB  
height  
sink  
sink: a vertex with no successor  
double order  
preorder  
postorder  
satisfying assignment  
candidates  
multiset

(b) Given any set  $A$  of strictly distinct literals, color  $l$  black if  $l \in A$ , white if  $\bar{l} \in A$ , otherwise gray. Then  $A$  is an autarky if and only if condition (a) holds.

[E. A. Hirsch, *Journal of Automated Reasoning* **24** (2000), 397–420.]

**161.** (a) If  $F'$  is satisfiable, so is  $F$ . If  $F$  is satisfiable with at least one blue literal false, so is  $F'$ . If  $F$  is satisfiable with all the blue literals true, make all the black literals true (but keep gray literals unchanged). Then  $F'$  is satisfied, because every clause of  $F'$  that contains a black or blue literal is true, hence every clause that contains a white literal is true; the remaining clauses, whose literals are only orange and gray, each contain at least one true gray literal. [The black-and-blue condition is equivalent to saying that  $A$  is a *conditional autarky*, namely an autarky of  $F|L$ . Tseytin's notion of "extended resolution" is a special case, because the literals of  $A$  and  $L$  need not appear in  $F$ . See S. Jeannicot, L. Oxusoff, and A. Rauzy, *Revue d'intelligence artificielle* **2** (1988), 41–60, Section 6; O. Kullmann, *Theoretical Comp. Sci.* **223** (1999), 1–72, Sections 3, 4, and 14.]

(b) Without affecting satisfiability, we are allowed to add or delete any clause  $C = (a \vee \bar{l}_1 \vee \dots \vee \bar{l}_q)$  for which all clauses containing  $\bar{a}$  also contain  $l_1$  or  $\dots$  or  $l_q$ . (Such a clause is said to be "blocked" with respect to  $a$ , because  $C$  produces nothing but tautologies when it is resolved with clauses that contain  $\bar{a}$ .)

(c) Without affecting satisfiability, we are allowed to add or delete any or all of the clauses  $(\bar{l} \vee a_1), \dots, (\bar{l} \vee a_p)$ , if  $A$  is an autarky of  $F|l$ ; that is, we can do this if  $A$  is *almost* an autarky, in the sense that every clause that touches  $\bar{A}$  but not  $A$  contains  $l$ .

(d) Without affecting satisfiability, we are allowed to add or delete the clause  $(\bar{l} \vee a)$  whenever every clause that contains  $\bar{a}$  also contains  $l$ .

**162.** Construct a "blocking digraph" with  $l' \hookrightarrow l$  when every clause that contains literal  $\bar{l}$  also contains  $l'$ . (If  $l$  is a pure literal, we'll have  $l' \hookrightarrow l$  for *all*  $l'$ ; this case can be handled separately. Otherwise all in-degrees will be less than  $k$  in a  $k$ SAT problem, and the blocking digraph can be constructed in  $O(k^2m)$  steps if there are  $m$  clauses.)

(a) Then  $(l \vee l')$  is a blocked binary clause if and only if  $\bar{l} \hookrightarrow l'$  or  $\bar{l}' \hookrightarrow l$ . (Hence we're allowed in such cases to add both  $\bar{l} \rightarrow l'$  and  $\bar{l}' \rightarrow l$  to the *dependency* digraph.)

(b) Also  $A = \{a, a'\}$  is an autarky if and only if  $a \hookrightarrow a' \hookrightarrow a$ . (Moreover, any strong component  $\{a_1, \dots, a_t\}$  with  $t > 1$  is an autarky of size  $t$ .)

**163.** Consider the recurrence relations  $T_n = 1 + \max(T_{n-1}, T_{n-2}, 2U_{n-1})$ ,  $U_n = 1 + \max(T_{n-1}, T_{n-2}, U_{n-1} + V_{n-1})$ ,  $V_n = 1 + U_{n-1}$  for  $n > 0$ , with  $T_{-1} = T_0 = U_0 = V_0 = 0$ . We can prove that  $T_n, U_n, V_n$  are upper bounds on the step counts, where  $U_n$  refers to cases where  $F$  is known to have a nonternary clause, and  $V_n$  refers to cases when  $s = 1$  and R2 was entered from R3: The terms  $T_{n-1}$  and  $T_{n-2}$  represent autarky reductions in step R2; otherwise the recursive call in R3 costs  $U_{n-1}$ , not  $T_{n-1}$ , because at least one clause contains  $\bar{l}_s$ . We also have  $V_n = 1 + U_{n-1}$ , not  $1 + T_{n-1}$ , because the preceding step R3 either had a clause containing  $l_2$  not  $l_1$  or a clause containing  $\bar{l}_1$  not  $\bar{l}_2$ .

Fibonacci numbers provide the solution:  $T_n = 2F_{n+2} - 3 + [n = 0]$ ,  $U_n = F_{n+3} - 2$ ,  $V_n = F_{n+2} - 1$ . [Algorithm R is a simplification of a procedure devised by B. Monien and E. Speckenmeyer, *Discrete Applied Mathematics* **10** (1985), 287–295, who introduced the term "autarky" in that paper. A Stanford student, Juan Bulnes, had discovered a Fibonacci-bounded algorithm for 3SAT already in 1976; his method was, however, unattractive, because it also required  $\Omega(\phi^n)$  space.]

**164.** If  $k < 3$ ,  $T_n = n$  is an upper bound; so we may assume that  $k \geq 3$ . Let  $U_n = 1 + \max(T_{n-1}, T_{n-2}, U_{n-1} + V_{n-1,1}, \dots, U_{n-1} + V_{n-1,k-2})$ ,  $V_{n,1} = 1 + U_{n-1}$ , and  $V_{n,s} = 1 + \max(U_{n-1}, T_{n-2}, U_{n-1} + V_{n-1,s-1})$  for  $s > 1$ , where  $V_{n,s}$  refers to an entry at R2 from R3. The use of  $U_{n-1}$  in the formula for  $V_{n,s}$  is justified, because the

Hirsch  
conditional autarky  
lookahead autarky clauses, see black and bl  
Tseytin  
extended resolution  
Jeannicot  
Oxusoff  
Rauzy  
Kullmann  
blocked clauses  
tautologies  
resolved  
blocking digraph  
pure literal  
*dependency* digraph  
strong component  
recurrence relations  
Fibonacci numbers  
Monien  
Speckenmeyer  
autarky  
Bulnes

previous R3 either had a clause containing  $l_{s+1}$  not  $l_s$  or one containing  $\bar{l}_s$  not  $\bar{l}_{s+1}$ . One can show by induction that  $V_{n,s} = s + U_{n-1} + \dots + U_{n-s}$ ,  $U_n = V_{n,k-1}$ ; and  $T_n = U_n + U_{n-k} + 1 = 2U_{n-1} + 1$  if  $n \geq k$ . For example, the running time when  $k = 4$  is bounded by Tribonacci numbers, whose growth rate  $1.83929^n$  comes from the root of  $x^3 = x^2 + x + 1$ .

**165.** Clause  $\bar{1}\bar{3}\bar{4}$  in the example tells us that  $1, 3, 4 \notin A$ . Then  $1\bar{3}\bar{6}$  implies  $6 \notin A$ . But  $A = \{2, 5\}$  works, so it is maximum. There always is a maximum (not just maximal) positive autarky, because the union of positive autarkies is a positive autarky.

Each clause  $(v_1 \vee \dots \vee v_s \vee \bar{v}_{s+1} \vee \dots \vee \bar{v}_{s+t})$  of  $F$ , where the  $v$ 's are positive, tells us that  $v_1 \notin A$  and  $\dots$  and  $v_s \notin A$  implies  $v_{s+j} \notin A$ , for  $1 \leq j \leq t$ . Thus it essentially generates  $t$  Horn clauses, whose core is the set of all positive literals not in any positive autarky. A simple variant of Algorithm 7.1.1C will find this core in linear time; namely, we can modify steps C1 and C5 in order to get  $t$  Horn clauses from a single clause of  $F$ .

[By complementing a subset of variables, and prohibiting another subset, we can find the largest autarky  $A$  contained in any given set of strictly distinct literals. This exercise is due to unpublished work of O. Kullmann, V. W. Marek, and M. Truszczyński.]

**166.** Assume first that  $\text{PARENT}(l_0) = \Lambda$ , so that  $H(l_0) = 0$  at the beginning of X9 (see X6). Since  $l_0 = \text{LL}[j]$  is not fixed in context  $T$ , we have  $R_F = l_0$  by (62). And  $A = \{R_F, R_{F+1}, \dots, R_{E-1}\}$  is an autarky, because no clause touched by  $A$  or  $\bar{A}$  is entirely false or contains two unfixed literals. Thus we're allowed to force  $l_0$  true (which is what "do step X12 with  $l \leftarrow l_0$ " means).

On the other hand if  $w = 0$  and  $\text{PARENT}(l_0) = p$ , so that  $H(l_0) = H(p) > 0$  in X6, the set  $A = \{R_F, \dots, R_{E-1}\}$  is an autarky with respect to the clauses of  $F|p$ . Hence the additional clause  $(l_0 \vee \bar{p})$  doesn't make the clauses any less satisfiable, by the black and blue principle. (Notice that  $(\bar{l}_0 \vee p)$  is already a known clause; so in this case  $l_0$  is essentially being made equal to its parent.)

[The author's implementation therefore goes further and includes the step

$$\text{VAL}[|l_0|] \leftarrow \text{VAL}[|p|] \oplus ((l_0 \oplus p) \& 1), \quad (*)$$

which promotes the truth degree of  $l_0$  to that of  $p$ . This step violates the invariant relation (71), but Algorithm X doesn't rely on (71).]

**167.** If a literal  $l$  is fixed in context  $T$  during the lookahead, it is implied by  $l_0$ . In step X11 we have a case where  $l$  is also implied by  $\bar{l}_0$ ; hence we're allowed to force its truth, if  $l$  isn't already proto true. In step X6,  $\bar{l}_0$  is implied by  $l_0$ , so  $l_0$  must be false.

**168.** The following method works well in march: Terminate happily if  $F = n$ . (At this point in Algorithm L,  $F$  is the number of fixed variables, all of which are really true or really false.) Otherwise find  $l \in \{\text{LL}[0], \dots, \text{LL}[S-1]\}$  with  $l \bmod 2 = 0$  and maximum  $(H(l) + .1)(H(l+1) + .1)$ . If  $l$  is fixed, set  $l \leftarrow 0$ . (In that case, Algorithm X found at least one forced literal, although  $U$  is now zero; we want to do another lookahead before branching again.) Otherwise, if  $H(l) > H(l+1)$ , set  $l \leftarrow l+1$ . (A subproblem that is less reduced will tend to be more satisfiable.)

**169.** When  $a$  and  $b$  are positive, the function  $f(x) = e^{-ax} + e^{-bx} - 1$  is convex and decreasing, and it has the unique root  $\ln \tau(a, b)$ . Newton's method for solving this equation refines an approximation  $x$  by computing  $x' = x + f(x)/(ae^{-ax} + be^{-bx})$ . Notice that  $x$  is less than the root if and only if  $f(x) > 0$ ; furthermore  $f(x) > 0$  implies  $f(x') > 0$ , because  $f(x') > f(x) + (x' - x)f'(x)$  when  $f$  is convex. In particular we have  $f(1/(a+b)) > 0$ , because  $f(0) = 1$  and  $0' = 1/(a+b)$ , and we can proceed as follows:

**K1.** [Initialize.] Set  $j \leftarrow k \leftarrow 1$ ,  $x \leftarrow 1/(a_1 + b_1)$ .

Tribonacci numbers  
Horn clauses  
core  
Kullmann  
Marek  
Truszczyński  
black and blue principle  
author  
truth degree  
VAL  
invariant relation  
march  
convex  
Newton's method

- K2.** [Done?] (At this point  $(a_j, b_j)$  is the best of  $(a_1, b_1), \dots, (a_k, b_k)$ , and  $e^{-a_j x} + e^{-b_j x} \geq 1$ .) Terminate if  $k = s$ . Otherwise set  $k \leftarrow k + 1$ ,  $x' \leftarrow 1/(a_k + b_k)$ .
- K3.** [Find  $\alpha, \beta$ .] If  $x' < x$ , swap  $j \leftrightarrow k$  and  $x \leftrightarrow x'$ . Then set  $\alpha \leftarrow e^{-a_j x'}$  and  $\beta \leftarrow e^{-b_j x'}$ . Go to K2 if  $\alpha + \beta \leq 1$ .
- K4.** [Newtonize.] Set  $x \leftarrow x' + (\alpha + \beta - 1)/(a_j \alpha + b_j \beta)$ ,  $\alpha' \leftarrow e^{-a_k x'}$ ,  $\beta' \leftarrow e^{-b_k x'}$ ,  $x' \leftarrow x' + (\alpha' + \beta' - 1)/(a_k \alpha' + b_k \beta')$ , and return to K3. ■

floating point  
Tarjan  
autarkies  
invariant  
windfall  
undone

(The floating point calculations should satisfy  $e^u \leq e^v$  and  $u + w \leq v + w$  when  $u < v$ .)

**170.** If the problem is unsatisfiable, Tarjan's algorithm discovers  $l$  and  $\bar{l}$  in the same strong component. If it's satisfiable, Algorithm X finds autarkies (because  $w$  is always zero), thus forcing the value of all literals at depth 0.

**171.** It prevents double-looking on the same literal twice at the same search tree node.

**172.** When Algorithm Y concludes normally, we'll have  $T = \text{BASE} + \text{LO}[j]$ , even though BASE has changed. This relation is assumed to be invariant in Algorithm X.

**173.** The run reported in the text, using nonoptimized parameters (see exercise 513), did 29,194,670 double-looks (that is, executions of step Y2), and exited 23,245,231 times to X13 in step Y8 (thus successfully forcing  $l_0$  false in about 80% of those cases). Disabling Algorithm Y (i) increased the running time from 0.68 teramems to 1.13 teramems, with 24.3 million nodes. Disabling wraparound (ii) increased the time to 0.85 teramems, with 13.3 million nodes. Setting  $Y = 1$ , which disabled wraparound only in Algorithm Y, yielded 0.72 teramems, 11.3 meganodes. (Incidentally, the loops of Algorithm X wrapped around 40% of the time in the regular run, with a mean of 0.62 and maximum of 12; those of Algorithm Y had 20% wraparound, with a mean of 0.25; the maximum  $Y = 8$  was reached only 28 times.) Disabling the lookahead forest (iii) gave surprisingly good results: 0.70 teramems, 8.5 meganodes; there were fewer nodes [hence a more discriminating lookahead], but more time spent per node because of duplicated effort, although strong components were not computed. (Structured problems that have numerous binary clauses tend to generate more helpful forests than random 3SAT problems do.) Disabling compensation resolvents (iv) made very little difference: 0.70 teramems, 9.9 meganodes. But disabling windfalls (v) raised the cost to 0.89 teramems and 13.5 meganodes. And branching on a random  $l \in \text{LL}$  (vi) made the running time soar to 40.20 teramems, with 594.7 meganodes. Finally, disabling Algorithm X altogether (vii) was a disaster, leading to an estimated run time of well over  $10^{20}$  mems.

The weaker heuristics of exercise 175 yield 3.09 teramems and 35.9 meganodes.

**174.** Setting  $Y$  to a huge value such as PT will never get to step Y2. (But for (ii), (iii), ..., (vii) one must change the programs, not the parameters as they stand.)

**175.** Precompute the weights, by setting  $K_2 = 1$  and  $K_s \leftarrow \gamma K_{s-1} + .01$ , for  $s$  between 3 and the maximum clause size. (The extra .01 keeps this from being zero.) The third line of (72) must change to "take account of  $c$  for all  $c$  in  $\text{KINX}(\bar{L})$ ," where that means "set  $s \leftarrow \text{CSIZE}(c) - 1$ ; if  $s \geq 2$ , set  $\text{CSIZE}(c) \leftarrow s$  and  $w \leftarrow w + K_s$ ; otherwise if all literals of  $c$  are fixed false, set a flag; otherwise if some literal  $u$  of  $c$  isn't fixed (there will be just one), put it on a temporary stack." Before performing the last line of (72), go to CONFLICT if the flag is set; otherwise, for each unfixed  $u$  on the temporary stack, set  $W_i \leftarrow u$  and  $i \leftarrow i + 1$  and perform (62) with  $l \leftarrow u$ ; go to CONFLICT if some  $u$  on the temporary stack is fixed false. (A "windfall" in this more general setting is a clause for which all but one literal has been fixed false as a consequence of  $l_0$  being fixed true.)

Of course those changes to CSIZE need to be undone; a simulated false literal that has been "virtually" removed from a clause must be virtually put back. Fortunately,

the invariant relation (71) makes this task fairly easy: We set  $G \leftarrow F$  in step X5, and insert the following restoration loop at the very beginning of (72): “While  $G > F$ , set  $u \leftarrow R_{G-1}$ ; stop if  $u$  is fixed in context  $T$ ; otherwise set  $G \leftarrow G - 1$ , and increase  $\text{CSIZE}(c)$  by 1 for all  $c \in \text{KINX}(\bar{u})$ .” The restoration loop should also be performed, with  $T \leftarrow \text{NT}$ , just before terminating Algorithm X in steps X7 or X13.

[The additional step (\*) in answer 166 can't be used, because (71) is now crucial.]

Algorithm Y should change in essentially the same way as Algorithm X.

[See O. Kullmann, Report CSR 23-2002 (Swansea: Univ. of Wales, 2002), §4.2.]

**176.** (a)  $a_j \text{ --- } a_{j+1}$ ,  $a_j \text{ --- } b_j$ ,  $a_j \text{ --- } b_{j+1}$ ,  $b_j \text{ --- } c_j$ ,  $b_j \text{ --- } d_j$ ,  $c_j \text{ --- } d_j$ ,  $c_j \text{ --- } e_j$ ,  $d_j \text{ --- } f_j$ ,  $e_j \text{ --- } d_{j+1}$ ,  $e_j \text{ --- } f_{j+1}$ ,  $f_j \text{ --- } c_{j+1}$ ,  $f_j \text{ --- } e_{j+1}$ .

(b) Let  $(t_j, u_j, v_j, w_j, a_j, b_j, c_j, d_j, e_j, f_j)$  have colors  $(1, 2, 1, 1, 1, 2, 1, 3, 3, 2)$  when  $j$  is even,  $(2, 1, 2, 2, 3, 2, 3, 1, 1, 2)$  when  $j$  is odd. The lower bounds are obvious.

(c) Vertices  $a_j, e_j, f_j$  can't all have the same color, because  $b_j, c_j, d_j$  have distinct colors. Let  $\alpha_j$  denote the colors of  $a_j e_j f_j$ . Then  $\alpha_j = 112$  implies  $\alpha_{j+1} = 332$  or  $233$ ;  $\alpha_j = 121$  implies  $\alpha_{j+1} = 233$  or  $323$ ;  $\alpha_j = 211$  implies  $\alpha_{j+1} = 323$  or  $332$ ;  $\alpha_j = 123$  implies  $\alpha_{j+1} = 213$  or  $321$ . Since  $\alpha_1 = \alpha_{q+1}$ , the colors of  $\alpha_1$  must be distinct, and we can assume that  $\alpha_1 = 123$ . But then  $\alpha_j$  will be an odd permutation whenever  $j$  is even.

[See Rufus Isaacs, *AMM* **82** (1975), 233–234. Unpublished notes of E. Grinberg show that he had independently investigated the graph  $J_5$  in 1972.]

**177.** There are 20 independent subsets of  $V_j = \{a_j, b_j, c_j, d_j, e_j, f_j\}$  when  $q > 1$ ; eight of them contain none of  $\{b_j, c_j, d_j\}$  while four contain  $b_j$ . Let  $A$  be a  $20 \times 20$  transition matrix, which indicates when  $RUC$  is independent for each independent subset  $R \subseteq V_j$  and  $C \subseteq V_{j+1}$ . Then  $I_q$  is  $\text{trace}(A^q)$ ; and the first eight values are 8, 126, 1052, 11170, 112828, 1159416, 11869768, 121668290. The characteristic polynomial of  $A$ ,  $x^{12}(x^2 - 2x - 1)(x^2 + 2x - 1)(x^4 - 8x^3 - 25x^2 + 20x + 1)$ , has nonzero roots  $\pm 1 \pm \sqrt{2}$  and  $\approx -2.91, -0.05, +0.71, +10.25$ ; hence  $I_q = \Theta(r^q)$ , where  $r \approx 10.24811166$  is the dominant root. *Note:* The number of *kernels* of  $L(J_q)$  is respectively 2, 32, 140, 536, 2957, 14336, 70093, 348872, for  $1 \leq q \leq 8$ , and its growth rate is  $\approx 4.93^q$ .

**178.** With the first ordering, the top  $18k$  levels of the search tree essentially represent all of the ways to 3-color the subgraph  $\{a_j, b_j, c_j, d_j, e_j, f_j \mid 1 \leq j \leq k\}$ ; and there are  $\Theta(2^k)$  ways to do that, by answer 176. But with the second ordering, the top  $6kq$  levels essentially represent all of the independent sets of the graph; and there are  $\Omega(10.2^k)$  of those, by answer 177.

Empirically, Algorithm B needs respectively 1.54 megamems, 1.57 gigamems, and 1.61 teramems to prove unsatisfiability when  $q = 9, 19$ , and  $29$ , using the first ordering; but it needs 158 gigamems already for  $q = 5$  with the second! Additional clauses, which require color classes to be kernels (see answer 14), reduce that time to 492 megamems.

Algorithm D does badly on this sequence of problems: When  $q = 19$ , it consumes 37.6 gigamems, even with the “good” ordering. And when  $q = 29$ , its cyclic method of working somehow transforms the good ordering into a bad ordering on many of the variables at depths 200 or more. It shows no sign of being anywhere near completion even after spending a *petamem* on that problem!

Algorithm L, which is insensitive to the ordering, needs 2.42 megamems, 2.01 gigamems, and 1.73 teramems when  $q = 9, 19$ , and  $29$ . Thus it appears to take  $\Theta(2^q)$  steps, and to be slightly slower than Algorithm B as  $q$  grows, although exercise 232 shows that a clairvoyant lookahead procedure could theoretically do much better.

Algorithm C triumphs here, as shown in Fig. 49.

Kullmann  
odd permutation  
Isaacs  
Grinberg  
transition matrix  
trace of a matrix: The sum of its diagonal elements  
kernels  
kernels  
comparison of SAT solvers

**179.** This is a straightforward exact cover problem. If we classify the solutions according to how many asterisks occur in each coordinate, it turns out that exactly (10, 240, 180, 360, 720, 480, 1440, 270, 200, 480) of them are respectively of type (00088, 00268, 00448, 00466, 02248, 02266, 02446, 04444, 22228, 22246).

By complementation, we see that 4380 choices of 8 clauses are unsatisfiable; hence  $q_8 = 1 - 4380/\binom{80}{8} = 1 - 4380/28987537150 \approx 0.9999998$ .

**180.** With  $N$  variables  $y_j$ , one for each possible clause  $C_j$ , the function  $f(y_1, \dots, y_N) = [\bigwedge\{C_j \mid y_j = 1\} \text{ is satisfiable}]$  is  $\bigvee_x f_x(y)$ , where  $f_x(y) = [x \text{ satisfies } \bigwedge\{C_j \mid y_j = 1\}]$  is simply  $\bigwedge\{\bar{y}_j \mid x \text{ makes } C_j \text{ false}\}$ . For instance if  $k = 2$  and  $n = 3$ , and if  $C_1, C_7, C_{11}$  are the clauses  $(x_1 \vee x_2), (x_1 \vee \bar{x}_3), (x_2 \vee \bar{x}_3)$ , then  $f_{001}(y_1, \dots, y_{12}) = \bar{y}_1 \wedge \bar{y}_7 \wedge \bar{y}_{11}$ .

Each function  $f_x$  has a very simple BDD, but of course the OR of  $2^n$  of them will not be simple. This problem is an excellent example where no natural ordering of the clause variables is evident, but the method of sifting is able to reduce the BDD size substantially. In fact, the clauses for  $k = 3$  and  $n = 4$  can be ordered cleverly so that the corresponding 32-variable BDD for satisfiability has only 1362 nodes! The author's best result for  $k = 3$  and  $n = 5$ , however, was a BDD of size 2,155,458. The coefficients of its generating function (exercise 7.1.4–25) are the desired numbers  $Q_m$ .

The largest such count,  $Q_{35} = 3,449,494,339,791,376,514,416$ , is so enormous that we could not hope to enumerate the relevant sets of 35 clauses by backtracking.

**181.** The previous exercise essentially computed the generating function  $\sum_m Q_m z^m$ ; now we want the *double* generating function  $\sum_{l,m} T_{l,m} w^l z^m$ , where  $T_{l,m}$  is the number of ways to choose  $m$  different  $k$ -clauses in such a way that these clauses are satisfied by exactly  $l$  vectors  $x_1 \dots x_n$ . To do this, instead of taking the OR of the simple functions  $f_x$ , we compute the BDD base that contains all of the symmetric Boolean functions  $S_l(f_{0\dots 0}, \dots, f_{1\dots 1})$  for  $0 \leq l \leq 2^n$ , as follows (see exercise 7.1.4–49): Consider the subscript  $x$  to be a binary integer, so that the functions are  $f_x$  for  $0 \leq x < 2^n$ . Start with  $S_l = 0$  for  $-1 \leq l \leq 2^n$ , except that  $S_0 = 1$ . Then do the following for  $x = 0, \dots, 2^n - 1$  (in that order): Set  $S_l = f_x ? S_{l-1} : S_l$  for  $l = x + 1, \dots, 0$  (in that order).

After this computation, the generating function for  $S_l$  will be  $\sum_m T_{l,m} z^m$ . In the author's experiments, the sifting algorithm found an ordering of the 80 clauses for  $k = 3$  and  $n = 5$  so that only about 6 million nodes were needed when  $x$  had reached 24; afterwards, however, sifting took too long, so it was turned off. The final BDD base had approximately 87 million nodes, with many nodes shared between the individual functions  $S_l$ . The total running time was about 22 gigamems.

**182.**  $T_0 = 32$  and  $T_1 = 28$  and  $T_m = 0$  for  $71 \leq m \leq 80$ . Otherwise  $\min T_m < \max T_m$ .

**183.** Let  $t_m = \Pr(T_m = 1)$ , and suppose that we obtain clauses one by one until reaching an unsatisfiable set. The fact that  $t_m$  gets reasonably large suggests that we probably have accumulated a *uniquely* satisfiable set just before stopping. (That probability is  $2^{-k} N \sum_m t_m / (N - m)$ , which turns out to be  $\approx 0.8853$  when  $k = 3$  and  $n = 5$ .)

However, except for the fact that both Figs. 42 and 43 are bell-shaped curves with roughly the same tendency to be relatively large or small at particular values of  $m$ , there is apparently no strong mathematical connection. The probabilities in Fig. 43 sum to 1; but the sum of probabilities in Fig. 42 has no obvious significance.

When  $n$  is large, uniquely satisfiable sets are encountered only rarely. The final set before stopping a.s. has at most  $f(n)$  solutions, for certain functions  $f$ ; but how fast does the smallest such  $f$  grow? [See D. J. Aldous, *J. Theoretical Probability* 4 (1991), 197–211, for related ideas.]

exact cover  
sifting  
backtracking  
generating function  
BDD base  
symmetric Boolean functions  
mux  
if-then-else, see mux  
uniquely  
Aldous



**184.** The probability  $\hat{q}_m$  is  $\hat{Q}_m/N^m$ , where  $\hat{Q}_m$  counts the choices  $(C_1, \dots, C_m)$  for which  $C_1 \wedge \dots \wedge C_m$  is satisfiable. The number of such choices that involve  $t$  distinct clauses is  $t! \binom{m}{t}$  times  $Q_t$ , because  $\binom{m}{t}$  enumerates set partitions; see Eq. 3.3.2-(5).

**185.**  $\hat{q}_m = \sum_{t=0}^N \binom{m}{t} t! q_t \binom{N}{t} / N^m \geq q_m \sum_{t=0}^N \binom{m}{t} t! \binom{N}{t} / N^m = q_m$ .

**186.**  $\sum_m \sum_t \binom{m}{t} t! q_t \binom{N}{t} N^{-m}$  can be summed on  $m$ , since  $\sum_m \binom{m}{t} N^{-m} = 1/(N-1)^t$  by Eq. 1.2.9-(28). Similarly, the derivative of 1.2.9-(28) shows that  $\sum_m m \binom{m}{t} N^{-m} = (N/(N-1) + \dots + N/(N-t))/(N-1)^t$ .

**187.** In this special case,  $q_m = [0 \leq m < N]$  and  $p_m = [m = N]$ ; hence  $S_{n,n} = N = 2^n$  (and the variance is zero). By (78), we also have  $\hat{S}_{n,n} = NH_N$ ; indeed, the coupon collector's test (exercise 3.3.2-8) is an equivalent way to view this situation.

**188.** Now  $q_m = 2^m n^m / (2n)^m$ . It follows by (78) that  $\hat{S}_{1,n} = \sum_{m=0}^n 2^m n^m / (2n-1)^m$ , because  $N = 2n$ . The identity  $2^m n^m / (2n-1)^m = 2q_m - q_{m+1}$  yields the surprising fact that  $\hat{S}_{1,n} = (2q_0 - q_1) + (2q_1 - q_2) + \dots = 1 + S_{1,n}$ ; and we also have  $\hat{S}_{1,n} - 1 = \frac{2n}{2n-1} S_{1,n-1}$ . Hence, by induction, we obtain the (even more surprising) closed forms

$$S_{1,n} = 4^n / \binom{2n}{n}, \quad \hat{S}_{1,n} = 4^n / \binom{2n}{n} + 1.$$

So random 1SAT problems become unsatisfiable after  $\sqrt{\pi n} + O(1)$  clauses, on average.

**189.** With the autosifting method in the author's experimental BDD implementation, the number of BDD nodes, given a sequence of  $m$  distinct clauses when  $k = 3$  and  $n = 50$ , increased past 1000 when  $m$  increased from 1 to about 30, and it tended to peak at about 500,000 when  $m$  was slightly more than 100. Then the typical BDD size fell to about 50,000 when  $m = 150$ , and to only about 500 when  $m = 200$ .

BDD methods break down when  $n$  is too large, but when they apply we can count the total number of solutions remaining after  $m$  steps. In the author's tests with  $k = 3$ ,  $n = 50$ , and  $m = 200$ , this number varied from about 25 to about 2000.

**190.** For example,  $S_1(x_1, \dots, x_n)$  can't be expressed in  $(n-1)$ CNF: All clauses of length  $n-1$  that are implied by  $S_1(x_1, \dots, x_n)$  are also implied by  $S_{\leq 1}(x_1, \dots, x_n)$ .

**191.** Let  $f(x_0, \dots, x_{2^n-1}) = 1$  if and only if  $x_0 \dots x_{2^n-1}$  is the truth table of a Boolean function of  $n$  variables that is expressible in  $k$ CNF. This function  $f$  is the conjunction of  $2^n$  constraints  $c(t)$ , for  $0 \leq t = (t_0 \dots t_{2^n-1})_2 < 2^n$ , where  $c(t)$  is the following condition: If  $x_t = 0$ , then  $\bigvee \{x_y \mid 0 \leq y < 2^n, (y \oplus t) \& m = 0\}$  is 0 for some  $n$ -bit pattern  $m$  that has  $\nu m = k$ . By combining these constraints we can compute the BDD for  $f$  when  $n = 4$  and  $k = 3$ ; it has 880 nodes, and 43,146 solutions.

Similarly we have the following results, analogous to those in Section 7.1.1:

|      | $n=0$ | $n=1$ | $n=2$ | $n=3$ | $n=4$  | $n=5$       | $n=6$             |
|------|-------|-------|-------|-------|--------|-------------|-------------------|
| 1CNF | 2     | 4     | 10    | 28    | 82     | 244         | 730               |
| 2CNF | 2     | 4     | 16    | 166   | 4,170  | 224,716     | 24,445,368        |
| 3CNF | 2     | 4     | 16    | 256   | 43,146 | 120,510,132 | 4,977,694,100,656 |

And if we consider equivalence under complementation and permutation, the counts are:

|      | $n=0$ | $n=1$ | $n=2$ | $n=3$ | $n=4$ | $n=5$  | $n=6$       |
|------|-------|-------|-------|-------|-------|--------|-------------|
| 1CNF | 2     | 3     | 4     | 5     | 6     | 7      | 8           |
| 2CNF | 2     | 3     | 6     | 14    | 45    | 196    | 1,360       |
| 3CNF | 2     | 3     | 6     | 22    | 253   | 37,098 | 109,873,815 |

**192.** (a)  $S(p) = \sum_{m=0}^N p^m (1-p)^{N-m} Q_m$ . (b) We have  $\int_0^N (t/N)^m (1-t/N)^{N-m} dt = NB(m+1, N-m+1) = \frac{N}{N+1} \binom{N}{m}$ , by exercises 1.2.6-40 and 41; hence  $\bar{S}_{k,n} = \frac{N}{N+1} \sum_{m=0}^N q_m = \frac{N}{N+1} S_{k,n}$ . [See B. Bollobás, *Random Graphs* (1985), Theorem II.4.]

set partitions  
Stirling subset numbers  
coupon collector's test  
autosifting  
sifting  
symmetric Boolean functions  
symmetric threshold functions  
truth table  
Boolean functions in kCNF  
BDD  
Bollobás

**194.** A similar question, about proofs of *unsatisfiability* when  $\alpha > \limsup_{n \rightarrow \infty} S_{3,n}/n$ , is also wide open.

**195.**  $E X = 2^n \Pr(0 \dots 0 \text{ satisfies all}) = 2^n (1 - 2^{-k})^m = \exp(n \ln 2 + m \ln(1 - 2^{-k})) < 2 \exp(-2^{-k-1} n \ln 2)$ . Thus  $S_k(\lfloor (2^k \ln 2)n \rfloor, n) = \Pr(X > 0) \leq \exp(-\Omega(n))$ . [*Discrete Applied Math.* **5** (1983), 77–87. Conversely, in *J. Amer. Math. Soc.* **17** (2004), 947–973, D. Achlioptas and Y. Peres use the second moment principle to show that  $(2^k \ln 2 - O(k))n$  random  $k$ SAT clauses are almost always satisfiable by vectors  $x$  with  $\nu x \approx n/2$ . Careful study of “covering assignments” (see exercise 364) leads to the sharp bounds

$$2^k \ln 2 - \frac{1 + \ln 2}{2} - O(2^{-\frac{k}{3}}) \leq \liminf_{n \rightarrow \infty} \alpha_k(n) \leq \limsup_{n \rightarrow \infty} \alpha_k(n) \leq 2^k \ln 2 - \frac{1 + \ln 2}{2} + O(2^{-\frac{k}{3}});$$

see A. Coja-Oghlan and K. Panagiotou, [arXiv:1310.2728 \[math.CO\]](https://arxiv.org/abs/1310.2728) (2013), 48 pages.]

**196.** The probability is  $((n - t)^k/n^k)^{\alpha n + O(1)} = e^{-kt\alpha}(1 + O(1/n))$  that  $\alpha n + O(1)$  random  $k$ SAT clauses omit  $t$  given letters. Let  $p = 1 - (1 - e^{-k\alpha})^k$ . By inclusion and exclusion, the first clause will be easy with probability  $p(1 + O(1/n))$ , and the first two will both be easy with probability  $p^2(1 + O(1/n))$ . Thus if  $X = \sum_{j=1}^m [\text{clause } j \text{ is easy}]$ , we have  $E X = pm + O(1)$  and  $E X^2 = p^2 m^2 + O(m)$ . Hence, by Chebyshev’s inequality,  $\Pr(|X - pm| \geq r\sqrt{m}) = O(1/r^2)$ .

**197.** By Stirling’s approximation,  $\ln q(a, b, A, B, n) = nf(a, b, A, B) + g(a, b, A, B) - \frac{1}{2} \ln 2\pi n - (\delta_{an} - \delta_{(a+b)n}) - (\delta_{bn} - \delta_{(b+B)n}) - (\delta_{An} - \delta_{(a+A)n}) - (\delta_{Bn} - \delta_{(A+B)n}) - \delta_{(a+b+A+B)n}$ , where  $\delta_n$  is positive and decreasing. And we must have  $f(a, b, A, B) \leq 0$ , since  $q(a, b, A, B, n) \leq 1$ . The  $O$  estimate is uniform when  $0 < \delta \leq a, b, A, B \leq M$ .

**198.** Consider one of the  $N^M$  possible sequences of  $M$  3SAT clauses, where  $N = 8 \binom{n}{3}$  and  $M = 5n$ . By exercise 196 it contains  $g = 5(1 - (1 - e^{-15})^3)n + O(n^{3/4})$  easy clauses, except with probability  $O(n^{-1/2})$ . Those clauses, though rare, don’t affect the satisfiability; and all  $\binom{M}{g}$  of the ways to insert them among the  $r = M - g$  others are equally likely, so they tend to dampen the transition.

Let  $l \leq r$  be maximum so that the first  $l$  noneasy clauses are satisfiable, and let  $p(l, r, g, m)$  be the probability that, when drawing  $m$  balls from an urn that contains  $g$  green balls and  $r$  red balls, at most  $l$  balls are red. Then  $S_3(m, n) = \sum p(l, r, g, m)/N^M$  and  $S_3(m', n) = \sum p(l, r, g, m')/N^M$ , summed over all  $N^M$  sequences.

To complete the proof we shall show that

$$p(l, r, g, m + 1) = p(l, r, g, m) - O(n^{-1/2}) \quad \text{when } 3.5n < m < 4.5n;$$

hence  $S_3(m + 1, n) = S_3(m, n) - O(n^{-1/2})$ ,  $S_3(m, n) - S_3(m', n) = O((m' - m)n^{-1/2})$ . Notice that  $p(l, r, g, m) = p(l, r, g, m + 1)$  when  $m < l$  or  $m > l + g$ ; thus we may assume that  $l$  lies between  $3.4n$  and  $4.6n$ . Furthermore the difference

$$d_m = p(l, r, g, m) - p(l, r, g, m + 1) = \frac{\binom{m}{l} \binom{r+g-m-1}{r-l-1}}{\binom{r+g}{r}} = \frac{\binom{m}{l} \binom{r+g-m}{r-l}}{\binom{r+g}{r}} \frac{r-l}{r+g-m}$$

has a decreasing ratio  $d_m/d_{m-1} = (m/(m-l))((l+g+1-m)/(r+g-m))$  when  $m$  increases from  $l$  to  $l+g$ . So  $\max d_m$  occurs at  $m \approx l(r+g)/r$ , where this ratio is  $\approx 1$ . Now exercise 197 applies with  $a = l/n$ ,  $b = \rho g/n$ ,  $A = (r-l)/n$ ,  $B = (1-\rho)g/n$ ,  $\rho = l/r$ .

[D. B. Wilson, in *Random Structures & Algorithms* **21** (2002), 182–195, showed that similar methods apply to many other threshold phenomena.]

Achlioptas  
Peres  
second moment principle  
covering assignments  
Coja-Oghlan  
Panagiotou  
inclusion and exclusion  
Chebyshev’s inequality  
Stirling’s approximation  
balls and urns  
urns and balls  
Wilson  
threshold phenomena

**199.** (a) Given the required letters  $\{a_1, \dots, a_t\}$ , there are  $m$  ways to place the leftmost  $a_1$ , then  $m - 1$  ways to place the leftmost  $a_2$ , and so on; then there are at most  $N$  ways to fill in each of the remaining  $m - t$  slots.

inclusion and exclusion  
second moment principle

(b) By inclusion and exclusion: There are  $(N - k)^m$  words that omit  $k$  of the letters.

(c)  $N^{-m} \sum_k \binom{t}{k} (-1)^k \sum_j \binom{m}{j} N^{m-j} (-k)^j = \sum_j \binom{m}{j} (-1)^{j+t} N^{-j} A_j$ , where  $A_j = \sum_k \binom{t}{k} (-1)^{t-k} k^j = \{j\}^t!$  by Eq. 1.2.6-(53).

**200.** (a) The unsatisfiable digraph must contain a strong component with a path

$$\bar{l}_t \rightarrow l_1 \rightarrow \dots \rightarrow \bar{l}_t \rightarrow l_{t+1} \rightarrow \dots \rightarrow l_t = \bar{l}_t,$$

where  $l_1, \dots, l_t$  are strictly distinct. This path yields an  $s$ -snare  $(C; t, u)$  if we set  $s$  to the smallest index such that  $|l_{s+1}| = |l_u|$  for some  $u$  with  $1 \leq u < s$ .

(b) No:  $(x \vee y) \wedge (\bar{y} \vee x) \wedge (\bar{x} \vee y)$  and  $(x \vee y) \wedge (\bar{y} \vee x) \wedge (\bar{x} \vee \bar{y})$  are both satisfiable.

(c) Apply exercise 199(a) with  $t = s + 1$ ,  $N = 2n(n - 1)$ ; note that  $m^{\frac{s+1}{2}} \leq m^{s+1}$ .

**201.** (a) Set  $(l_i, l_{i+1}) \leftarrow (x_1, x_2)$  or  $(\bar{x}_2, \bar{x}_1)$ , where  $0 \leq i < 2t$  (thus  $4t$  ways).

(b) Set  $(l_i, l_{i+1}, l_{i+2}) \leftarrow (x_1, x_2, x_3)$  or  $(\bar{x}_3, \bar{x}_2, \bar{x}_1)$ , where  $0 \leq i < 2t$ ; also  $(\bar{l}_1, l_t, l_{t+1})$  or  $(l_{-1}, l_t, \bar{l}_{2t-1}) \leftarrow (x_1, x_2, x_3)$  or  $(\bar{x}_3, \bar{x}_2, \bar{x}_1)$  (total  $4t + 4$  ways, if  $t > 2$ ).

(c)  $(l_1, l_{t-1}, l_t)$  or  $(\bar{l}_{2t-1}, \bar{l}_{t+1}, \bar{l}_t) \leftarrow (x_1, x_{t-1}, x_t)$  or  $(\bar{x}_{t-1}, \bar{x}_1, x_t)$  (4 ways).

(d)  $l_i$  or  $\bar{l}_{2t-i} \leftarrow x_i$  or  $\bar{x}_{t-i}$ , for  $1 \leq i \leq t$  (4 ways, if you understand this notation).

(e) By part (a), it is  $2t \times 4t = 8t^2$ .

(f) Parts (b) and (c) combine to give  $N(3, 2) = (2t + 2) \times (4t + 4) + 2 \times 4 = 8(t^2 + 2t + 2)$  when  $t > 2$ . From part (d),  $N(t, t) = 8$ . Also  $N(2t - 1, 2t) = 8$ ; this is the number of snakes that specify the same  $2t$  clauses. (Incidentally, when  $t = 5$  the generating function  $\sum_{q,r} N(q, r) w^q z^r$  is  $1 + 200w^2 z^1 + (296w^3 + 7688w^4) z^2 + (440w^4 + 12800w^5 + 55488w^6) z^3 + (640w^5 + 12592w^6 + 66560w^7 + 31104w^8) z^4 + (8w^5 + 736w^6 + 8960w^7 + 32064w^8 + 6528w^9) z^5 + (32w^6 + 704w^7 + 4904w^8 + 4512w^9) z^6 + (48w^7 + 704w^8 + 1232w^9) z^7 + (64w^8 + 376w^9) z^8 + 80w^9 z^9 + 8w^9 z^{10}$ .)

(g) The other  $l$ 's can be set in at most  $2^{2t-1-q} (n - q)^{\frac{2t-1-q}{2}} = R / (2^q n^{\frac{q}{2}})$  ways.

(h) We may assume that  $r < 2t$ . The  $r$  chosen clauses divide into connected components, which are either paths or a "central" component that contains either  $(\bar{x}_0 \vee x_1)$  and  $(\bar{x}_{t-1} \vee x_t)$  or  $(\bar{x}_t \vee x_{t+1})$  and  $(\bar{x}_{2t-1} \vee x_0)$ . Thus  $q$  equals  $r$  plus the number of components, minus 1 if the central component includes a cycle. If the central component is present, we must set  $l_t \leftarrow x_t$  or  $\bar{x}_t$ , and there are at most 8 ways to complete the mapping of that component. And  $N(r, r) = 16(r + 1 - t)$  for  $t < r < 2t$ .

Cases with  $k > 0$  paths can be chosen in at most  $\binom{2t+2}{2k}$  ways, because we choose the starting and ending points, and they can be mapped in at most  $2^k k! \binom{2t+2}{2k}$  ways; so they contribute  $\sum_{k>0} O(t^{4k} k / (k!^3 n^k)) = O(t^4/n)$  to  $(2n)^r p_r$ . The noncyclic central components, which can be chosen in  $\Theta(t^4)$  ways, also contribute  $O(t^4/n)$ .

**202.** (a)  $m(m-1) \dots (m-r+1)/m^r \geq (1 - \binom{r}{2}/m)$ ;  $(2n(n-1)-r)^{m-r}/(2n(n-1))^{m-r} \geq 1 - (m-r)r/(2n(n-1))$  when  $r \leq m < 2n(n-1)$ ; and both factors are  $\leq 1$ .

(b) The term of (95) for  $r = 0$  is 1 plus a negligible error. The contribution of  $O(t^4/n)$  for  $r > 0$  is  $O(n^{4/5+1/6-1})$ , because  $\sum_{r \geq 0} (1 + n^{-1/6})^{-r} = n^{1/6} + 1$ . And the contributions of (96) to (95) for  $r \geq t$  are exponentially small, because in that range we have  $(1 + n^{-1/6})^{-t} = \exp(-t \ln(1 + n^{-1/6})) = \exp(-\Omega(n^{1/30}))$ . Finally, then, by the second moment principle MPR-(22),  $S_2([n + n^{5/6}], n) \leq 1 - \Pr(X > 0) \leq 1 - (EX)^2/(EX^2) = 1 - 1/((EX^2)/(EX)^2) = 1 - 1/(1 + O(n^{-1/30})) = O(n^{-1/30})$ .

**203.** (a)  $EX = d^n EX(1, \dots, 1)$ , by symmetry; and  $EX(1, \dots, 1) = (1 - p)^m$ , because each set of  $q$  clauses is falsified with probability  $p$ . So  $EX = \exp((r \ln(1 - p) + 1)n \ln d)$  is exponentially small when  $r \ln(1 - p) + 1 < 0$ ; and we know that  $\Pr(X > 0) \leq EX$ .

Tovey

(b) Let  $\theta_s = \binom{s}{2} / \binom{n}{2} = \frac{s(s-1)}{n(n-1)}$ , and consider a random constraint set, given that  $X(1, \dots, 1) = 1$ . With probability  $\theta_s$ , both  $u$  and  $v$  have color 1 and the constraint is known to be satisfied. But with probability  $1 - \theta_s$ , it holds with probability  $\binom{d^2-2}{q} / \binom{d^2-1}{q}$ . Thus  $p_s = (\theta_s + (1 - \theta_s)(d^2 - pd^2 - 1) / (d^2 - 1))^m$ .

(c) We have  $\Pr(X > 0) \geq d^n(1-p)^m / E(X | X(1, \dots, 1) = 1)$ , from the inequality and symmetry; and the denominator is  $\sum_{s=0}^n \binom{n}{s} (d-1)^{n-s} p_s$ . We can replace  $p_s$  by the simpler value  $p'_s = (1-p + ps^2/n^2)^m$ , because  $p_s < (\theta_s + (1 - \theta_s)(1-p))^m = (1-p + \theta_s p)^m < p'_s$ . And we can divide the simplified sum by  $d^n(1-p)^m$ .

(d) We have  $\sum_{s=0}^{3n/d} t_s = e^{O(m/d^2)} \sum_{s=0}^{3n/d} \binom{n}{s} (\frac{1}{d})^s (1 - \frac{1}{d})^{n-s}$ , because  $s^2/n^2 = O(1/d^2)$  when  $s \leq 3n/d$ . This sum is  $\geq 1 - (e^2/27)^{n/d}$  by exercise 1.2.10-22; and the crucial assumption that  $\alpha > \frac{1}{2}$  makes  $m/d^2 \rightarrow 0$ .

(e) Transition between increase and decrease occurs when  $x_s \approx 1$ ; and we have

$$x_s = \frac{n-s}{s+1} \frac{1}{d-1} \left( 1 + \frac{(2s+1)p}{(1-p)n^2 + ps^2} \right)^m \approx \exp\left( \ln \frac{1-\sigma}{\sigma} + \left( \frac{2pr\sigma}{1-p+p\sigma^2} - 1 \right) \ln d \right)$$

when  $s = \sigma n$ . Let  $f(\sigma) = 2pr\sigma / (1-p+p\sigma^2) - 1$ , and notice that  $f'(\sigma) > 0$  for  $0 \leq \sigma < 1$  because  $p \leq \frac{1}{2}$ . Furthermore our choice of  $r$  makes  $f(\frac{1}{2}) < 0 < f(1)$ . Setting  $g(\sigma) = f(\sigma) / \ln \frac{\sigma}{1-\sigma}$ , we seek values of  $\sigma$  with  $g(\sigma) = 1/\ln d$ . There are three such roots, because  $g(1/N) \approx -f(0) / \ln N \geq 1/\ln N$ ;  $g(\frac{1}{2} \pm 1/N) \approx \mp f(\frac{1}{2})N/4$ ; and  $g(1 - 1/N) \approx f(1) / \ln N$ .

(f) At the second peak, where  $s = n - n/d^{f(1)}$ , we have (see exercise 1.2.6-67)

$$t_s < \left( \frac{ned}{n-s} \right)^{n-s} \left( \frac{1}{d} \right)^n \left( 1 + \frac{p}{1-p} \right)^m = \exp((- \epsilon + O(1/d^{f(1)}))n \ln d),$$

which is exponentially small. And when  $s = 3n/d$ ,  $t_s < (\frac{ne}{sd})^s e^{O(m/d^2)} = O((e/3)^{3n/d})$  is also exponentially small. Consequently  $\sum_{s=3n/d}^n t_s$  is exponentially small.

[This derivation holds also when the random constraints are  $k$ -ary instead of binary, with  $q = pd^k$  and  $\alpha > 1/k$ . See *J. Artificial Intelligence Res.* **12** (2000), 93-103.]

**204.** (a) If the original literals  $\pm x_j$  that involve variable  $x_j$  correspond to  $\sigma_1 X_{i(1)}, \dots, \sigma_p X_{i(p)}$ , with signs  $\sigma_h$ , add the clauses  $(-\sigma_h X_{i(h)} \vee \sigma_{h+} X_{i(h+)})$  for  $1 \leq h \leq p$  to enforce consistency, where  $h^+ = 1 + (h \bmod p)$ . (This transformation, due to C. A. Tovey, works even in degenerate cases. For example, if  $m = 1$  and if the given clause is  $(x_1 \vee x_1 \vee \bar{x}_2)$ , the transformed clauses are  $(X_1 \vee X_2 \vee X_3)$ ,  $(\bar{X}_1 \vee X_2)$ ,  $(\bar{X}_2 \vee X_1)$ ,  $(X_3 \vee \bar{X}_3)$ .)

(b) After  $F_0 = \{\epsilon\}$ ,  $F_1 = F_0 \sqcup F_0$ ,  $F_2 = F_0 \sqcup F_1$ ,  $F_3 = F_0 \sqcup F_2$ ,  $F_4 = F_3 \sqcup F_3'$ ,  $F_5 = F_4 \sqcup F_4''$ , always putting the new variable into the four shortest possible clauses, we get  $F_5 = \{345, 234, 12\bar{3}, \bar{1}\bar{2}\bar{3}, 3'45, 2'\bar{3}'4, 1'\bar{2}'\bar{3}', \bar{1}'\bar{2}'\bar{3}', 3''4''5, 2''\bar{3}''4'', 1''\bar{2}''\bar{3}''', \bar{1}''\bar{2}''\bar{3}''', 3'''\bar{4}''\bar{5}, 2'''\bar{3}'''\bar{4}''', 1'''\bar{2}'''\bar{3}'''\}$ .

(c) If we delete  $\bar{1}\bar{2}\bar{3}$  from  $F_5$  there are 288 solutions, namely  $1 \wedge 2 \wedge 3 \wedge \bar{4} \wedge \bar{5} \wedge c' \wedge (4''? c'' \wedge \bar{3}''': c''' \wedge \bar{3}'')$ , where  $c = \bar{2} \vee \bar{3}$ .

(d) Add  $\lceil m/2 \rceil$  disjoint clones of the 15 clauses of (c) to the  $4m$  clauses of (a), giving  $m + 15\lceil m/2 \rceil$  3-clauses and  $3m$  2-clauses that are satisfiable only if all literals cloned from  $\bar{1}$ ,  $\bar{2}$ , or  $\bar{3}$  are false. Each clone provides six such false literals  $\{\bar{1}, \bar{1}, \bar{1}, \bar{2}, \bar{2}, \bar{3}\}$  without using any variable five times. So we can stick those literals into the 2-clauses, obtaining  $\approx 11.5m$  3-clauses in  $N \approx 10.5m$  variables. (The new clauses have  $288^{\lceil m/2 \rceil}$  times as many solutions as the original ones. Can the ratio  $N/m \approx 10.5$  be lowered?)

**205.** Let  $F_0 = \{\epsilon\}$ ,  $F_1 = F_0 \sqcup F_0$ ,  $F_2 = F_0 \sqcup F_1$ ,  $F_3 = F_0 \sqcup F_2$ ,  $F_4 = F_0 \sqcup F_3$ ,  $F_5 = F_1 \sqcup F_4$ ,  $F_6 = F_0 \sqcup F_5$ ,  $F_7 = F_0 \sqcup F_6$ ,  $F_8 = F_4 \sqcup F_7'$ ,  $F_9 = F_0 \sqcup F_8$ ,  $F_{10} = F_7 \sqcup F_9'$ ,

$F_{11} = F_7 \sqcup F'_{10}$ ,  $F_{12} = F_0 \sqcup F_{11}$ ,  $F_{13} = F_9 \sqcup F'_{12}$ ,  $F_{14} = F_{10} \sqcup F_{12}^{(3)}$ ,  $F_{15} = F_{12} \sqcup F_{14}^{(4)}$ ,  $F_{16} = F_{13} \sqcup F_{14}^{(6)}$ ,  $F_{17} = F_{14} \sqcup F_{15}^{(7)}$ ,  $F_{18} = F_{16} \sqcup F_{17}^{(13)}$ . (Here ‘ $x^{(3)}$ ’ stands for ‘ $x'''$ ’, etc.) Then  $F_{18}$  consists of 257 unsatisfiable 4-clauses in 234 variables.

(Is there a shorter solution? This problem was first solved by J. Střibrná in her M.S. thesis (Prague: Charles University, 1994), with 449 clauses. The  $\sqcup$  method was introduced by S. Hoory and S. Szeider, *Theoretical Computer Science* **337** (2005), 347–359, who presented an unsatisfiable 5SAT problem that uses each variable at most 7 times. It’s not known whether 7 can be decreased to 6 when every clause has size 5.)

**206.** Suppose  $F$  and  $F'$  are minimally unsatisfiable, and delete a clause of  $F \sqcup F'$  that arose from  $F'$ ; then we can satisfy  $F \sqcup F'$  with  $x$  true.

Conversely, if  $F \sqcup F'$  is minimally unsatisfiable,  $F$  and  $F'$  can’t both be satisfiable. Suppose  $F$  is unsatisfiable but  $F'$  is satisfied by  $L'$ . Removing a clause of  $F \sqcup F'$  that arose from  $F'$  is satisfiable only with  $x$  true; but then we can use  $L'$  to satisfy  $F \sqcup F'$ . Hence  $F$  and  $F'$  are both unsatisfiable. Finally, if  $F \setminus C$  is unsatisfiable, so is  $(F \sqcup F') \setminus (C \mid \bar{x})$ , because any solution would satisfy either  $F \setminus C$  or  $F'$ .

**207.** The five clauses of  $C(x, y, z; a, b, c) = \{x\bar{a}b, y\bar{b}c, z\bar{c}a, abc, a\bar{b}\bar{c}\}$  resolve to the single clause  $xyz$ . Thus  $C(x, y, y; 1, 2, 3) \cup C(x, \bar{y}, \bar{y}; 4, 5, 6) \cup C(\bar{x}, z, z; 7, 8, 9) \cup C(\bar{x}, \bar{z}, \bar{z}; a, b, c)$  is a solution. [K. Iwama and K. Takaki, *DIMACS* **35** (1997), 315–333, noted that the 16 clauses  $\{\bar{x}\bar{y}\bar{z}\} \cup C(x, x, x; 1, 2, 3) \cup C(y, y, y; 4, 5, 6) \cup C(z, z, z; 7, 8, 9)$  involve each variable exactly four times, and proved that no set of twelve clauses does so.]

**208.** Make  $m$  clones of all but one of the 20 clauses in answer 207, and put the other  $3m$  cloned literals into the  $3m$  binary clauses of answer 204(a). This gives  $23m$  3-clauses in which every literal occurs twice, except that the  $3m$  literals  $\bar{X}_i$  occur only once.

To complete the solution, we “pad” them with additional clauses that are always satisfiable. For example, we could introduce  $3m$  more variables  $u_i$ , with new clauses  $\bar{X}_i u_i \bar{u}_{i+1}$  for  $1 \leq i \leq 3m$  and  $\{u'_{3j} u'_{3j+1} u'_{3j+2}, \bar{u}'_{3j} \bar{u}'_{3j+1} \bar{u}'_{3j+2}\}$  for  $1 \leq j \leq m$  (treating subscripts mod  $3m$ ), where  $u'_i$  denotes ( $i$  even?  $u_i$ :  $\bar{u}_i$ ).

**209.** Since the multiset of  $kt$  literals in any  $t$  clauses contains at least  $t$  different variables, the “marriage theorem” (Theorem 7.5.1M) implies that we can choose a different variable in each clause, easily satisfying it. [*Discr. Applied Math.* **8** (1984), 85–89.]

**210.** [P. Berman, M. Karpinski, A. D. Scott, *Electronic Colloquium on Computational Complexity* (2003), TR22.] This answer uses the magic number  $\varepsilon = \delta^7 \approx 1/58$ , where  $\delta$  is the smallest root of  $\delta((1 - \delta^7)^6 + (1 - \delta^7)^7) = 1$ . We will assign random values to each variable so that  $\Pr[\text{all clauses are satisfied}] > 0$ .

Let  $\eta_j = (1 - \varepsilon)^j / ((1 - \varepsilon)^j + (1 - \varepsilon)^{13-j})$ , and observe that  $\eta_j \leq \delta(1 - \varepsilon)^j$  for  $0 \leq j \leq 13$ . If variable  $x$  occurs  $d^+$  times and  $\bar{x}$  occurs  $d^-$  times, let  $x$  be true with probability  $\eta_{d^-}$ , false with probability  $1 - \eta_{d^-} = \eta_{13-d^-} \leq \delta(1 - \varepsilon)^{13-d^-} \leq \delta(1 - \varepsilon)^{d^+}$ .

Let  $\text{bad}(C) = [\text{clause } C \text{ is falsified by the random assignment}]$ , and construct the lopsidedependency graph for these events as in exercise 351. Then, if the literals of  $C = (l_1 \vee \dots \vee l_7)$  have contrary appearances in  $d_1, \dots, d_7$  other clauses, we have

$$\Pr(\text{bad}(C)) \leq (\delta(1 - \varepsilon)^{d_1}) \dots (\delta(1 - \varepsilon)^{d_7}) = \varepsilon(1 - \varepsilon)^{d_1 + \dots + d_7} \leq \varepsilon(1 - \varepsilon)^{\text{degree}(C)},$$

because  $C$  has at most  $d_1 + \dots + d_7$  neighbors. Theorem L, with parameter  $\theta_i = \varepsilon$  for each event  $\text{bad}(C)$ , now tells us that  $\Pr[\text{all } m \text{ clauses are satisfied}] \geq (1 - \varepsilon)^m$ .

[See H. Gebauer, T. Szabó, and G. Tardos, *SODA* **22** (2011), 664–674, for asymptotic results that apply to  $k$ SAT as  $k \rightarrow \infty$ .]

Střibrná  
Hoory  
Szeider  
5SAT  
resolution  
Iwama  
Takaki  
marriage theorem  
Berman  
Karpinski  
Scott  
lopsidedependency graph  
Gebauer  
Szabó  
Tardos

**211.** If  $m$  clauses in  $n$  variables are given, so that  $3m = 4n$ , let  $N = 8n$ . Consider  $N$  “colors” named  $jk$  or  $\overline{jk}$ , where  $1 \leq j \leq n$  and  $k$  is one of the four clauses that contains  $\pm x_j$ . Let  $\sigma$  be a permutation on the colors, consisting of 4-cycles that involve the same variable, with the properties that (i)  $(jk)\sigma = jk'$  for some  $k'$  and (ii)  $(\overline{jk})\sigma = \overline{(jk)\sigma}$ .

There are  $4n$  vertices of  $K_N$  named  $jk$ , having the respective color lists

$$L(jk, 1) = \{jk, \overline{jk}\}, \quad L(jk, 2) = \{jk, (jk)\sigma\}, \quad L(jk, 3) = \{\overline{jk}, (jk)\sigma\}.$$

The other  $3m$  vertices of  $K_N$  are named  $a_k, b_k, c_k$  for each clause  $k$ . If that clause is, say,  $x_2 \vee \overline{x_5} \vee x_6$ , the color lists are

$$\begin{aligned} L(a_k, 1) &= \{2k, \overline{5k}, 6k\}, & L(b_k, 1) &= L(c_k, 1) = \{2k, \overline{2k}, 5k, \overline{5k}, 6k, \overline{6k}\}; \\ L(a_k, 2) &= \{(2k)\sigma\}, & L(b_k, 2) &= \{(5k)\sigma\}, & L(c_k, 2) &= \{(6k)\sigma\}; \\ L(a_k, 3) &= \{(\overline{2k})\sigma^2, (2k)\sigma\}, & L(b_k, 3) &= \{(\overline{5k})\sigma^2, (5k)\sigma\}, & L(c_k, 3) &= \{(\overline{6k})\sigma^2, (6k)\sigma\}. \end{aligned}$$

Then  $K_N \square K_3$  is list-colorable if and only if the clauses are satisfiable. (For example,  $(jk, 1)$  is colored  $jk \iff ((jk)\sigma, 1)$  is colored  $(jk)\sigma \iff (a_k, 1)$  is not colored  $jk$ .)

**212.** (a) Let  $x_{ijk} = 1$  if and only if  $X_{ij} = k$ . [Note: Another equivalent problem is to find an exact cover of the rows  $\{\{Pij, Rik, Cjk\} \mid p_{ij} = r_{ik} = c_{jk} = 1\}$ . This is a special case of 3D matching; see the discussion of sudoku in Section 7.2.2.1. Incidentally, the 3D matching problem can be formulated as the problem of finding a binary tensor  $(x_{ijk})$  such that  $x_{ijk} \leq y_{ijk}$  and  $x_{i**} = x_{**j} = x_{**k} = 1$ , given  $(y_{ijk})$ .]

(b)  $c_{31} = c_{32} = r_{13} = r_{14} = 0$  forces  $x_{13*} = 0 \neq p_{13}$  when  $r = c = \begin{pmatrix} 1100 \\ 0110 \\ 0011 \\ 1001 \end{pmatrix}$ ,  $p = \begin{pmatrix} 1010 \\ 1100 \\ 0101 \\ 0011 \end{pmatrix}$ .

(c) Make  $L(I, J) = \{1, \dots, N\}$  for  $M < I \leq N$ ,  $1 \leq J \leq N$ . It is well known (Theorem 7.5.1L) that a latin rectangle can always be extended to a latin square.

(d) Index everything by the set  $\{1, \dots, N\} \cup \bigcup_{I,J} \{(I, J, K) \mid K \in L(I, J)\}$ . The elements  $(I, J, K)$  where  $K = \min L(I, J)$  are called *headers*. Set  $p_{ij} = 1$  if and only if (i)  $i = j = (I, J, K)$  is not a header; or (ii)  $i = (I, J, K)$  is a header, and  $j = J$  or  $j = (I, J, K')$  is not a header; or (iii)  $j = (I, J, K)$  is a header, and  $i = I$  or  $i = (I, J, K')$  is not a header. Set  $r_{ik} = c_{ik} = 1$  if and only if (i)  $1 \leq i, k \leq N$ ; or (ii)  $i = (I, J, K)$  and  $k = (I, J, K')$ , and if  $i$  is not a header then  $(K' = K$  or  $K'$  is the largest element  $< K$  in  $L(I, J)$ ). [Reference: SICOMP **23** (1994), 170–184.]

**213.** The hinted probability is  $(1 - (1 - p)^{n'}(1 - q)^{n-n'})^m$ , where  $n' = b_1 + \dots + b_n$ . Thus if  $p \leq q$ , every  $x$  has probability at least  $(1 - (1 - p)^n)^m$  of satisfying every clause. This is huge, unless  $n$  is small or  $m$  is large: If  $m$  is less than  $\alpha^n$ , where  $\alpha$  is any constant less than  $1/(1 - p)$ , then when  $n > -1/\lg(1 - p)$  the probability  $(1 - (1 - p)^n)^m > \exp(\alpha^n \ln(1 - (1 - p)^n)) > \exp(-2(\alpha(1 - p))^n) > 1 - 2(\alpha(1 - p))^n$  is exponentially close to 1. Nobody needs a SAT solver for such an easy problem.

Even if, say,  $p = q = k/(2n)$ , so that the average clause size is  $k$ , a clause is empty — hence unsatisfiable — with probability  $e^{-k} + O(n^{-1})$ ; and indeed a clause has exactly  $r$  elements with the Poisson probability  $e^{-k} k^r / r! + O(n^{-1})$  for fixed  $r$ . So the model isn't very relevant. [See J. Franco, *Information Proc. Letters* **23** (1986), 103–106.]

**214.** (a)  $T(z) = ze^z + 2T(pz)(e^{(1-p)z} - 1)$ .

(b) If  $f(z) = \prod_{m=1}^{\infty} (1 - e^{(p-1)z/p^m})$  and  $\tau(z) = f(z)T(z)e^{-z}$ , we have  $\tau(z) = zf(z) + 2\tau(pz) = zf(z) + 2pzf(pz) + 4p^2zf(p^2z) + \dots$ .

(c) See P. Jacquet, C. Knessl, and W. Szpankowski, *Combinatorics, Probability, and Computing* **23** (2014), 829–841. [The sequence  $\langle T_n \rangle$  was first studied by A. T. Goldberg, *Courant Computer Science Report* **16** (1979), 48–49.]

4-cycles  
exact cover  
3D matching  
sudoku  
headers  
Poisson probability  
Franco  
Jacquet  
Knessl  
Szpankowski  
Goldberg

**215.** Since any given  $x_1 \dots x_l$  is a partial solution in  $(8\binom{n}{3} - \binom{l}{3})^m$  of the  $(8\binom{n}{3})^m$  possible cases, level  $l$  contains  $P_l = 2^l(1 - \frac{1}{8}l^2/n^2)^m$  nodes on the average. When  $m = 4n$  and  $n = 50$ , the largest levels are  $(P_{31}, P_{32}, \dots, P_{36}) \approx (6.4, 6.9, 7.2, 7.2, 6.8, 6.2) \times 10^6$ , and the mean total tree size  $P_0 + \dots + P_{50}$  is about 85.6 million.

If  $l = 2tn$  and  $m = \alpha n$  we have  $P_l = 2^{f(t)n}$ , where  $f(t) = 2t + \alpha \lg(1-t^3) + O(1/n)$  for  $0 \leq t \leq 1/2$ . The maximum  $f(t)$  occurs when  $\ln 4 = 3\alpha t^2/(1-t^3)$ , at which point  $t = t_\alpha = \beta - \frac{1}{2}\beta^4 + \frac{5}{8}\beta^7 + O(\beta^{10})$ , where  $\beta = \sqrt{\ln 4/(3\alpha)}$ ; for example,  $t_4 \approx 0.334$ . Now

$$P_{L+k} = 2^E, \quad E = (2t_\alpha + \alpha \lg(1-t_\alpha^3))n + 1 - 2t_\alpha - \frac{\gamma k^2}{n} + O\left(\frac{1}{n}\right) + O\left(\frac{k^3}{n^2}\right), \quad \gamma = \frac{\alpha + t_\alpha \ln 2}{2\alpha t_\alpha},$$

when  $L = 2t_\alpha n$ ; hence the expected total tree size is  $\sqrt{\pi n/\gamma} P_L (1 + O(1/\sqrt{n}))$ .

[This question was first studied by C. A. Brown and P. W. Purdom, Jr., *SICOMP* **10** (1981), 583-593; K. M. Buglara and C. A. Brown, *Inf. Sciences* **40** (1986), 21-37.]

**216.** If the search tree has  $q$  two-way branches, it has fewer than  $2nq$  nodes; we shall find an upper bound on  $E q$ . Consider such branches after values have been assigned to the first  $l$  variables  $x_1, \dots, x_l$ , and also to  $s$  additional variables  $y_1, \dots, y_s$  because of unit-clause forcing; the branch therefore occurs on level  $t = l + s$ . The values can be assigned in  $2^t$  ways, and the  $y$ 's can be chosen in  $\binom{n-1-l}{s}$  ways. For  $1 \leq i \leq s$  the  $m$  given clauses must contain  $j_i \geq 1$  clauses chosen (with replacement) from the  $F = \binom{t-1}{2}$  that force the value of  $y_i$  from other known values. The other  $m - j_1 - \dots - j_s$  must be chosen from the  $R = 8\binom{n}{3} - sF - \binom{t}{3} - 2\binom{t}{2}(n-t)$  remaining clauses that aren't entirely false and don't force anything further. Thus the expected number of two-way branches is at most

$$P_{lt} = 2^t \binom{n-l-1}{s} \sum_{j_1, \dots, j_s \geq 1} \binom{m}{j_1, \dots, j_s, m-j} \frac{F^j R^{m-j}}{N^m}, \quad j = j_1 + \dots + j_s, \quad N = 8\binom{n}{3},$$

summed over  $0 \leq l \leq t < n$ . Let  $b = F/N$  and  $c = R/N$ ; the sum on  $j_1, \dots, j_s$  is

$$m! [z^m] (e^{bz} - 1)^s e^{cz} = \sum_r \binom{s}{r} (-1)^{s-r} (c + rb)^m = s! c^m \sum_q \binom{m}{q} \left\{ \begin{matrix} q \\ s \end{matrix} \right\} \left( \frac{b}{c} \right)^q.$$

These values  $P_{lt}$  are almost all quite small when  $m = 200$  and  $n = 50$ , rising above 100 only when  $l \geq 45$  and  $t = 49$ ;  $\sum P_{lt} \approx 4404.7$ .

If  $l = xn$  and  $t = yn$ , we have  $b \approx \frac{3}{8}y^2/n$  and  $c \approx 1 - \frac{1}{8}(3(y-x)y^2 + y^3 + 6y^2(1-y))$ . The asymptotic value of  $[z^{\alpha n}] (e^{\beta z/n} - 1)^{\delta n} e^{\gamma z}$  can be found by the saddle point method: Let  $\zeta$  satisfy  $\beta \delta e^\zeta / (e^\zeta - 1) + \gamma = \alpha \beta / \zeta$ , and let  $\rho^2 = \alpha / \zeta^2 - \delta e^\zeta / (e^\zeta - 1)^2$ . Then the answer is approximately  $(e^\zeta - 1)^{\delta n} e^{\gamma \zeta n / \beta} \sqrt{n} / (\sqrt{2\pi} \rho \beta (\zeta n / \beta)^{\alpha n + 1})$ .

[For exact formulas and lower bounds, see *SICOMP* **12** (1983), 717-733. The total time to find all solutions grows approximately as  $(2(\frac{7}{8})^\alpha)^n$  when  $\alpha < 4.5$ , according to H.-M. Méjean, H. Morel, and G. Reynaud, *SICOMP* **24** (1995), 621-649.]

**217.** True, unless both  $l$  and  $\bar{l}$  belong to  $A$  or to  $B$  (making  $A$  or  $B$  tautological). For if  $L$  is a set of strictly distinct literals that covers both  $A$  and  $B$ , we know that neither  $A$  nor  $B$  nor  $L$  contains both  $l$  and  $\bar{l}$ ; hence  $L \setminus \{l, \bar{l}\}$  covers  $(A \setminus \{l, \bar{l}\}) \cup (B \setminus \{l, \bar{l}\}) = C$ .

(This generalization of resolution is, however, useless if  $C \supseteq A$  or  $C \supseteq B$ , because a large clause is easier to cover than any of its subsets. Thus we generally assume that  $l \in A$  and  $\bar{l} \in B$ , and that  $C$  isn't tautological, as in the text.)

**218.**  $x? B: A$ . [Hence  $(x \vee A) \wedge (\bar{x} \vee B)$  always implies  $A \vee B$ .]

Brown  
Purdom  
Buglara  
Stirling subset numbers  
asymptotic  
saddle point method  
Méjean  
Morel  
Reynaud  
tautological  
generalization of resolution

**219.** If  $C'$  or  $C''$  is tautological ( $\wp$ ), we define  $\wp \diamond C = C \diamond \wp = C$ . Otherwise, if there's a unique literal  $l$  such that  $C'$  has the form  $l \vee A'$  and  $C''$  has the form  $\bar{l} \vee A''$ , we define  $C' \diamond C'' = A' \vee A''$  as in the text. If there are two or more such literals, strictly distinct, we define  $C' \diamond C'' = \wp$ . And if there are no such literals, we define  $C' \diamond C'' = C' \vee C''$ .

[This operation is obviously commutative but not associative. For example, we have  $(\bar{x} \diamond \bar{y}) \diamond (x \vee y) = \wp$  while  $\bar{x} \diamond (\bar{y} \diamond (x \vee y)) = \epsilon$ .]

**220.** (a) True: If  $C \subseteq C'$  and  $C' \subseteq C''$  and  $C'' \neq \wp$  then  $C' \neq \wp$ ; hence every literal of  $C$  appears in  $C'$  and in  $C''$ . [The notion of subsumption goes back to a paper by Hugh McColl, *Proc. London Math. Soc.* **10** (1878), 16–28.]

(b) True: Otherwise we'd necessarily have  $(C \diamond C') \vee \alpha \vee \alpha' \neq \wp$  and  $C \neq \wp$  and  $C' \neq \wp$  and  $C \diamond C' \neq C \vee C'$ ; hence there's a literal  $l$  with  $C = l \vee A$ ,  $C' = \bar{l} \vee A'$ , and the literals of  $A \vee A' \vee \alpha \vee \alpha'$  are strictly distinct. So the result is easily checked, whether or not  $\alpha$  or  $\alpha'$  contains  $l$  or  $\bar{l}$ . (Notice that we always have  $C \diamond C' \subseteq C \vee C'$ .)

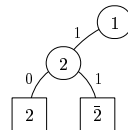
(c) False:  $\bar{x}y \subseteq \wp$  but  $x \diamond \bar{x}y = y \not\subseteq x = x \diamond \wp$ . Also  $\epsilon \subseteq \bar{x}$  but  $x \diamond \epsilon = x \not\subseteq \epsilon = x \diamond \bar{x}$ .

(d) Such examples are possible if  $C \neq \epsilon$ : We have  $x, \bar{x} \vdash y$  (and also  $x, \bar{x} \vdash \wp$ ), although the only clauses obtainable from  $x$  and  $\bar{x}$  by resolution are  $x, \bar{x}$ , and  $\epsilon$ . (On the other hand we do have  $F \vdash \epsilon$  if and only if there's a refutation chain (104) for  $F$ .)

(e) Given a resolution chain  $C'_1, \dots, C'_{m+r}$ , we can construct another chain  $C_1, \dots, C_{m+r}$  in which  $C_i \subseteq C'_i$  for  $1 \leq i \leq m+r$ . Indeed, if  $i > m$  and  $C'_i = C'_j \diamond C'_k$ , it's easy to see that either  $C_j \diamond C_k$  or  $C_k \diamond C_j$  will subsume  $C'_i$ .

(f) It suffices by (e) to prove this when  $\alpha_1 = \dots = \alpha_m = \alpha$ ; and by induction we may assume that  $\alpha = l$  is a single literal. Given a resolution chain  $C_1, \dots, C_{m+r}$  we can construct another one  $C'_1, \dots, C'_{m+r}$  such that  $C'_i = C_i \vee l$  for  $1 \leq i \leq m$  and  $C'_i \subseteq C_i \vee l$  for  $m+1 \leq i \leq m+r$ , with  $C'_i = C'_j$  or  $C'_k$  or  $C'_j \diamond C'_k$  whenever  $C_i = C_j \diamond C_k$ .

**221.** Algorithm A recognizes '1' as a pure literal, but then finds a contradiction because the *other* two clauses are unsatisfiable. The resolution refutation uses only the other two clauses. (This is an example of an unnecessary branch. Indeed, a pure literal never appears in a refutation tree, because it can't be canceled; see the next exercise.)



**222.** If  $A$  is an autarky that satisfies  $C$ , it also satisfies every clause on the path to  $\epsilon$  from a source vertex labeled  $C$ , because all of the satisfied literals cannot simultaneously vanish. For the converse, see *Discrete Appl. Math.* **107** (2000), 99–137, Theorem 3.16.

**223.** (The author has convinced himself of this statement, but he has not been able to construct a formal proof.)

**224.** At every leaf labeled by an axiom  $A$  of  $F \mid \bar{x}$  that is not an axiom of  $F$ , change the label to  $A \cup x$ ; also include  $x$  in the labels of all this leaf's ancestors. We obtain a resolution tree in which the leaves are labeled by axioms of  $F$ . The root is labeled  $x$ , if any labels have changed; otherwise it is still labeled  $\epsilon$ .

[See J. A. Robinson, *Machine Intelligence* **3** (1968), 77–94.]

**225.** Let's say that a regular resolution tree for clause  $A$  is *awkward* if at least one of its nodes resolves on one of the variables in  $A$ . An awkward tree  $T$  for  $A$  can always be transformed into a regular non-awkward tree  $T'$  for some clause  $A' \subseteq A$ , where  $T'$  is smaller than  $T$ . *Proof:* Suppose  $T$  is awkward, but none of its subtrees are. Without loss of generality we can find a sequence of subtrees  $T_0, \dots, T_p, T'_1, \dots, T'_p$ , where  $T_0 = T$  and  $T_{j-1}$  for  $1 \leq j \leq p$  is obtained from  $T_j$  and  $T'_j$  by resolving on the variable  $x_j$ ; furthermore  $x_p \in A$ . We can assume that the labels of  $T_j$  and  $T'_j$  are  $A_j$  and  $A'_j$ , where  $A_j = x_j \cup R_j$  and  $A'_j = \bar{x}_j \cup R'_j$ ; hence  $A_{j-1} = R_j \cup R'_j$ . Let  $B_p = A_p$ ; and for

tautological  
commutative  
associative  
McColl  
refutation chain  
pure literal  
unnecessary branch  
Knuth  
Robinson  
awkward



$j = p - 1, p - 2, \dots, 1$ , let  $B_j = B_{j+1}$  if  $x_j \notin B_{j+1}$ , otherwise obtain  $B_j$  by resolving  $B_{j+1}$  with  $A'_j$ . It follows by induction that  $B_j \subseteq x_p \cup A_{j-1}$ . Thus  $B_1 \subseteq x_p \cup A_0 = A$ , and we've derived  $B_1$  with a non-awkward tree smaller than  $T$ .

Kullmann  
tautological  
transitive law  
hyperresolution

Now we can prove more than was asked: If  $T$  is any resolution tree that derives clause  $A$ , and if  $A \cup B$  is any clause that contains  $A$ , there's a non-awkward regular resolution tree  $T_r$  no larger than  $T$  that derives some clause  $C \subseteq A \cup B$ . The proof is by induction on the size of  $T$ : Suppose  $A = A' \cup A''$  is obtained at the root of  $T$  by resolving the clauses  $x \cup A'$  with  $\bar{x} \cup A''$  that label the subtrees  $T'$  and  $T''$ . Find non-awkward regular trees  $T'_r$  and  $T''_r$  that derive  $C'$  and  $C''$ , where  $C' \subseteq x \cup A' \cup B$  and  $C'' \subseteq \bar{x} \cup A'' \cup B$ . If  $x \in C'$  and  $\bar{x} \in C''$ , we obtain the desired  $T_r$  by resolving  $T'_r$  and  $T''_r$  on  $x$ . Otherwise we can either let  $C = C'$  and  $T_r = T'_r$ , or  $C = C''$  and  $T_r = T''_r$ . [It's interesting to apply this construction to the highly irregular resolutions in (105).]

**226.** Initially  $\alpha$  is the root,  $C(\alpha) = \epsilon$ ,  $\|\alpha\| = N$ , and  $s = 0$ . If  $\alpha$  isn't a leaf, we have  $C(\alpha) = C(\alpha') \diamond C(\alpha'')$  where  $x \in C(\alpha')$  and  $\bar{x} \in C(\alpha'')$  for some variable  $x$ . The Prover names  $x$ , and changes  $\alpha \leftarrow \alpha'$  or  $\alpha \leftarrow \alpha''$  if the Delayer sets  $x \leftarrow 0$  or  $x \leftarrow 1$ , respectively. Otherwise  $\min(\|\alpha'\|, \|\alpha''\|) \leq \|\alpha\|/2$ , and the Prover can keep going.

**227.** The proof is by induction on the number of variables,  $n$ : If  $F$  contains the empty clause, the game is over, the Delayer has scored 0, and the root is labeled 0. Otherwise the Prover names  $x$ , and the Delayer considers the smallest possible labels  $(m, m')$  on the roots of refutations for  $F \mid x$  and  $F \mid \bar{x}$ . If  $m > m'$ , the reply  $x \leftarrow 0$  guarantees  $m$  points; and the reply  $x \leftarrow *$  is no better, because  $m' + 1 \leq m$ . If  $m < m'$ , the reply  $x \leftarrow 1$  guarantees  $m'$ ; and if  $m = m'$ , the reply  $x \leftarrow *$  guarantees  $m + 1$ . Thus an optimum Delayer can always score at least as many points as the root label of any branch of a refutation tree constructed by the Prover. Conversely, if the Prover always names an optimal  $x$ , the Delayer can't do better.

(This exercise was suggested by O. Kullmann. One can compute the optimum score "bottom up" by considering all  $3^n$  possible partial assignments as in answer 133.)

**228.** We need only assume the transitivity clauses  $T_{ijk}$  of (100) when  $i < j$  and  $k < j$ . [Notice further that  $T_{ijk}$  is tautological when  $i = j$  or  $k = j$ , thus useless for resolution.]

**229.** Using the binary-relation interpretation, these clauses say that  $j \not\prec j$ , that the transitive law " $i \prec j$  and  $j \prec k$  implies  $i \prec k$ " holds whenever  $i \leq k$  and  $j < k$ , and that every  $j$  has a successor such that  $j \prec k$ . The latter axiom combines with the finiteness of  $m$  to imply that there must be a cycle  $j_0 \prec j_1 \prec \dots \prec j_{p-1} \prec j_p = j_0$ .

Consider the *shortest* such cycle, and renumber the subscripts so that  $j_p = \max\{j_0, \dots, j_p\}$ . We cannot have  $p \geq 2$ , because (100') implies  $j_{p-2} \prec j_p$ , yielding a shorter cycle. Hence  $p = 1$ ; but that contradicts (99).

**230.** Call the axioms  $I_j, T_{ijk}$ , and  $M_{jm}$  as in the text. If  $I_{j_0}$  is omitted, let  $x_{ij} = [j = j_0]$  for all  $i$  and  $j$ . If  $T_{i_0j_0k_0}$  is omitted, let  $x_{ij} = [j \in A]$  for all  $i \notin A = \{i_0, j_0, k_0\}$ ; also let  $x_{i_0j} = [j = j_0]$ ,  $x_{j_0j} = [j = k_0]$ , and (if  $i_0 \neq k_0$ )  $x_{k_0j} = [j = i_0]$ . Finally, if  $M_{j_0m}$  is omitted, let  $x_{ij} = [p_i < p_j]$ , where  $p_1 \dots p_m = 1 \dots (j_0 - 1)(j_0 + 1) \dots mj_0$ . (The same construction shows that the clauses of answer 228 are minimally unsatisfiable.)

**231.** Since  $G_{11} = M_{1m}$ , we can assume that  $j > 1$ . Then  $G_{(j-1)j} = G_{(j-1)(j-1)} \diamond I_{j-1}$ . And if  $1 \leq i < j - 1$  we have  $G_{ij} = (\dots((G_{(j-1)j} \diamond A_{ijj}) \diamond A_{ij(j+1)}) \diamond \dots) \diamond A_{ijm}$ , where  $A_{ijk} = G_{i(j-1)} \diamond T_{i(j-1)k} = G_{ij} \vee \bar{x}_{(j-1)k}$ . These clauses make it possible to derive  $B_{ij} = (\dots((G_{ij} \diamond T_{jij}) \diamond T_{ji(j+1)}) \diamond \dots) \diamond T_{jim} = G_{jj} \vee \bar{x}_{ji}$  for  $1 \leq i < j$ , from which we obtain  $G_{jj} = (\dots((M_{jm} \diamond B_{1j}) \diamond B_{2j}) \diamond \dots) \diamond B_{(j-1)j}$ . Finally  $G_{mm} \diamond I_{mm} = \epsilon$ .

**232.** It suffices to exhibit a backtrack tree of depth  $6 \lg q + O(1)$ . By branching on at most 6 variables we can find the color-triplet  $\alpha_1$  in answer 176(c).

Cook  
Cook

Suppose we know that  $\alpha_j = \alpha$  and  $\alpha_{j+p} = \alpha'$ , where  $\alpha'$  cannot be obtained from  $\alpha$  in  $p$  steps; this is initially true with  $j = 1$ ,  $\alpha = \alpha' = \alpha_1$ , and  $p = q$ . If  $p = 1$ , a few more branches will find a contradiction. Otherwise at most 6 branches will determine  $\alpha_l$ , where  $l = j + \lfloor p/2 \rfloor$ ; and either  $\alpha_l$  will be unreachable from  $\alpha$  in  $\lfloor p/2 \rfloor$  steps, or  $\alpha'$  will be unreachable from  $\alpha_l$  in  $\lfloor p/2 \rfloor$  steps, or both. Recurse.

**233.**  $C_9 = C_4 \diamond C_8$ ,  $C_{10} = C_1 \diamond C_9$ ,  $C_{11} = C_5 \diamond C_{10}$ ,  $C_{12} = C_6 \diamond C_{10}$ ,  $C_{13} = C_7 \diamond C_{11}$ ,  $C_{14} = C_3 \diamond C_{12}$ ,  $C_{15} = C_{13} \diamond C_{14}$ ,  $C_{16} = C_2 \diamond C_{15}$ ,  $C_{17} = C_4 \diamond C_{15}$ ,  $C_{18} = C_8 \diamond C_{15}$ ,  $C_{19} = C_{12} \diamond C_{17}$ ,  $C_{20} = C_{11} \diamond C_{18}$ ,  $C_{21} = C_{16} \diamond C_{19}$ ,  $C_{22} = C_{20} \diamond C_{21}$ .

**234.** Reply  $x_{jk} \leftarrow *$  to any query that doesn't allow the Prover to violate (107). Then the Prover can violate (106) only after every hole has been queried.

**235.** Let  $C(k, A) = (\bigvee_{j=0}^k \bigvee_{a \in A} x_{ja})$ , so that  $C(0, \{1, \dots, m\}) = (x_{01} \vee \dots \vee x_{0m})$  and  $C(m, \emptyset) = \epsilon$ . The chain consists of  $k$  stages for  $k = 1, \dots, m$ , where stage  $k$  begins by deriving the clauses  $\bar{x}_{ka} \vee C(k-1, A)$  from the clauses of stage  $k-1$ , for all  $(m-k)$ -element subsets  $A$  of  $\{1, \dots, m\} \setminus a$ ; every such clause requires  $k$  resolutions with (107). Stage  $k$  concludes by deriving  $C(k, A)$  for all  $(m-k)$ -element subsets  $A$  of  $\{1, \dots, m\}$ , each using one resolution from (106) and  $k-1$  resolutions from the beginning of the stage. (See (103).) Thus stage  $k$  involves a total of  $\binom{m}{m-k} (k^2 + k)$  resolutions.

For example, the resolutions when  $m = 3$  successively yield  $\overline{11}0203$ ,  $\overline{12}0103$ ,  $\overline{13}0102$ ;  $01021112$ ,  $01031113$ ,  $02031213$  (stage 1);  $\overline{21}021112$ ,  $\overline{21}0212$ ,  $\overline{21}031113$ ,  $\overline{21}0313$ ,  $\overline{22}011211$ ,  $\overline{22}0111$ ,  $\overline{22}031213$ ,  $\overline{22}0313$ ,  $\overline{23}011311$ ,  $\overline{23}0111$ ,  $\overline{23}021312$ ,  $\overline{23}0212$ ;  $01112122$ ,  $011121$ ,  $02122223$ ,  $021222$ ,  $03132322$ ,  $031323$  (stage 2); and  $\overline{31}1121$ ,  $\overline{31}21$ ,  $\overline{31}$ ,  $\overline{32}1222$ ,  $\overline{32}22$ ,  $\overline{32}$ ,  $\overline{33}1323$ ,  $\overline{33}23$ ,  $\overline{33}$ ;  $3233$ ,  $33$ ,  $\epsilon$  (stage 3).

[Stephen A. Cook constructed such chains in 1972 (unpublished).]

**236.** The symmetry of the axioms should allow exhaustive verification by computer for  $m = 2$ , possibly also for  $m = 3$ . The construction certainly seems hard to beat. Cook conjectured in 1972 that any minimum-length resolution proof would include, for every subset  $S$  of  $\{1, \dots, m\}$ , at least one clause  $C$  such that  $\bigcup_{\pm x_{jk} \in C} \{k\} = S$ .

**237.** The idea is to define  $y_{jk} = x_{jk} \vee (x_{jm} \wedge x_{mk})$  for  $0 \leq j < m$  and  $1 \leq k < m$ , thus reducing from  $m$  pigeons to  $m-1$ . First we append  $6(m-1)(m-2)$  new clauses

$$(x_{jm} \vee z_{jk}) \wedge (x_{mk} \vee z_{jk}) \wedge (\bar{x}_{jm} \vee \bar{x}_{mk} \vee \bar{z}_{jk}) \wedge (\bar{x}_{jk} \vee y_{jk}) \wedge (y_{jk} \vee z_{jk}) \wedge (x_{jk} \vee \bar{y}_{jk} \vee \bar{z}_{jk}),$$

involving  $2(m-1)(m-2)$  new variables  $y_{jk}$  and  $z_{jk}$ . Call these clauses  $A_{jk}, \dots, F_{jk}$ .

Now if  $P_j$  stands for (106) and  $H_{ijk}$  for (107), we want to use resolution to derive  $P'_j = (y_{j1} \vee \dots \vee y_{j(m-1)})$  and  $H'_{ijk} = (\bar{y}_{ik} \vee \bar{y}_{jk})$ . First,  $P_j$  can be resolved with  $D_{j1}, \dots, D_{j(m-1)}$  to get  $P'_j \vee x_{jm}$ . Next,  $P_m \diamond H_{jmm} = x_{m1} \vee \dots \vee x_{m(m-1)} \vee \bar{x}_{jm}$  can be resolved with  $G_{jk} = C_{jk} \diamond E_{jk} = \bar{x}_{jm} \vee \bar{x}_{mk} \vee y_{jk}$  for  $1 \leq k < m$  to get  $P'_j \vee \bar{x}_{jm}$ . One more step yields  $P'_j$ . (The intuitive "meaning" guides these maneuvers.)

From  $B_{jk} \diamond F_{jk} = x_{jk} \vee x_{mk} \vee \bar{y}_{jk}$ , we obtain  $Q_{ijk} = \bar{x}_{ik} \vee \bar{y}_{jk}$  after resolving with  $H_{ijk}$  and  $H_{imk}$ . Then  $(Q_{ijk} \diamond F_{ik}) \diamond A_{ik} = x_{im} \vee \bar{y}_{ik} \vee \bar{y}_{jk} = R_{ijk}$ , say. Finally,  $(R_{jik} \diamond H_{ijm}) \diamond R_{ijk} = H'_{ijk}$  as desired. (When forming  $R_{jik}$  we need  $Q_{ijk}$  with  $j > i$ .)

We've done  $5m^3 - 6m^2 + 3m$  resolutions to reduce  $m$  to  $m-1$ . Repeating until  $m = 0$ , with fresh  $y$  and  $z$  variables each time, yields  $\epsilon$  after about  $\frac{5}{4}m^4$  steps.

[See Stephen A. Cook, *SIGACT News* 8, 4 (October 1976), 28–32.]

**238.** The function  $(1 - cx)^{-x} = \exp(cx^2 + c^2x^3/2 + \dots)$  is increasing and  $> e^{cx^2}$ . Setting  $c = \frac{1}{2n}$ ,  $W = \sqrt{2n \ln r}$ , and  $b = \lceil W \rceil$  makes  $f \leq r < \rho^{-b}$ . Also  $W \geq w(\alpha_0)$

when  $n \geq w(\alpha_0)^2$  and  $r \geq 2$ ; hence  $w(\alpha_0 \vdash \epsilon) \leq W + b \leq \sqrt{8n \ln r} + 1$  as desired. The ‘-2’ in the lemma handles the trivial cases that arise when  $r < 2$ .

(It is important to realize that we don’t change  $n$  or  $W$  in the induction proof. Incidentally, the exact minimum of  $W + b$ , subject to  $r = (1 - W/(2n))^{-b}$ , occurs when

$$W = 2n(1 - e^{-2T(z)}) = 4nz + \frac{2nz^3}{3} + \dots, \quad b = \frac{\ln r}{2T(z)} = (\ln r) \left( \frac{1}{2z} - \frac{1}{2} - \frac{z}{4} - \dots \right),$$

where  $z^2 = (\ln r)/(8n)$  and  $T(z)$  is the tree function. Thus it appears likely that the proof of Lemma B supports the stronger result  $w(\alpha_0 \vdash \epsilon) < \sqrt{8n \ln r} - \frac{1}{2} \ln r + 1$ .)

**239.** Let  $\alpha_0$  consist of all  $2^n$  nontautological clauses of length  $n$ . The shortest refutation is the complete binary tree with these leaves, because every nontautological clause must appear. Algorithm A shows that  $2^n - 1$  resolutions suffice to refute any clauses in  $n$  variables; hence  $\|\alpha_0 \vdash \epsilon\| = 2^n - 1$ , and this is the worst case.

**240.** If  $A'$  has  $t$  elements and  $\partial A'$  has fewer than  $t$ , the sequence of  $5t$  integers  $f_{ij}$  for its neighbors must include at least  $2t$  repeats of values seen earlier. (In fact there are at least  $2t + 1$  repeats, because  $2t$  would leave at least  $t$  in the boundary; but the calculations are simpler with  $2t$ , and we need only a rather crude bound.)

The probability  $p_t$  that some such  $A'$  exists is therefore less than  $\binom{m+1}{t} \binom{5t}{2t} \left(\frac{3t}{m}\right)^{2t}$ , because there are  $\binom{m+1}{t}$  ways to select  $A'$ ,  $\binom{5t}{2t}$  to select the repeating slots, and at most  $(3t)^{2t}$  out of  $m^{2t}$  ways to fill those slots. Also  $\binom{m+1}{t} = \binom{m}{t} + \binom{m}{t-1} < 2\binom{m}{t}$  when  $t \leq \frac{1}{2}m$ .

By exercise 1.2.6-67 we have  $p_t \leq 2\left(\frac{m\epsilon}{t}\right)^t \left(\frac{5t\epsilon}{2t}\right)^{2t} \left(\frac{3t}{m}\right)^{2t} = 2(ct/m)^t$ , where  $c = 225e^3/4 \approx 1130$ . Also  $p_0 = p_1 = 0$ . Thus the sum of  $p_t$  for  $t \leq m/3000$  is less than  $2\sum_{t=2}^{\infty} (c/3000)^t \approx .455$ ; and the probability of strong expansion exceeds .544.

**241.** If  $0 < |A'| \leq m/3000$ , we can put one of its elements into a hole  $b_k \in \partial A'$ . Then we can place the other elements in the same way, since  $b_k$  isn’t their neighbor.

**242.** The proof of Theorem B remains valid when these new axioms are added.

**243.** (a) The probability that  $F'$  has  $t$  elements and  $V(F')$  has fewer than  $t$  is at most  $\binom{\alpha n}{t} \binom{n}{t} \left(\frac{t}{n}\right)^{3t} \leq \left(\frac{\alpha e^2 t}{n}\right)^t$ . The sum of this quantity for  $1 \leq t \leq \lg n$  is  $O(n^{-1})$ , and so is the sum for  $\lg n \leq t \leq n/(2\alpha e^2)$ .

(b) If the condition in (a) holds, there’s a matching from  $F'$  into  $V(F')$ , by Theorem 7.5.1M; hence we can satisfy  $F'$  by assigning to its variables, one by one. If  $F$  is unsatisfiable we’ll therefore need to invoke more than  $n/(2\alpha e^2)$  of its axioms.



(c) The probability  $p_t$  that  $F'$  has  $t$  elements and  $2|V(F')| - 3|F'| < \frac{1}{2}|F'|$  is at most  $\binom{\alpha n}{t} \binom{n}{\lambda t} \left(\frac{\lambda t}{n}\right)^{3t} \leq (\alpha e^{1+\lambda} \lambda^{3-\lambda} (t/n)^{1/4})^t$ , where  $\lambda = \frac{7}{4}$ . We have  $(e^{1+\lambda} \lambda^{3-\lambda})^4 < 10^6$ ; so  $p_t < c^t$  when  $t \leq n'$ , where  $c < 1$ , and  $\sum_{t=n'/2}^{n'} p_t$  is exponentially small.

(d) Since  $n' < n/(2\alpha e^2)$ , every refutation a.s. contains a clause  $C$  with  $n'/2 \leq \mu(C) < n'$ . The minimal axioms  $F'$  on which  $C$  depends have  $|F'| = \mu(C)$ . Let  $k$  be the number of “boundary” variables that occur in just one axiom of  $F'$ . If  $v$  is such a variable, we can falsify  $C$  and the axiom containing  $v$ , while the other axioms of  $F'$  are true; hence  $V$  must contain  $v$  or  $\bar{v}$ . We have  $|V(F')| = k + |\text{nonboundary}| \leq k + \frac{1}{2}(3|F'| - k)$ , because each nonboundary variable occurs at least twice. Therefore  $k \geq 2|V(F')| - 3|F'| \geq n'/4$ , q.s. (Notice the similarities to the proof of Theorem B.)

**244.** We have  $[A \cup B]^0 = [A]^0[B]^0 \cup [A]^1[B]^1$  and  $[A \cup B]^1 = [A]^0[B]^1 \cup [A]^1[B]^0$ , where concatenation of sets has the obvious meaning. These relations hold also when  $A = \emptyset$  or  $B = \emptyset$ , because  $[\emptyset]^0 = \{\epsilon\}$  and  $[\emptyset]^1 = \emptyset$ .

asymptotic methods  
generating functions  
tree function  
complete binary tree  
matching  
boundary

**245.** (a) When conditioning on  $e_{uv}$ , simply delete the edge  $u - v$  from  $G$ . When conditioning on  $\bar{e}_{uv}$ , also complement  $l(u)$  and  $l(v)$ . The graph might become disconnected; in that case, there will be exactly two components, one even and one odd, with respect to the sums of their labels. The axioms for the even component are satisfiable and may be discarded.

For example,  $\alpha(G) \mid \{\bar{b}, e\}$  corresponds to  while  $\alpha(G) \mid \{\bar{b}, \bar{e}\}$  corresponds to . We toss out the left component in the first case, the right one in the other.

(b) If  $C \in \alpha(v)$  we may take  $V' = \{v\}$ . And we have  $\mu(\epsilon) = |V|$ , because the axioms  $\bigcup_{v \in V \setminus u} \alpha(v)$  are satisfiable for all  $u \in V$ .

(c) If  $u \in V'$  and  $v \notin V'$ , there's an assignment that falsifies  $C$  and some axiom of  $\alpha(u)$  while satisfying all  $\alpha(w)$  for  $w \in V' \setminus u$ , because  $|V'|$  is minimum. Setting  $e_{uv} \leftarrow \bar{e}_{uv}$  will satisfy  $\alpha(u)$  without affecting the axioms  $\alpha(w)$  (which don't contain  $e_{uv}$ ).

(d) By (b), every refutation of  $\alpha(G)$  must contain a clause  $C$  with  $\frac{1}{3}m \leq \mu(C) < \frac{2}{3}m$ . The corresponding  $V'$  has  $|V'|/(|V'| + |\partial V'|) < (\frac{2}{3} + 8)/9$ , hence  $|\partial V'| > \frac{1}{26}|V'|$ .

[Property (i) is interesting but irrelevant for this proof. Notice that  $\alpha(G)$  has exactly  $\frac{8}{3}n \approx 2.67n$  3SAT clauses in  $n = 3m/2$  variables when  $G$  is cubic; every literal occurs four times. G. Tseytin proved lower bounds for refutations of  $\alpha(G)$  by regular resolution in 1966, before graphs with property (iii) were known; A. Urquhart obtained them for general resolution in *JACM* **34** (1987), 209–219, and the simplified argument given here is due to Ben-Sasson and Wigderson. The fact that  $\alpha(G)$  requires exponentially long refutation chains, although the same axioms can be refuted easily by working with linear equations mod 2, amounts to a proof that backtracking is a poor way to deal with linear equations! Suitable Ramanujan graphs  $raman(2, q, 3, 0)$  are part of the Stanford GraphBase for infinitely many prime numbers  $q$ . We can also obtain the same lower bounds with the multigraphs  $raman(2, q, 1, 0)$  and  $raman(2, q, 2, 0)$ . Section 7.4.3 will explore expander graphs in detail.]

**246.** Let's write  $[a_1 \dots a_k]^\ell$  for what exercise 244 calls  $[\{a_1, \dots, a_k\}]^\ell$ . With new variables  $x, y, z$  we can introduce  $\{xa, x\bar{b}, \bar{x}ab, y\bar{a}, yb, y\bar{a}\bar{b}, zx, zy, z\bar{x}\bar{y}\}$  and resolve those clauses to  $[zab]^\ell$ , which means  $z = a \oplus b$ . So we can assume that ' $z \leftarrow a \oplus b$ ' is a legal primitive operation of "extended resolution hardware," when  $z$  is a new variable. Furthermore we can compute  $a_1 \oplus \dots \oplus a_k$  in  $O(k)$  steps, using  $z_0 \leftarrow 0$  (which is the clause  $[z_0]^\ell$ , namely  $\bar{z}_0$ ) and  $z_k \leftarrow z_{k-1} \oplus a_k$  when  $k \geq 1$ .

Let the edge variables  $E(v)$  be  $a_1, \dots, a_d$ , where  $d$  is the degree of  $v$ . We compute  $s_v \leftarrow a_1 \oplus \dots \oplus a_d$  by setting  $s_{v,0} \leftarrow 0$ ,  $s_{v,k} \leftarrow s_{v,k-1} \oplus a_k$ , and  $s_v \leftarrow s_{v,d}$ . We can resolve  $s_v$  with the axioms  $\alpha(v)$  in  $O(2^d)$  steps, to get the singleton clause  $[s_v]^{\ell(v) \oplus 1}$ , meaning  $s_v = \ell(v)$ . Summing over  $v$ , these operations therefore take  $O(N)$  steps.

On the other hand, we can also compute  $z_n \leftarrow \bigoplus_v s_v$  and get zero (namely ' $\bar{z}_n$ '). Doing this cleverly, by omnisciently knowing  $G$ , we can in fact compute it in  $O(mn)$  steps: Start with any vertex  $v$  and set  $z_1 \leftarrow s_v$  (more precisely, set  $z_{1,k} \leftarrow s_{v,k}$  for  $0 \leq v \leq d$ ). Given  $z_j$  for  $1 \leq j < n$ , with all its subvariables  $z_{j,k}$ , we then compute  $z_{j+1} \leftarrow z_j \oplus s_u$ , where  $u$  is the unused vertex with  $s_{u,1} = z_{j,1}$ . We can arrange the edges into an order so that if  $z_j$  has  $p$  edge variables in common with  $s_u$ , then  $z_{j,k} = s_{u,k}$  for  $1 \leq k \leq p$ . Suppose the other variables of  $z_j$  and  $s_u$  are respectively  $a_1, \dots, a_q$  and  $b_1, \dots, b_r$ ; we want to merge them into the sequence  $c_1, \dots, c_{q+r}$  that will be needed later when  $z_{j+1}$  is used. So we set  $z_{j+1,0} \leftarrow 0$ ,  $z_{j+1,k} \leftarrow z_{j+1,k-1} \oplus c_k$ ,  $z_{j+1} \leftarrow z_{j+1,q+r}$ .

From the clauses constructed in the previous paragraph, resolution can deduce  $[z_{j,k} s_{u,k}]^\ell$  for  $1 \leq k \leq p$ , and hence  $[z_{j+1,0} z_{j,p} s_{u,p}]^\ell$  (namely that  $z_{j+1,0} = z_{j,p} \oplus s_{u,p}$ ). Furthermore, if  $c_k = a_i$ , and if we know that  $z_{j+1,k-1} = z_{j,s} \oplus s_{u,t}$  where  $s = p + i - 1$

3SAT  
cubic  
Tseytin  
regular resolution  
Urquhart  
Ben-Sasson  
Wigderson  
linear equations  
backtracking  
Stanford GraphBase  
*raman*  
multigraphs  
expander graphs  
merge

and  $t = p + k - i$ , resolution can deduce that  $z_{j+1,k} = z_{j,s+1} \oplus s_{u,t}$ ; a similar formula applies when  $c_k = b_i$ . Thus resolution yields  $z_{j+1} \leftarrow z_j \oplus s_u$  as desired. Ultimately we deduce both  $z_n$  and  $\bar{z}_n$  from the singleton clauses  $s_v = \ell(v)$ .

Stålmarck  
redundant

**247.** Eliminating  $x_2$  from  $\{12, \bar{1}2, \bar{1}\bar{2}\}$  gives  $\{\bar{1}\}$ ; eliminating  $x_1$  then gives  $\emptyset$ . So those five clauses are satisfiable.

**248.** We have  $F(x_1, \dots, x_n) = (x_n \vee A'_1) \wedge \dots \wedge (x_n \vee A'_p) \wedge (\bar{x}_n \vee A''_1) \wedge \dots \wedge (\bar{x}_n \vee A''_q) \wedge A'''_1 \wedge \dots \wedge A'''_r = (x_n \vee G') \wedge (\bar{x}_n \vee G'') \wedge G'''$ , where  $G' = A'_1 \wedge \dots \wedge A'_p$ ,  $G'' = A''_1 \wedge \dots \wedge A''_q$ , and  $G''' = A'''_1 \wedge \dots \wedge A'''_r$  depend only on  $\{x_1, \dots, x_{n-1}\}$ . Hence  $F' = (G' \vee G'') \wedge G'''$ ; and the clauses of  $G' \vee G'' = \bigwedge_{i=1}^p \bigwedge_{j=1}^q (A'_i \vee A''_j)$  are the resolvents eliminating  $x_n$ .

**249.** After learning  $C_7 = \bar{2}\bar{3}$  as in the text, we set  $d \leftarrow 2$ ,  $l_2 \leftarrow \bar{2}$ ,  $C_j = \bar{2}\bar{3}$ , learn  $C_8 = \bar{3}$ , and set  $d \leftarrow 1$ ,  $l_1 \leftarrow \bar{3}$ . Then  $l_2 \leftarrow \bar{4}$  (say); and  $l_3 \leftarrow \bar{1}$ ,  $l_4 \leftarrow \bar{2}$ . Now  $C_i = 1234$  has been falsified; after  $l_4 \leftarrow 2$  and  $C_j = \bar{1}\bar{2}$  we learn  $C_9 = 134$ , set  $l_3 \leftarrow 1$ , and learn  $C_{10} = 134 \diamond \bar{1}3 = 34$ . Finally  $l_2 \leftarrow 4$ , we learn  $C_{11} = 3$ ;  $l_1 \leftarrow 3$ , and we learn  $C_{12} = \epsilon$ .

**250.**  $l_1 \leftarrow 1$ ,  $l_2 \leftarrow 3$ ,  $l_3 \leftarrow \bar{2}$ ,  $l_4 \leftarrow 4$ ; learn  $\bar{1}2\bar{3}$ ;  $l_3 \leftarrow 2$ ,  $l_4 \leftarrow 4$ ; learn  $\bar{1}\bar{2}\bar{3}$  and  $\bar{1}\bar{3}$ ;  $l_2 \leftarrow \bar{3}$ ,  $l_3 \leftarrow \bar{2}$ ,  $l_4 \leftarrow 4$ ; learn  $\bar{1}23$ ;  $l_3 \leftarrow 2$ ,  $l_4 \leftarrow 4$ ; learn  $\bar{1}\bar{2}3$ ,  $\bar{1}3$ ,  $\bar{1}$ ;  $l_1 \leftarrow \bar{1}$ ,  $l_2 \leftarrow 3$ ,  $l_3 \leftarrow \bar{4}$ ,  $l_4 \leftarrow 2$ ; learn  $\bar{1}\bar{3}4$ ;  $l_3 \leftarrow 4$ ,  $l_4 \leftarrow \bar{2}$ ,  $l_4 \leftarrow 2$ .

**251.** Algorithm I has the property that  $\bar{l}_{i_1}, \dots, \bar{l}_{i_{k-1}}, l_{i_k}$  are on the stack whenever the new clause  $l_{i_1} \vee \dots \vee l_{i_k}$  has been learned, if  $i_1 < \dots < i_k = d$  and step I4 returns to I2. These literals limit our ability to exploit the new clause; so it appears to be impossible to solve this problem without doing more resolutions than Stålmarck did.

However, we can proceed as follows. Let  $M''_{i_m k}$  be the clause  $x_{m1} \vee \dots \vee x_{m(k-1)} \vee x_{ik} \vee \dots \vee x_{i(m-1)} \vee \bar{x}_{im}$ , for  $1 \leq i, k < m$ . Using  $\overline{ij}$  to stand for  $x_{ij}$ , the process for  $m = 3$  begins by putting  $\bar{1}\bar{1}$ ,  $\bar{1}\bar{2}$ ,  $13$ ,  $\bar{2}\bar{1}$ ,  $\bar{2}\bar{2}$ ,  $23$ ,  $\bar{3}\bar{1}$ ,  $\bar{3}\bar{2}$ ,  $33$  on the stack. Then step I3 has  $C_i = I_3$ , step I4 has  $C_j = M_{33}$ ; so step I5 learns  $I_3 \diamond M_{33} = M_{32}$ . Step I4 now changes  $\bar{3}\bar{2}$  to  $32$  and chooses  $C_j = T_{232}$ ; so I5 learns  $M_{32} \diamond T_{232} = M''_{232}$ . Step I4 changes  $\bar{3}\bar{1}$  to  $31$  and chooses  $C_j = T_{231}$ ; now we learn  $M''_{232} \diamond T_{231} = M''_{231}$ . Next, we learn  $M''_{231} \diamond M_{23} = M_{22}$ ; and after changing  $\bar{2}\bar{2}$  to  $22$  we also learn  $M_{21}$ .

The stack now contains  $\bar{1}\bar{1}$ ,  $\bar{1}\bar{2}$ ,  $13$ ,  $21$ . We add  $\bar{3}\bar{1}$ ,  $\bar{3}\bar{2}$ , and proceed to learn  $M_{32} \diamond T_{132} = M'_{132}$ ,  $M'_{132} \diamond T_{131} = M''_{131}$ ,  $M'_{131} \diamond M_{13} = M_{12}$ . The stack now contains  $\bar{1}\bar{1}$ ,  $12$ , and we've essentially reduced  $m$  from 3 to 2.

In a similar way,  $O(m^2)$  resolutions will learn  $M_{i(m-1)}$  for  $i = m - 1, \dots, 1$ ; and they'll leave  $\bar{x}_{11}, \dots, \bar{x}_{1(m-2)}, x_{1(m-1)}$  on the stack so that the process can continue.

**252.** No; large numbers of clauses such as  $\bar{x}_{12} \vee \bar{x}_{23} \vee \dots \vee \bar{x}_{89} \vee x_{19}$  are generated by the elimination process. Although these clauses are valid, they're not really helpful.

Exercise 373 proves, however, that the proof *is* completed in polynomial time if we restrict consideration to the transitivity clauses of exercise 228(!).

**253.** A conflict arises when we follow a chain of forced moves:

| $t$ | $L_t$     | level | reason                  | $t$ | $L_t$     | level | reason                  |
|-----|-----------|-------|-------------------------|-----|-----------|-------|-------------------------|
| 0   | $\bar{6}$ | 1     | $\Lambda$               | 5   | $\bar{7}$ | 2     | $\bar{5}\bar{7}\bar{9}$ |
| 1   | 4         | 1     | 46                      | 6   | $\bar{1}$ | 2     | $\bar{1}\bar{5}\bar{9}$ |
| 2   | 5         | 2     | $\Lambda$               | 7   | 8         | 2     | 678                     |
| 3   | $\bar{3}$ | 2     | $\bar{3}\bar{4}\bar{5}$ | 8   | 2         | 2     | 123                     |
| 4   | 9         | 2     | 369                     | 9   | $\bar{2}$ | 2     | $\bar{2}\bar{5}\bar{8}$ |

Now  $\bar{2}\bar{5}\bar{8} \rightarrow \bar{2}\bar{5}\bar{8} \diamond 123 = 13\bar{5}\bar{8} \rightarrow 13\bar{5}67 \rightarrow 3\bar{5}67\bar{9} \rightarrow 3\bar{5}6\bar{9} \rightarrow 3\bar{5}6 \rightarrow \bar{4}\bar{5}6$ ; so we learn  $\bar{4}\bar{5}6$  (which can be simplified to  $\bar{5}6$ , because  $\bar{4}$  is "redundant" as explained in exercise 257).

Setting  $L_2 \leftarrow \bar{5}$ , with reason  $\bar{4}\bar{5}\bar{6}$  or  $\bar{5}\bar{6}$ , now forces  $7, \bar{1}, 3, 9, \bar{2}, \bar{8}, 8$ , all at level 1; this conflict soon allows us to learn the *unit* clause 6. (Next we'll inaugurate level 0, setting  $L_0 \leftarrow 6$ . No "reasons" need to be given at level 0.)

**254.** Deducing  $3, 2, 4, \bar{4}$  at level 1, it will find  $\bar{2}\bar{4} \diamond 4\bar{3} = \bar{2}\bar{3}$  and  $\bar{2}\bar{3} \diamond 2\bar{3} = \bar{3}$ , learning  $\bar{3}$ . (Or it might learn  $\bar{3}$  after deducing  $\bar{2}$ .) Then it will deduce  $3, \bar{1}, 2, 4$  at level 0.

**255.** For example,  $\{\bar{1}\bar{2}\bar{4}, \bar{2}\bar{3}\bar{5}, 456, 45\bar{6}\}$ . [Since the clause  $c'$  that is learned by the procedure described in the text contains just one literal  $l$  from the conflict level  $d$ , the trail position for  $\bar{l}$  has been called a "unique implication point" (UIP). If  $l$  isn't the decision literal for its level, we could resolve  $c'$  with  $l$ 's reason and find another UIP; but each new resolution potentially increases the  $b$  array and limits the amount of backjumping. Therefore we stop at the first UIP.]

**256.** If it is false, literals  $50, 26, \dots, 30$  are true; hence also  $\bar{2}\bar{5}, 23$ , and  $29$ , a conflict. Consequently we can obtain '\*\*' by starting with  $\bar{2}\bar{3}\bar{2}\bar{6} \dots \bar{5}\bar{0}$  and resolving with  $23\ 25\ 27, 25\ 27\ 29$ , and  $\bar{2}\bar{5}\bar{3}\bar{0} \dots \bar{7}\bar{0}$ . [Similarly, and more simply, one can learn (122) by resolving  $\bar{1}\bar{1}\bar{1}\bar{6} \dots \bar{5}\bar{6}$  with  $31\ 61\ 91, 41\ 66\ 91$ , and  $56\ 61\ 66$ .]

**257.** (a) Suppose  $\bar{l}'$  on level  $d' > 0$  is redundant. Then some  $l''$  in the reason for  $l'$  is also on level  $d'$ ; and  $l''$  is either in  $c$  or redundant. Use induction on trail position.

(b) We can assume that the stamp value  $s$  used when resolving conflicts is a multiple of 3, and that all stamps are  $\leq s$ . Then we can stamp literal  $l$  with  $S(|l|) \leftarrow s + 1$  if  $\bar{l}$  is known to be redundant, or  $s + 2$  if  $\bar{l}$  is known to be nonredundant and not in  $c$ . (These stamps serve as a "memo cache" to avoid repeated work.) While building  $c$  we can also stamp *levels* as well as literals, setting  $LS[d'] \leftarrow s$  if level  $d'$  has exactly one of the  $b_i$ , or  $s + 1$  if it has more than one.

Then for  $1 \leq j \leq r$ ,  $\bar{b}_j$  is redundant if and only if  $LS[lev(b_j)] = s + 1$  and  $red(\bar{b}_j)$  is true, where  $lev(l) = VAL(|l|) \gg 1$  and where  $red(l)$  is the following recursive procedure: "If  $l$  is a decision literal, return false. Otherwise let  $(l \vee \bar{a}_1 \vee \dots \vee \bar{a}_k)$  be  $l$ 's reason. For  $1 \leq i \leq k$  with  $lev(a_i) > 0$ , if  $S(|a_i|) = s + 2$  return false; if  $S(|a_i|) < s$  and either  $LS[lev(a_i)] < s$  or  $red(\bar{a}_i)$  is false, set  $S(|a_i|) \leftarrow s + 2$  and return false. But if none of these conditions hold, set  $S(|l|) \leftarrow s + 1$  and return true."

[See Allen Van Gelder, *LNCS 5584* (2009), 141–146.]

**258.** That statement is true in Table 3, but false in general. Indeed, consider the sequel to Table 3: The decision  $L_{44} = \bar{5}\bar{7}$  causes the watch list of 57 to be examined, thus forcing  $15, 78$ , and  $87$  (among other literals) in some order because of the clauses  $15\ 57\ 36, 78\ 57\ 36, 87\ 57\ 27$ . Then  $\bar{9}\bar{6}$  will be forced by the clause  $\bar{9}\bar{6}\bar{8}\bar{7} \dots \bar{1}\bar{5}$ ; and the second literal of that clause at the time of forcing will be  $\bar{1}\bar{5}$ , regardless of trail order, if the watched literals of that clause were  $\bar{9}\bar{6}$  and  $\bar{1}\bar{5}$  (making it invisible to  $\bar{7}\bar{8}$  and  $\bar{8}\bar{7}$ ).

**259.**  $1 + \rho^6 + \rho^7 < \rho + \rho^2$  when  $.7245 < \rho < .7548$ . (There can in fact be any number of crossover points: Consider the polynomial  $(1 - \rho - \rho^2)(1 - \rho^3 - \rho^6)(1 - \rho^9 - \rho^{18})$ .)

**260.** First, to get a random permutation in the heap we can use a variant of Algorithm 3.4.2P: For  $k \leftarrow 1, 2, \dots, n$ , let  $j$  be a random integer in  $[0..k - 1]$  and set  $HEAP[k - 1] \leftarrow HEAP[j]$ ,  $HEAP[j] \leftarrow k$ . Then set  $HLOC(HEAP[j]) \leftarrow j$  for  $0 \leq j < n$ .

Next, set  $F \leftarrow 0$  and  $W_l \leftarrow 0$  for  $2 \leq l \leq 2n + 1$  and  $c \leftarrow 3$ . Do the following for each input clause  $l_0 l_1 \dots l_{k-1}$ : Terminate unsuccessfully if  $k = 0$ , or if  $k = 1$  and  $0 \leq VAL(|l_0|) \neq l_0 \& 1$ . If  $k = 1$  and  $VAL(|l_0|) < 0$ , set  $VAL(|l_0|) \leftarrow l_0 \& 1$ ,  $TLOC(|l_0|) \leftarrow F$ ,  $F \leftarrow F + 1$ . If  $k > 1$ , set  $MEM[c + j] \leftarrow l_j$  for  $0 \leq j < k$ ; also  $MEM[c - 1] \leftarrow k$ ,  $MEM[c - 2] \leftarrow W_{l_0}$ ,  $W_{l_0} \leftarrow c$ ,  $MEM[c - 3] \leftarrow W_{l_1}$ ,  $W_{l_1} \leftarrow c$ ,  $c \leftarrow c + k + 3$ .

level 0  
reasons  
unique implication point  
UIP  
backjumping  
memo cache  
memoization technique  
*lev*  
recursive procedure  
Van Gelder  
watched literals  
random permutation

Finally, set  $\text{MINL} \leftarrow \text{MAXL} \leftarrow c+2$  (allowing two cells for extra data in the preamble of the first learned clause). Of course we must also ensure that  $\text{MEM}$  is large enough.

**261.** (Throughout this answer,  $l_j$  is an abbreviation for  $\text{MEM}[c+j]$ .) Set  $q \leftarrow 0$  and  $c \leftarrow W_{\bar{l}}$ . While  $c \neq 0$ , do the following: Set  $l' \leftarrow l_0$ . If  $l' \neq \bar{l}$  (hence  $l_1 = \bar{l}$ ), set  $c' \leftarrow l_{-3}$ ; otherwise set  $l' \leftarrow l_1$ ,  $l_0 \leftarrow l'$ ,  $l_1 \leftarrow \bar{l}$ ,  $c' \leftarrow l_{-2}$ ,  $l_{-2} \leftarrow l_{-3}$ , and  $l_{-3} \leftarrow c'$ . If  $\text{VAL}(|l_0|) \geq 0$  and  $\text{VAL}(|l_0|) + l_0$  is even (that is, if  $l_0$  is true), perform the steps

if  $q \neq 0$ , set  $\text{MEM}[q-3] \leftarrow c$ , else set  $W_{\bar{l}} \leftarrow c$ ; then set  $q \leftarrow c$ . (\*)

Otherwise set  $j \leftarrow 2$ ; while  $j < l_{-1}$  and  $\text{VAL}(|l_j|) \geq 0$  and  $\text{VAL}(|l_j|) + l_j$  is odd, set  $j \leftarrow j+1$ . If now  $j < l_{-1}$ , set  $l_1 \leftarrow l_j$ ,  $l_j \leftarrow \bar{l}$ ,  $l_{-3} \leftarrow W_{l_1}$ ,  $W_{l_1} \leftarrow c$ . But if  $j = l_{-1}$ , do (\*) above; jump to C7 if  $\text{VAL}(|l_0|) \geq 0$ ; otherwise set  $L_F \leftarrow l_0$ , etc. (see step C4) and  $c \leftarrow c'$ .

Finally, when  $c = 0$ , do (\*) above to terminate  $\bar{l}$ 's new watch list.

**262.** To delete  $k = \text{HEAP}[0]$  in C6: Set  $h \leftarrow h-1$  and  $\text{HLOC}(k) \leftarrow -1$ . Stop if  $h = 0$ . Otherwise set  $i \leftarrow \text{HEAP}[h]$ ,  $\alpha \leftarrow \text{ACT}(i)$ ,  $j \leftarrow 0$ ,  $j' \leftarrow 1$ , and do the following while  $j' < h$ : Set  $\alpha' \leftarrow \text{ACT}(\text{HEAP}[j'])$ ; if  $j'+1 < h$  and  $\text{ACT}(\text{HEAP}[j'+1]) > \alpha'$ , set  $j' \leftarrow j'+1$  and  $\alpha' \leftarrow \text{ACT}(\text{HEAP}[j'])$ ; if  $\alpha \geq \alpha'$ , set  $j' \leftarrow h$ , otherwise set  $\text{HEAP}[j] \leftarrow \text{HEAP}[j']$ ,  $\text{HLOC}(\text{HEAP}[j']) \leftarrow j$ ,  $j \leftarrow j'$ , and  $j' \leftarrow 2j+1$ . Then set  $\text{HEAP}[j] \leftarrow i$  and  $\text{HLOC}(i) \leftarrow j$ .

In C7, set  $k \leftarrow |l|$ ,  $\alpha \leftarrow \text{ACT}(k)$ ,  $\text{ACT}(k) \leftarrow \alpha + \text{DEL}$ ,  $j \leftarrow \text{HLOC}(k)$ , and if  $j > 0$  perform the “siftup” operation: “Looping repeatedly, set  $j' \leftarrow (j-1) \gg 1$  and  $i \leftarrow \text{HEAP}[j']$ , exit if  $\text{ACT}(i) \geq \alpha$ , else set  $\text{HEAP}[j] \leftarrow i$ ,  $\text{HLOC}(i) \leftarrow j$ ,  $j \leftarrow j'$ , and exit if  $j = 0$ . Then set  $\text{HEAP}[j] \leftarrow k$  and  $\text{HLOC}(k) \leftarrow j$ .”

To insert  $k$  in C8, set  $\alpha \leftarrow \text{ACT}(k)$ ,  $j \leftarrow h$ ,  $h \leftarrow h+1$ ; if  $j = 0$  set  $\text{HEAP}[0] \leftarrow k$  and  $\text{HLOC}(k) \leftarrow 0$ ; otherwise perform the siftup operation.

**263.** (This answer also sets the level stamps  $\text{LS}[d]$  needed in answer 257, assuming that the  $\text{LS}$  array is initially zero.) Let “bump  $l$ ” mean “increase  $\text{ACT}(|l|)$  by  $\text{DEL}$ ” as in answer 262. Also let  $\text{blit}(l)$  be the following subroutine: “If  $\text{S}(|l|) = s$ , do nothing. Otherwise set  $\text{S}(|l|) \leftarrow s$ ,  $p \leftarrow \text{lev}(l)$ . If  $p > 0$ , bump  $l$ ; then if  $p = d$ , set  $q \leftarrow q+1$ ; else set  $r \leftarrow r+1$ ,  $b_r \leftarrow \bar{l}$ ,  $d' \leftarrow \max(d', p)$ , and if  $\text{LS}[p] \leq s$  set  $\text{LS}[p] \leftarrow s + [\text{LS}[p] = s]$ .”

When step C7 is entered from C4, assuming that  $d > 0$ , set  $d' \leftarrow q \leftarrow r \leftarrow 0$ ,  $s \leftarrow s+3$ ,  $\text{S}(|l_0|) \leftarrow s$ , bump  $l_0$ , and do  $\text{blit}(l_j)$  for  $1 \leq j < k$ . Also set  $t \leftarrow \max(\text{TLOC}(|l_1|), \dots, \text{TLOC}(|l_{k-1}|))$ . Then, while  $q > 0$ , set  $l \leftarrow L_t$ ,  $t \leftarrow t-1$ ; if  $\text{S}(|l|) = s$  then set  $q \leftarrow q-1$ , and if  $R_l \neq \Lambda$  let clause  $R_l$  be  $l_0 l_1 \dots l_{k-1}$  and do  $\text{blit}(l_j)$  for  $1 \leq j < k$ . Finally set  $l' \leftarrow L_t$ , and while  $\text{S}(|l'|) \neq s$  set  $t \leftarrow t-1$  and  $l' \leftarrow L_t$ .

The new clause can now be checked for redundancies as in answer 257. To install it during step C9, there's a subtle point: We must watch a literal that was defined on level  $d'$ . Thus we set  $c \leftarrow \text{MAXL}$ ,  $\text{MEM}[c] \leftarrow \bar{l}'$ ,  $k \leftarrow 0$ ,  $j' \leftarrow 1$ ; and for  $1 \leq j \leq r$  if  $\text{S}(|b_j|) = s$  set  $k \leftarrow k+1$  and do this: If  $j' = 0$  or  $\text{lev}(|b_j|) < d'$ , set  $\text{MEM}[c+k+j'] \leftarrow \bar{b}_j$ , otherwise set  $\text{MEM}[c+1] \leftarrow \bar{b}_j$ ,  $j' \leftarrow 0$ ,  $\text{MEM}[c-2] \leftarrow W_{\bar{l}'}$ ,  $W_{\bar{l}'} \leftarrow c$ ,  $\text{MEM}[c-3] \leftarrow W_{\bar{b}_j}$ ,  $W_{\bar{b}_j} \leftarrow c$ . Finally set  $\text{MEM}[c-1] \leftarrow k+1$ ,  $\text{MAXL} \leftarrow c+k+6$ .

**264.** We can maintain a “history code” array, setting  $H_F$  to 0, 2, 4, or 6 when  $L_F$  is set, and then using  $H_t + (L_t \& 1)$  as the move code that represents trail location  $t$  for  $0 \leq t < F$ . History codes 6, 4, and 0 are appropriate in steps C1, C4, and C6, respectively; in C9, use code 2 if  $l'$  was a decision literal, otherwise use code 6.

[These move codes do *not* increase lexicographically when the trail is flushed and restarted; hence they don't reveal progress as nicely as they do in the other algorithms.]

**265.** (1) A literal  $L_t$  on the trail with  $G \leq t < F$  has become true, but the watch list of  $\bar{L}_t$  has not yet been examined. (2) If  $l_0$  is true, so that  $c$  is satisfied, step C4 doesn't

watched literals  
lazy data structures  
heap deletion  
deletion from heap  
siftup  
heap insertion  
insertion into a heap  
*blit*  
redundancies  
watch a literal  
flushed  
restarted

remove  $c$  from the watch list of  $l_1$  when  $l_1$  becomes false. (This behavior is justified, because  $c$  won't be examined again until  $l_1$  has become free during the backtracking step C8.) (3) A clause that becomes a reason for  $l_0$  remains on the watch list of its false  $l_1$ . (4) During a full run, a clause that triggers a conflict is allowed to keep both of its watched literals false.

In general, a false watched literal must be defined at the highest trail level of all literals in its clause.

**266.** If  $U < p$ , where  $U$  is a uniform deviate between 0 and 1, do this: Set  $j$  to a random integer with  $0 \leq j < h$ , and  $k \leftarrow \text{HEAP}[j]$ . If  $j = 0$ , or if  $\text{VAL}(k) \geq 0$ , use the normal C6. Otherwise branch on  $k$  (and don't bother to remove  $k$  from the heap).

**267.** As in Algorithm L, let there be a sequential table  $\text{BIMP}(l)$  for each literal  $l$ , containing all literals  $l'$  such that  $\bar{l} \vee l'$  is a binary clause. Furthermore, when the propagation algorithm sets  $L_F \leftarrow l'$  because  $l' \in \text{BIMP}(l)$ , we may set  $R_{l'} \leftarrow -l$ , instead of using a positive clause number as the "reason." (Notice that a binary clause therefore need not be represented explicitly in MEM, if it is represented implicitly in the BIMP tables. The author's implementation of Algorithm C uses BIMP tables only to expedite binary clauses that appear in the original input. This has the advantage of simplicity, since the exact amount of necessary space can be allocated permanently for each table. Learned binary clauses are comparatively rare in practice; thus they can usually be handled satisfactorily with watched literals, instead of by providing the elaborate buddy-system scheme that was important in Algorithm L.)

Here, more precisely, is how the inner loop goes faster with BIMPs. We want to carry out binary propagations as soon as possible, because of their speed; hence we introduce a breadth-first exploration process analogous to (62):

$$\begin{aligned} &\text{Set } H \leftarrow F; \text{ take account of } l' \text{ for all } l' \in \text{BIMP}(l_0); \\ &\text{while } H < F, \text{ set } l_0 \leftarrow R_H, H \leftarrow H + 1, \text{ and} \\ &\quad \text{take account of } l' \text{ for all } l' \in \text{BIMP}(l_0). \end{aligned} \quad (**)$$

Now "take account of  $l'$ " means "if  $l'$  is true, do nothing; if  $l'$  is false, go to C7 with conflict clause  $\bar{l} \vee l'$ ; otherwise set  $L_F \leftarrow l'$ ,  $\text{TLOC}(|l'|) \leftarrow F$ ,  $\text{VAL}(|l'|) \leftarrow 2d + (l' \& 1)$ ,  $R_{l'} \leftarrow -l$ ,  $F \leftarrow F + 1$ ." We do (\*\*) just before setting  $c \leftarrow c'$  in answer 261. Furthermore, we set  $E \leftarrow F$  just after  $G \leftarrow 0$  in step C1 and just after  $F \leftarrow F + 1$  in steps C6 and C9; and if  $G \leq E$  after  $G \leftarrow G + 1$  in step C4, we do (\*\*) with  $l_0 \leftarrow \bar{l}$ .

Answer 263 is modified in straightforward ways so that "clause  $R_l$ " is treated as if it were the binary clause  $(l \vee \bar{l})$  when  $R_l$  has the negative value  $-l'$ .

**268.** If  $\text{MEM}[c - 1] = k \geq 3$  is the size of clause  $c$ , and if  $1 < j < k$ , we can delete the literal  $l$  in  $\text{MEM}[c + j]$  by setting  $k \leftarrow k - 1$ ,  $\text{MEM}[c - 1] \leftarrow k$ ,  $l' \leftarrow \text{MEM}[c + k]$ ,  $\text{MEM}[c + j] \leftarrow l'$ , and  $\text{MEM}[c + k] \leftarrow l + f$ , where  $f$  is a flag (typically  $2^{31}$ ) that distinguishes a deleted literal from a normal one. (This operation does not need to be done when the current level  $d$  is zero; hence we can assume that  $k \geq 3$  and  $j > 1$  before deletion. The flag is necessary so that global operations on the entire set of clauses, such as the purging algorithm, can pass safely over deleted literals. The final clause in MEM should be followed by 0, an element that's known to be unflagged.)

**269.** (a) If the current clause contains a literal  $l = \bar{L}_t$  that is not in the trivial clause, where  $t$  is maximum, resolve the current clause with  $R_{\bar{l}}$  and repeat.

(b)  $(\bar{u}_1 \vee \bar{b}_j) \wedge (l_j \vee \bar{l}_{j-1} \vee \bar{b}_j)$  for  $1 \leq j \leq 9$ ,  $(l_0 \vee \bar{u}_2 \vee \bar{u}_3) \wedge (\bar{l}_9 \vee \bar{l}_8 \vee \bar{b}_{10})$ ;  $l' = l_0$ .

(c) If  $r \geq d' + \tau$ , where  $\tau$  is a positive parameter, learn the trivial clause instead of  $(\bar{l}' \vee \bar{b}_1 \vee \dots \vee \bar{b}_r)$ . (The watched literals should be  $\bar{l}'$  and  $\bar{u}_{d'}$ .)

full run  
BIMP( $l$ )  
Knuth  
buddy-system  
breadth-first  
flag  
purging  
watched literals



Notice that this procedure will learn more than simple backtrack à la Algorithm D does, even when the trivial clause is *always* substituted (that is, even when  $\tau = -\infty$ ), because it provides for backjumping when  $d' < d + 1$ .

**270.** (a) Consider the clauses  $3\bar{2}$ ,  $4\bar{3}\bar{2}$ ,  $5\bar{4}\bar{3}\bar{1}$ ,  $6\bar{5}\bar{4}\bar{1}$ ,  $\bar{6}\bar{5}\bar{4}$ , with initial decisions  $L_1 \leftarrow 1$ ,  $L_2 \leftarrow 2$ . Then  $L_3 \leftarrow 3$  with reason  $R_3 \leftarrow 3\bar{2}$ ; similarly  $L_4 \leftarrow 4$ ,  $L_5 \leftarrow 5$ . If  $L_6 \leftarrow 6$ , the conflict clause  $\bar{6}\bar{5}\bar{4}$  allows us to strengthen  $R_6$  to  $\bar{5}\bar{4}\bar{1}$ ; but if  $L_6 \leftarrow \bar{6}$ , with  $R_{\bar{6}} \leftarrow \bar{6}\bar{5}\bar{4}$ , we don't notice that  $6\bar{5}\bar{4}\bar{1}$  can be strengthened. In either case we can, however, strengthen  $R_5$  to  $\bar{4}\bar{3}\bar{1}$ , before learning the clause  $\bar{2}\bar{1}$ .

(b) After doing *blit*( $l_j$ ) to the literals of  $R_i$ , we know that  $R_i \setminus l$  is contained in  $\{\bar{b}_1, \dots, \bar{b}_r\}$  together with  $q + 1$  unresolved false literals that have been stamped at level  $d$ . (Exercise 268 ensures that  $p \neq 0$  within each *blit*.) Thus we can subsume clause  $R_i$  on the fly if  $q + r + 1 < k$  and  $q > 0$ .

In such cases the procedure of answer 268 can be used to delete  $l$  from  $c = R_i$ . But there's a complication, because  $l = l_0$  is a watched literal ( $j = 0$  in that answer), and all other literals are false. After  $l$  is deleted, it will be essential to watch a false literal  $l'$  that is defined at trail level  $d$ . So we find the largest  $j' \leq k$  such that  $\text{VAL}(\text{MEM}[c + j']) \geq 2d$ , and we set  $l' \leftarrow \text{MEM}[c + j']$ . If  $j' \neq k$ , we also set  $\text{MEM}[c + j'] \leftarrow \text{MEM}[c + k]$ ; we can assume that  $j' > 1$ . Finally, after setting  $\text{MEM}[c] \leftarrow l'$  and  $\text{MEM}[c + k] \leftarrow l + f$  as in answer 268, we also delete  $c$  from the watch list  $W_i$ , and insert it into  $W_{i'}$ .

[This enhancement typically saves 1%–10% of the running time, but sometimes it saves a lot more. It was discovered in 2009, independently by two different groups of researchers: See H. Han and F. Somenzi, *LNCS 5584* (2009), 209–222; Y. Hamadi, S. Jabbour, and L. Saïs, *Int. Conf. Tools with Artif. Int.* (ICTAI) **21** (2009), 328–335.]

**271.** We shall check for discards only if the current clause  $C_i$  is not trivial (see exercise 269), and if the first literal of  $C_{i-1}$  does not appear in the trail. (Indeed, experience shows that almost every permissible discard falls into this category.) Thus, let  $C_{i-1}$  be  $l_0 l_1 \dots l_{k-1}$  where  $\text{VAL}(|l_0|) < 0$ ; we want to decide if  $\{\bar{l}', \bar{b}_1, \dots, \bar{b}_r\} \subseteq \{l_1, \dots, l_{k-1}\}$ .

The secret is to use the stamp fields that have already been set up. Set  $j \leftarrow k - 1$ ,  $q \leftarrow r + 1$ , and do the following while  $q > 0$  and  $j \geq q$ : If  $l_j = \bar{l}'$ , or if  $\text{VAL}(|l_j|) \leq 2d' + 1$  and  $\text{S}(|l_j|) = s$ , set  $q \leftarrow q - 1$ ; in any case set  $j \leftarrow j - 1$ . Then discard if  $q = 0$ .

**272.** Reflection isn't as easy to implement as it may seem, unless  $C$  is a unit clause, because  $C^R$  must be placed carefully in  $\text{MEM}$  and it must be consistent with the trail. Furthermore, experience shows that it's best not to learn the reflection of *every* learned clause, because excess clauses make unit propagation slower. The author has obtained encouraging results, however, by doing the following operations just before returning to C3 in step C9, whenever the length of  $C$  doesn't exceed a given parameter  $R$ :

Assign ranks to the literals of  $C^R$  by letting  $\text{rank}(l) = \infty$  if  $l$  is on the trail,  $\text{rank}(l) = d''$  if  $\bar{l}$  is on the trail at level  $d'' < d'$ ,  $\text{rank}(l) = d$  otherwise. Let  $u$  and  $v$  be two of the highest ranking literals, with  $\text{rank}(u) \geq \text{rank}(v)$ . Put them into the first two positions of  $C^R$ , so that they will be watched. Do nothing further if  $\text{rank}(v) > d'$ . Otherwise, if  $\text{rank}(v) < d'$ , backjump to level  $\text{rank}(v)$  and set  $d' \leftarrow \text{rank}(v)$ . Then if  $\text{rank}(u) = \text{rank}(v) = d'$ , treat  $C^R$  as a conflict clause by going to step C7 with  $c \leftarrow C^R$ . (That is a rare event, but it can happen.) Otherwise, if  $u$  doesn't appear in the current trail, set  $L_F \leftarrow u$ ,  $\text{TLOC}(|u|) \leftarrow F$ ,  $R_u \leftarrow C^R$ ,  $F \leftarrow F + 1$ . (Possibly  $F = E + 2$  now.)

(For example, this method with  $R \leftarrow 6$  roughly halved the running time of *waarden*(3, 10; 97) and *waarden*(3, 13; 160) with parameters (193) except for  $\rho \leftarrow .995$ .)

A similar idea works with the clauses *langford*( $n$ ), and in general whenever the input clauses have an automorphism of order 2.

backtrack  
backjumping  
*blit*  
watched literal  
Han  
Somenzi  
Hamadi  
Jabbour  
Saïs  
trivial  
stamp  
trail  
unit propagation  
author  
backjump  
*langford*( $n$ )  
automorphism

**273.** (a) We can convert Algorithm C into a “clause learning machine” by keeping the process going after  $F$  reaches  $n$  in step C5: Instead of terminating, start over again by essentially going back to step C1, except that the current collection of clauses should be retained, and the OVAL polarities should be reset to random bits. Learned clauses of size  $K$  or less, where  $K$  is a parameter, should be written to a file. Stop when you’ve found a given number of short clauses, or when you’ve exceeded a given time limit.

For example, here’s what happened when the author first tried to find  $W(3, 13)$ : Applying this algorithm to  $waerden(3, 13; 158)$  with  $K = 3$ , and with a timeout limit of  $30 \text{ G}\mu$  (gigamems), yielded the five clauses 65 68 70, 68 78 81, 78 81 90, 78 79 81, 79 81 82. So fifteen clauses 65 68 70, 66 69 71,  $\dots$ , 81 83 84 could be added to  $waerden(3, 13; 160)$ , as well as their fifteen reflections 96 93 91, 95 92 90,  $\dots$ , 80 78 77. Then the algorithm “C<sup>R</sup>” of exercise 272 proved this augmented set unsatisfiable after an additional  $107 \text{ G}\mu$ . In a second experiment, using  $K = 2$  with  $waerden(3, 13; 159)$  led to three binary clauses 76 84, 81 86, and 84 88. Shifting and reflecting gave twelve binary clauses, which in company with  $waerden(3, 13; 160)$  were refuted by C<sup>R</sup> in another  $80 \text{ G}\mu$ . (For comparison, Algorithm C<sup>R</sup> refuted  $waerden(3, 13; 160)$  unaided in about  $120 \text{ G}\mu$ , compared to about  $270 \text{ G}\mu$  for both Algorithm C and Algorithm L.) Optimum strategies for learning useful clauses from satisfiable subproblems are far from clear, especially because running times are highly variable. But this method does show promise, especially on more difficult problems—when more time can be devoted to the preliminary learning.

(b) Short clauses that can be learned from satisfiable instances of, say,  $X_0 \rightarrow X_1 \rightarrow \dots \rightarrow X_{r-1}$ , when  $X_0$  is *not* required to be an initial state, can be shifted and used to help refute  $X_0 \rightarrow X_1 \rightarrow \dots \rightarrow X_r$ .

**274.** With care, circular reasoning can (and must) be avoided. But the author’s elaborate experiments with such ideas (and with the related notion of “better conflicts”) were disappointing; they didn’t beat the running time of the simpler algorithm. However, an intriguing idea by Allen Van Gelder [*Journal on Satisfiability, Boolean Modeling and Computation* 8 (2012), 117–122] shows promise.

**275.** When a solution has been found, let  $k$  be minimum such that  $x_k = 1$  and the value of  $x_k$  has not been assigned at level 0. If no such  $k$  exists, we stop. Otherwise we are entitled to force variables  $x_1$  through  $x_{k-1}$  all to have their current values, at level 0, because we know that this doesn’t produce an unsatisfiable problem. So we fix those values, and we restart the solution process at level 1 with the tentative decision ‘ $x_k = 0$ ’. If a conflict occurs, we’ll know that  $x_k = 1$  at level 0; if not, we’ll have a solution with  $x_k = 0$ . In either case we can increase  $k$ . (This method is considerably better than that of answer 109, because *every learned clause remains valid*.)

**276.** True. Unit propagation essentially transforms  $F \wedge L$  into  $F|L$ .

**277.** Otherwise  $F \wedge C_1 \wedge \dots \wedge C_{t-1} \vdash_1 \epsilon$  fails (unit propagation wouldn’t start).

**278.** For example,  $(46, \bar{5}6, \bar{5}4, 6, 4, \epsilon)$ . (Six steps are necessary.)

**279.** True, because the dependency digraph contains a literal  $l$  with  $l \rightarrow^* \bar{l} \rightarrow^* l$ .

**280.** (a) They’re satisfied if and only if  $x_1 \dots x_n$  has at least  $j$  0s and at least  $k$  1s. [The problem  $cook(k, k)$  was introduced by Stephen A. Cook (unpublished) in 1971.]

(b) Take all positive  $(j - t)$ -clauses on  $\{1, \dots, n - 1 - t\}$  for  $t = 1, 2, \dots, j$ .

(c) Suppose the very first decision is  $L_0 \leftarrow x_n$ . The algorithm will proceed to act as if the input were  $cook(j, k) | x_n = cook(j, k - 1)$ . Furthermore, with these clauses, every clause that it learns initially will include  $\bar{x}_n$ . Therefore, by induction, the unit clause  $(\bar{x}_n)$  will be learned clause number  $\binom{n-2}{j-1}$ . All previously learned clauses are

OVAL  
polarities  
author  
Knuth  
Van Gelder  
dependency digraph  
Cook

subsumed by this one, hence they're no longer relevant. The remaining problem is  $cook(j, k) \mid \bar{x}_n = cook(j - 1, k)$ ; so the algorithm will finish after learning  $\binom{n-2}{j-2}$  more.

Similarly, if the first decision is  $L_0 \leftarrow \bar{x}_n$ , the  $\binom{n-2}{j-2}$ th learned clause will be  $(x_n)$ .

**281.** Stålmarck's refutation corresponds to the sequence  $(M'_{jk1}, M'_{jk2}, \dots, M'_{jk(k-1)}, M_{j(k-1)})$  for  $j = 1, \dots, k - 1$ , for  $k = m, m - 1, \dots, 1$ . ( $M'_{jk(k-1)}$  can be omitted.)

**282.** First learn the exclusion clauses (17). In the next clauses we shall write  $a_j, b_j, \dots$ , as shorthand for  $a_{j,p}, b_{j,p}, \dots$ , where  $p$  is a particular color,  $1 \leq p \leq 3$ . Notice that the  $12q$  edges appear in  $4q$  triangles, namely  $\{b_j, c_j, d_j\}, \{a_j, a_{j'}, b_{j'}\}, \{f_j, e_{j'}, c_{j'}\}, \{e_j, f_{j'}, d_{j'}\}$ , for  $1 \leq j \leq q$ , where  $j'$  is  $j + 1$  (modulo  $q$ ). For every such triangle  $\{u, v, w\}$ , learn  $(\bar{u}_{p'} \vee v_p \vee w_p)$  and then  $(u_p \vee v_p \vee w_p)$ , where  $p'$  is  $p + 1$  (modulo 3).

Now for  $j = 1, 2, \dots, q$ , learn  $(a_j \vee f_j \vee a_{j'} \vee e_{j'}), (a_j \vee e_j \vee a_{j'} \vee f_{j'}), (e_j \vee f_j \vee e_{j'} \vee f_{j'}), (\bar{a}_j \vee \bar{e}_j \vee \bar{e}_{j'}), (\bar{a}_j \vee \bar{f}_j \vee \bar{f}_{j'}), (\bar{e}_j \vee \bar{f}_j \vee \bar{a}_{j'})$ , as well as eighteen more:

$$\begin{aligned} &(\bar{u}_1 \vee \bar{v}_1 \vee u'_j \vee v'_j), (\bar{u}_2 \vee \bar{v}_2 \vee u'_{j'} \vee v'_{j'}), & \text{if } j \geq 3 \text{ is odd;} \\ &(\bar{u}_1 \vee \bar{v}_1 \vee \bar{u}'_j), (\bar{u}_2 \vee \bar{v}_2 \vee \bar{u}'_{j'}), & \text{if } j \geq 3 \text{ is even;} \end{aligned}$$

here  $u, v \in \{a, e, f\}$  and  $u', v' \in \{a, e, f\}$  yield  $3 \times 3$  choices of  $(u, v, u', v')$ . Then we're ready to learn  $(\bar{a}_j \vee \bar{e}_j), (\bar{a}_j \vee \bar{f}_j), (\bar{e}_j \vee \bar{f}_j)$  for  $j \in \{1, 2\}$  and  $(a_j \vee e_j \vee f_j \vee a_{j'}), (a_j \vee e_j \vee f_j)$  for  $j \in \{1, q\}$ . All of these clauses are to be learned for  $1 \leq p \leq 3$ .

Next, for  $j = q, q - 1, \dots, 2$ , learn  $(\bar{a}_j \vee \bar{e}_j), (\bar{a}_j \vee \bar{f}_j), (\bar{e}_j \vee \bar{f}_j)$  for  $1 \leq p \leq 3$  and then  $(a_{j-1} \vee e_{j-1} \vee f_{j-1} \vee a_j), (a_{j-1} \vee e_{j-1} \vee f_{j-1})$  for  $1 \leq p \leq 3$ . We have now established all clauses in the hint.

The endgame consists of the following for  $1 \leq p \leq 3$ : For all choices of  $p'$  and  $p''$  with  $\{p, p', p''\} = \{1, 2, 3\}$  (thus two choices), and for  $j = 2, 3, \dots, q$ , learn three clauses

$$\begin{aligned} &(\bar{a}_{1,p} \vee \bar{e}_{1,p'} \vee \bar{a}_{j,p} \vee e_{j,p''}), (\bar{a}_{1,p} \vee \bar{e}_{1,p'} \vee \bar{a}_{j,p'} \vee e_{j,p}), (\bar{a}_{1,p} \vee \bar{e}_{1,p'} \vee \bar{a}_{j,p''} \vee e_{j,p'}), & j \text{ even;} \\ &(\bar{a}_{1,p} \vee \bar{e}_{1,p'} \vee \bar{a}_{j,p} \vee e_{j,p'}), (\bar{a}_{1,p} \vee \bar{e}_{1,p'} \vee \bar{a}_{j,p'} \vee e_{j,p''}), (\bar{a}_{1,p} \vee \bar{e}_{1,p'} \vee \bar{a}_{j,p''} \vee e_{j,p}), & j \text{ odd;} \end{aligned}$$

then learn  $(\bar{a}_{1,p} \vee \bar{e}_{1,p'})$ . Finally learn  $\bar{a}_{1,p}$ .

[Not all of these clauses are actually necessary. For example, the exclusion clauses for  $b$ 's,  $c$ 's, and  $d$ 's aren't used. This certificate doesn't assume that the symmetry-breaking unit clauses  $b_{1,1} \wedge c_{1,2} \wedge d_{1,3}$  of  $fsnark(q)$  are present; indeed, those clauses don't help it much. The actual clauses learned by Algorithm C are considerably longer and somewhat chaotic (indeed mysterious); it's hard to see just where an "aha" occurs!]

**283.** A related question is to ask whether the expected length of learned clauses is  $O(1)$  as  $q \rightarrow \infty$ .

**284.** It's convenient to represent each unit clause  $(l)$  in  $F \cup C_1 \cup \dots \cup C_t$  as if it were the binary clause  $(l \vee \bar{x}_0)$ , where  $x_0$  is a new variable that is always true. We borrow some of the data structures of Algorithm C, namely the trail array  $L$ , the reason array  $R$ , and the fields **TLOC**, **S**, **VAL** associated with each variable. We set  $\text{VAL}(k) = 0, 1$ , or  $-1$  when  $x_k$  has been forced true, forced false, or not forced, respectively.

To verify the clause  $C_i = (a_1 \vee \dots \vee a_k)$ , we begin with  $\text{VAL}(j) \leftarrow 0$  for  $0 \leq j \leq n$ ,  $L_0 \leftarrow 0$ ,  $L_1 \leftarrow \bar{a}_1, \dots, L_k \leftarrow \bar{a}_k$ ,  $E \leftarrow F \leftarrow k + 1$ ,  $G \leftarrow 0$ , and  $\text{VAL}(|L_p|) \leftarrow L_p \ \& \ 1$  for  $0 \leq p < F$ ; then we carry out unit propagation as in Algorithm C, expecting to reach a conflict before  $G = F$ . (Otherwise verification fails.)

A conflict arises when a clause  $c = l_0 \dots l_{k-1}$  forces  $l_0$  at a time when  $\bar{l}_0$  has already been forced. Now we mimic step C7 (see exercise 263), but the operations are much simpler: Mark  $c$ , stamp  $\text{S}(|l_j|) \leftarrow i$  for  $0 \leq j < k$ , and set  $p \leftarrow \max(\text{TLOC}(|l_1|), \dots,$

Stålmarck  
exclusion clauses  
at-most-one  
symmetry-breaking  
unit clause  
data structures

$\text{TLOC}(|l_{k-1}|)$ ). Now, while  $p \geq E$ , we set  $l \leftarrow L_p$ ,  $p \leftarrow p - 1$ , and if  $\mathbf{S}(|l|) = i$  we also “resolve with the reason of  $l$ ” as follows: Let clause  $R_l$  be  $l_0 l_1 \dots l_{k-1}$ , mark  $R_l$ , and set  $\mathbf{S}(|l_j|) \leftarrow i$  for  $1 \leq j < k$ .

[Wetzler, Heule, and Hunt have suggested an interesting improvement, which will often mark significantly fewer clauses at the expense of a more complicated algorithm: Give preference to already-marked clauses when doing the unit propagations, just as Algorithm L prefers binary implications to the implications of longer clauses (see (62)).]

Wetzler  
Heule  
Hunt  
worst case  
tie-breakers  
clause activity  
trivial  
backjump  
floating point

**285.** (a)  $j = 77$ ,  $s_{77} = 12 + 2827$ ,  $m_{77} = 59$ ,  $b_{77} = 710$ .

(b)  $j = 72$ ,  $s_{72} = 12 + 2048$ ,  $m_{72} = 99 + 243 + 404 + 536 = 1282$ ,  $b_{72} = 3 + 40 + 57 + 86 = 186$ . (The RANGE statistic is rather coarse when  $\alpha = \frac{1}{2}$ , because many different signatures yield the same value.)

(c)  $j = 71$ ,  $s_{71} = 12 + 3087$ ,  $m_{71} = 243$ ,  $b_{71} = 40$ .

**286.** The maximum, 738, is achieved uniquely by the RANGE-oriented solution with  $\alpha = \frac{15}{16}$ , except that we can optionally include also the signatures (6, 0) and (7, 0) for which  $a_{pq} = 0$ . [This solution optimizes the worst case of clause selection, because the stated problem implicitly assumes that the secondary heuristic is bad. If we assume, however, that the choice of tie-breakers based on clause activity is at least as good as a random choice, then the expected number  $738 + 45 \cdot \frac{10}{59} \approx 745.6$  from  $\alpha = \frac{15}{16}$  is *not* as good as the expected number  $710 + 287 \cdot \frac{57}{404} \approx 750.5$  from  $\alpha = \frac{9}{16}$ .]

**287.** When a conflict is detected in step C7 (with  $d > 0$ ), keep going as in step C3; but remember the first clause  $C_d$  that detected a conflict at each level  $d$ .

Eventually step C5 will find  $F = n$ . That’s when clauses get their RANGE scores, if we’re doing a full run because we want to purge some of them. (Sometimes, however, it’s also useful to do a few full runs at the very beginning, or just after a restart, because some valuable clauses might be learned.)

New clauses can be learned in the usual way from the remembered clauses  $C_d$ , in decreasing order of  $d$ , except that “trivial” clauses (exercise 269) are considered only at the lowest such level. We must keep track of the minimum backjump level  $d'$ , among all of these conflicts. And if several new clauses have the same  $d'$ , we must remember all of the literals that will be placed at the end of the trail after we eventually jump back.

**288.** Step C5 initiates a full run, then eventually finds  $F = n$ . At this point we’re done, in the unlikely event that no conflicts have arisen. Otherwise we set  $\text{LS}[d] \leftarrow 0$  for  $0 \leq d < n$  and  $m_j \leftarrow 0$  for  $1 \leq j < 256$ . The activity  $\text{ACT}(c)$  of each learned clause  $c$  has been maintained in  $\text{MEM}[c - 5]$ , as a 32-bit floating point number. The following steps compute  $\text{RANGE}(c)$ , which will be stored in  $\text{MEM}[c - 4]$  as an integer, for all learned  $c$  in increasing order, assuming that  $c$ ’s literals are  $l_0 l_1 \dots l_{s-1}$ :

If  $R_{l_0} = c$ , set  $\text{RANGE}(c) \leftarrow 0$ . Otherwise set  $p \leftarrow r \leftarrow 0$ , and do the following for  $0 \leq k < s$ : If  $v < 2$  and  $v + l_k$  is even, set  $\text{RANGE}(c) \leftarrow 256$  and exit the loop on  $k$  (because  $c$  is permanently satisfied, hence useless). If  $v \geq 2$  and  $\text{LS}[\text{lev}(l_k)] < c$ , set  $\text{LS}[\text{lev}(l_k)] \leftarrow c$  and  $r \leftarrow r + 1$ . Then if  $v \geq 2$  and  $\text{LS}[\text{lev}(l_k)] = c$  and  $l_k + v$  is even, set  $\text{LS}[\text{lev}(l_k)] \leftarrow c + 1$  and  $p \leftarrow p + 1$ . After  $k$  reaches  $s$ , set  $r \leftarrow \min([\mathbf{16}(p + \alpha(r - p))], 255)$ ,  $\text{RANGE}(c) \leftarrow r$ , and  $m_r \leftarrow m_r + 1$ .

Now resolve conflicts (see answer 287), giving  $\text{ACT}(c) \leftarrow 0$  and  $\text{RANGE}(c) \leftarrow 0$  to all newly learned clauses  $c$ , and jump back to trail level 0. (A round of purging is a major event, something like spring cleaning. It is possible that  $d' = 0$ , in which case one or more literals have been appended to trail level 0 and their consequences have not yet been explored.) Find the median range  $j$  as defined in (124), where  $T$  is half the total current number of learned clauses. If  $j < 256$  and  $T > s_j$ , find  $h = T - s_j$  clauses

with  $\text{RANGE}(c) = j$  and  $\text{ACT}(c)$  as small as possible, and bump their range up to  $j + 1$ . (This can be done by putting the first  $m_j - h$  of them into a heap, then repeatedly bumping the least active as the remaining  $h$  are encountered; see exercise 6.1–22.)

Finally, go again through all the learned clauses  $c$ , in order of increasing  $c$ , ignoring  $c$  if  $\text{RANGE}(c) > j$ , otherwise copying it into a new location  $c' \leq c$ . (Permanently false literals, which are currently defined at level 0, can also be removed at this time; thus the clause's size in  $\text{MEM}[c' - 1]$  might be less than  $\text{MEM}[c - 1]$ . It is possible, but unlikely, that a learned clause becomes reduced to a unit in this way, or even that it becomes empty.) The activity score in  $\text{MEM}[c - 5]$  should be copied into  $\text{MEM}[c' - 5]$ ; but  $\text{RANGE}(c)$  and the watch links in  $\text{MEM}[c - 2]$  and  $\text{MEM}[c - 3]$  needn't be copied.

When copying is complete, all the watch lists should be recomputed from scratch, as in answer 260, including original clauses as well as the learned clauses that remain.

**289.** By induction,  $y_k = (2 - 2^{1-k})\Delta + (2(k - 2) + 2^{2-k})\delta$  for all  $k \geq 0$ .

**290.** Set  $k \leftarrow \text{HEAP}[0]$ ; then if  $\text{VAL}(k) \geq 0$ , delete  $k$  from the heap as in answer 262, and repeat this loop.

**291.**  $\text{OVAL}(49)$  will be the even number 36, because of the propagations on level 18 that led to (115).

**292.** If  $\text{AGILITY} \geq 2^{32} - 2^{13}$ , then (127) either subtracts  $2^{19} - 1$  or adds 1. Hence there's a minuscule chance that  $\text{AGILITY}$  will overflow by passing from  $2^{32} - 1$  to  $2^{32}$  (zero). (But overflow won't be a calamity even if—unbelievably—it happens. So this is one “bug” in the author's program that he will *not* try to fix.)

**293.** Maintain integers  $u_f$ ,  $v_f$ , and  $\theta_f$ , where  $\theta_f$  has 64 bits. Initially  $u_f = v_f = M_f = 1$ . When  $M \geq M_f$  is detected in step C5, do this: If  $u_f \& -u_f = v_f$ , set  $u_f \leftarrow u_f + 1$ ,  $v_f \leftarrow 1$ ,  $\theta_f \leftarrow 2^{32}\psi$ ; otherwise set  $v_f \leftarrow 2v_f$  and  $\theta_f \leftarrow \theta_f + (\theta_f \gg 4)$ . Flush if  $\text{AGILITY} \leq \theta_f$ .

**294.** We have, for example,  $g_{1100} = \frac{2}{3}(g_{0100} + g_{1000} + g_{1110})$ , and  $g_{01*1} = 1$ . The solution is  $g_{00*1} = g_{01*0} = g_{11*1} = A/D$ ,  $g_{00*0} = g_{10*1} = g_{11*0} = B/D$ ,  $g_{10*0} = C/D$ , where  $A = 3z - z^2 - z^3$ ,  $B = z^2$ ,  $C = z^3$ ,  $D = 9 - 6z - 3z^2 + z^3$ . Hence the overall generating function is  $g = (6A + 6B + 2C + 2D)/(16D)$ ; and we find  $g'(1) = 33/4$ ,  $g''(1) = 147$ . Thus  $\text{mean}(g) = 8.25$ ,  $\text{var}(g) = 87.1875$ , and the standard deviation is  $\approx 9.3$ .

**295.** Consider all  $3\binom{n}{3}$  clauses  $\bar{x}_i \vee x_j \vee x_k$  for distinct  $\{i, j, k\}$ , plus two additional clauses  $(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_4 \vee \bar{x}_5 \vee \bar{x}_6)$  to make the solution  $0 \dots 0$  unique. Only the two latter clauses cause the variables  $X_i$  and  $Y_i$  in the proof of Theorem U to deviate from each other. [C. Papadimitriou, *Computational Complexity* (1994), Problem 11.5.6. These clauses spell trouble for a lot of other SAT algorithms too.]

**296.** The hinted ratio  $2(2p+q+1)(2p+q)/(9(p+1)(p+q+1))$  is  $\approx 1$  when  $p \approx q$  (more precisely when  $p = q - 7 + O(1/q)$ ). And  $f(q+1, q+1)/f(q, q) = 2(n-q)(3q+3)^2/(27(q+1)^2(2q+2)^2)$  is  $\approx 1$  when  $q \approx n/3$ . Finally,  $f(n/3, n/3) = \frac{3}{4\pi n}(3/4)^n(1+O(1/n))$  by Stirling's approximation, when  $n = 3q$ .

**297.** (a)  $G_q(z) = (z/3)^q C(2z^2/9)^q = G(z)^q$  where  $G(z) = (3 - \sqrt{9 - 8z^2})/(4z)$ , by Eqs. 7.2.1.6–(18) and (24). [See *Algorithmica* **32** (2002), 620–622.]

(b)  $G_q(1) = 2^{-q}$  is the probability that  $Y_t$  actually reaches 0, for some finite  $t$ .

(c) If the  $Y$  process does stop,  $G_q(z)/G_q(1) = (2G(z))^q$  describes the distribution of stopping times. Hence  $G'_q(1)/G_q(1) = 2qG'(1) = 3q$  is the mean length of the random walk, *given* that it terminates. (The variance, incidentally, is  $24q$ . A random  $Y$ -walker who doesn't finish quickly is probably doomed to wander forever.)

(d) The generating function for  $T$ , the stopping time of the  $Y$  process, is  $T(z) = \sum_q \binom{n}{q} 2^{-n} G_q(z)$ ; and  $T$  is finite with probability  $T(1) = (\frac{3}{4})^n$  by (b). If we restrict

heap  
OVAL  
overflow  
bug  
author  
standard deviation  
Papadimitriou  
variance

consideration to such scenarios, the mean  $T'(1)/T(1)$  is  $n$ ; and Markov's inequality tells us that  $\Pr(T \geq N) \leq n/N$ .

(e) The algorithm succeeds with probability  $p > \Pr(T < N) \geq (1 - n/N)(3/4)^n$ , when it is given satisfiable clauses. So it fails after  $K(4/3)^n$  trials with probability less than  $\exp(K(4/3)^n \ln(1 - p)) < \exp(-K(4/3)^n p) < \exp(-K/2)$  when  $N = 2n$ .

**298.** Change  $1/3$  and  $2/3$  in (129) to  $1/k$  and  $(k-1)/k$ . The effect is to change  $G(z)$  to  $(z/k)C((k-1)z^2/k^2)$ , with  $G(1) = 1/(k-1)$  and  $G'(1) = k/((k-1)(k-2))$ . As before,  $T(1) = 2^{-n}(1+G(1))^n$  and  $T'(1)/T(1) = nG'(1)/(1+G(1))$ . So the generalized Corollary W gives success probability  $> 1 - e^{-K/2}$  when we apply Algorithm P  $K(2 - 2/k)^n$  times with  $N = \lfloor 2n/(k-2) \rfloor$ .

**299.** In this case  $G(z) = (1 - \sqrt{1-z^2})/z$ ; thus  $G(1) = T(1) = 1$ . But  $G'(1) = \infty$ , so we must use a different method. The probability of failure if  $N = n^2$  is

$$\begin{aligned} \frac{1}{2^n} \sum_{p,q} \binom{n}{q} \frac{q}{2p+q} \binom{2p+q}{p} \frac{[2p+q > n^2]}{2^{2p+q}} &= \sum_{t > n^2} \frac{2^{-n-t}}{t} \sum_p \binom{n}{t-2p} \binom{t}{p} (t-2p) \\ &\leq \sum_{t > n^2} \frac{2^{-n-t}}{t} \binom{t}{\lfloor t/2 \rfloor} \sum_p \binom{n}{t-2p} (t-2p) = \frac{n}{4} \sum_{t > n^2} \frac{2^{-t}}{t} \binom{t}{\lfloor t/2 \rfloor} \\ &< \frac{n}{4} \sum_{t > n^2} \sqrt{\frac{2}{\pi t^3}} = \frac{n}{\sqrt{8\pi}} \int_{n^2}^{\infty} \frac{dx}{\lceil x^{3/2} \rceil} < \frac{n}{\sqrt{8\pi}} \int_{n^2}^{\infty} \frac{dx}{x^{3/2}} = \frac{1}{\sqrt{2\pi}}. \end{aligned}$$

[See C. Papadimitriou, *Computational Complexity* (1994), Theorem 11.1.]

**300.** In this algorithm, variables named with uppercase letters (except  $C$  and  $N$ ) denote bit vectors of some fixed size (say 64); each bit position represents a separate trial. The notation  $U_r$  stands for a vector of *random* bits, each of which is 1 with probability  $1/r$ , independently of all other bits and all previous  $U$ 's. The maximum number of flips per bit position in this variant of Algorithm P is only *approximately* equal to  $N$ .

**P1'**. [Initialize.] Set  $X_i \leftarrow U_2$  for  $1 \leq i \leq n$ . Also set  $t \leftarrow 0$ .

**P2'**. [Begin pass.] Set  $Z \leftarrow 0$  and  $j \leftarrow 0$ . (Flipped positions are remembered in  $Z$ .)

**P3'**. [Move to next clause.] If  $j = m$ , go to P5'. Otherwise set  $j \leftarrow j + 1$ .

**P4'**. [Flip.] Let  $C_j$  be the clause  $(l_1 \vee \dots \vee l_k)$ . Set  $Y \leftarrow \bar{L}_1 \& \dots \& \bar{L}_k$ , where  $L_i$  denotes  $X_h$  if  $l_i = x_h$  and  $L_i$  denotes  $\bar{X}_h$  if  $l_i = \bar{x}_h$ . (Thus  $Y$  has 1s in positions that violate clause  $C_j$ .) Set  $Z \leftarrow Z \vee Y$  and  $t \leftarrow t + (Y \& 1)$ . Then for  $r = k, k-1, \dots, 2$  set  $Y' \leftarrow Y \& U_r$ ,  $L_r \leftarrow L_r \oplus Y'$ ,  $Y \leftarrow Y - Y'$ . Finally set  $L_1 \leftarrow L_1 \oplus Y$  and return to P3'.

**P5'**. [Done?] If  $Z \neq -1$ , terminate successfully: One solution is given by the bits  $(X_1 \& B) \dots (X_n \& B)$ , where  $B = \bar{Z} \& (Z + 1)$ . Otherwise, if  $t > N$ , terminate unsuccessfully. Otherwise return to P2'. ■

The shenanigans in step P4' have the effect of flipping the offending bits of each literal with probability  $1/k$ , thus distributing the 1s of  $Y$  in an unbiased fashion.

**301.** In practice we can assume that all clauses have limited size, so that (say)  $k \leq 4$  in step P4'. The clauses can also be sorted by size.

A traditional random number generator produces bits  $U_2$ ; and one can use  $U_2 \& U_2$  to get  $U_4$ . The method of exercise 3.4.1-25 can be used for other cases; for example,

$$U_2 \& (U_2 \mid (U_2 \& (U_2 \mid (U_2 \& (U_2 \mid (U_2 \& (U_2 \mid (U_2 \& U_2))))))))$$

is a sufficiently close approximation to  $U_3$ . The random numbers needed in step P1' must be of top quality; but those used in step P4' don't have to be especially accurate, because most of their bits are irrelevant. We can precompute the latter, making tables of  $2^d$  values for each of  $U_2, U_3, U_4$ , and running through them cyclically by means of table indices U2P, U3P, U4P as in the code below, where  $UMASK = 2^{d+3} - 1$ . The values of U2P, U3P, and U4P should be initialized to (truly) random bits whenever step P2' starts a new pass over the clauses.

branchless computation  
ZSEV  
mone  
swapping to the front  
Knuth

Here is sample code for the inner loop, step P4', for clauses with  $k = 3$ . The octabyte in memory location  $L + 8(i-1)$  is the address in memory where  $X_h$  is stored, plus 1 if it should be complemented; for example, if  $l_2$  is  $\bar{x}_3$ , the address  $X + 3 \times 8 + 1$  will be in location  $L + 8$ , where  $L$  is a global register. Register `mone` holds the constant  $-1$ .

```

LDOU $1,L,0    addr(L1)    XOR   $9,$6,$0    $\bar{L}_3$         STOU  $6,$3,0    $|L_3| \oplus Y'$ 
LDOU $4,$1,0    $|L_1|$        AND   $7,$7,$8        SUBU  $7,$7,$0
LDOU $2,L,8    addr(L2)    AND   $7,$7,$9   Y        LDOU  $0,U2,U2P
LDOU $5,$2,0    $|L_2|$        OR    Z,Z,$7        Z | Y      ADD   U2P,U2P,8
LDOU $3,L,16   addr(L3)    AND   $0,$7,1   Y & 1     AND   U2P,U2P,UMASK
LDOU $6,$3,0    $|L_3|$        ADD   T,T,$0    new t     AND   $0,$0,$7    $U_2 \& Y$ 
ZSEV $0,$1,mone    LDOU  $0,U3,U3P        XOR   $5,$5,$0
XOR   $7,$4,$0  $\bar{L}_1$        ADD   U3P,U3P,8        STOU  $5,$2,0    $|L_2| \oplus Y'$ 
ZSEV $0,$2,mone    AND   U3P,U3P,UMASK        SUBU  $7,$7,$0
XOR   $8,$5,$0  $\bar{L}_2$        AND   $0,$0,$7    $U_3 \& Y$     XOR   $4,$4,$7
ZSEV $0,$3,mone    XOR   $6,$6,$0        STOU  $4,$1,0    $|L_1| \oplus Y$  ■

```

**302.** Assume that literals are represented internally as in Algorithm A, and that all clauses have strictly distinct literals. An efficient implementation actually requires more arrays than are stated in the text: We need to know exactly which clauses contain any given literal, just as we need to know the literals of any given clause. And we also need a (small) array  $b_0 \dots b_{k-1}$  to list the best candidate literals in step W4:

**W4.** [Choose  $l$ .] Set  $c \leftarrow \infty$ ,  $j \leftarrow 0$ , and do the following while  $j < k$ : Set  $j \leftarrow j + 1$ ,  $l \leftarrow l_j$ ; and if  $c_{|l|} < c$ , set  $c \leftarrow c_{|l|}$ ,  $b_0 \leftarrow l$ ,  $s \leftarrow 1$ ; or if  $c_{|l|} = c$ , set  $b_s \leftarrow l$ ,  $s \leftarrow s + 1$ . Then if  $c = 0$ , or if  $c \geq 1$  and  $U \geq p$ , set  $l \leftarrow b_{\lfloor sU \rfloor}$ ; otherwise set  $l \leftarrow l_{\lfloor kU \rfloor + 1}$ . (Each random fraction  $U$  is independent of the others.)

**W5.** [Flip  $l$ .] Set  $s \leftarrow 0$ . For each  $j$  such that  $C_j$  contains  $l$ , make clause  $C_j$  happier as follows: Set  $q \leftarrow k_j$ ,  $k_j \leftarrow q + 1$ ; and if  $q = 0$ , set  $s \leftarrow s + 1$  and delete  $C_j$  from the  $f$  array (see below); or if  $q = 1$ , decrease the cost of  $C_j$ 's critical variable (see below). Then set  $c_{|l|} \leftarrow s$  and  $x_{|l|} \leftarrow \bar{x}_{|l|}$ . For each  $j$  such that  $C_j$  contains  $\bar{l}$ , make clause  $C_j$  sadder as follows: Set  $q \leftarrow k_j - 1$ ,  $k_j \leftarrow q$ ; and if  $q = 0$ , insert  $C_j$  into the  $f$  array (see below); or if  $q = 1$ , increase the cost of  $C_j$ 's critical variable (see below). Set  $t \leftarrow t + 1$  and return to W2. ■

To insert  $C_j$  into  $f$ , we set  $f_r \leftarrow j$ ,  $w_j \leftarrow r$ , and  $r \leftarrow r + 1$  (as in step W1). To delete it, we set  $h \leftarrow w_j$ ,  $r \leftarrow r - 1$ ,  $f_h \leftarrow f_r$ ,  $w_{f_r} \leftarrow h$ .

Whenever we want to update the cost of  $C_j$ 's critical variable in step W5, we know that  $C_j$  has exactly one true literal. Thus, if the literals of  $C_j$  appear sequentially in a master array  $M$ , it's easy to locate the critical variable  $x_{|M_i|}$ : We simply set  $i \leftarrow \text{START}(j)$ ; then while  $M_i$  is false (namely while  $x_{|M_i|} = M_i \& 1$ ), set  $i \leftarrow i + 1$ .

A slight refinement is advantageous when we will be increasing  $c_{|M_i|}$ : If  $i \neq \text{START}(j)$ , swap  $M_{\text{START}(j)} \leftrightarrow M_i$ . This change significantly shortens the search when  $c_{|M_i|}$  is subsequently decreased. (In fact, it reduced the total running time by more than 5% in the author's experiments with random 3SAT problems.)

**303.** In this case  $D = 3 - z - z^2 = A/z$ , and we have  $g'(1) = 3$ ,  $g''(1) = 73/4$ . Thus  $\text{mean}(g) = 3$  and  $\text{var}(g) = 12.25 = 3.5^2$ .

random walk

**304.** If  $\nu x = x_1 + \cdots + x_n = a$ , there are  $a(n-a)$  unsatisfied clauses; hence there are two solutions,  $0 \dots 0$  and  $1 \dots 1$ . If  $x_1 \dots x_n$  isn't a solution, Algorithm P will change  $a$  to  $a \pm 1$ , each with probability  $\frac{1}{2}$ . Thus the probability generating function  $g_a$  for future flips is 1 when  $a = 0$  or  $a = n$ , otherwise it is  $z(g_{a-1} + g_{a+1})/2$ . And the overall generating function is  $g = \sum_a \binom{n}{a} g_a / 2^n$ . Clearly  $g_a = g_{n-a}$ .

Exercise MPR-105 determines  $g_a$  and proves that the mean number of flips,  $g'_a(1)$ , is  $a(n-a)$  for  $0 \leq a \leq n$ . Thus  $g'(1) = 2^{-n} \sum_{a=0}^n \binom{n}{a} g'_a(1) = \frac{1}{2} \binom{n}{2}$ .

Turning now to Algorithm W, again with  $x_1 + \cdots + x_n = a$ , the cost of  $x_i$  is  $a-1$  when  $x_i = 1$ ,  $n-a-1$  when  $x_i = 0$ . Therefore  $g_1 = g_{n-1} = z$  in this case. And for  $2 \leq a \leq n-2$ , we will move closer to a solution with probability  $q$  and farther from a solution with probability  $p$ , where  $p+q=1$  and  $p = p'/2 \leq 1/2$ ; here  $p'$  is the greed-avoidance parameter of Algorithm W. Thus for  $2 \leq a \leq n/2$  we have  $g_a = g_{n-a} = z(qg_{a-1} + pg_{a+1})$ .

If  $p' = 0$ , so that the walk is 100% greedy, Algorithm W zooms in on the solution, with  $g_a = z^a$ . Exercise 1.2.6-68 with  $p = 1/2$  tells us that  $g'(1) = n/2 - m \binom{n}{m} / 2^n = n/2 - \sqrt{n/2\pi} + O(1)$  in that case. On the other hand if  $p' = 1$ , so that the walk is greedy only when  $a = 1$  or  $a = n-1$ , we're almost in the situation of Algorithm P but with  $n$  decreased by 2. Then  $g'(1) = 2^{-n} \sum_{a=1}^{n-1} \binom{n}{a} (1 + (a-1)(n-2) - (a-1)^2) = n(n-5)/4 + 2 + (2n-4)/2^n$ ; greed triumphs.

What happens as  $p'$  rises from 0 to 1? Let's decrease  $n$  by 2 and use the rule  $g_a = z(qg_{a-1} + pg_{a+1})$  for  $1 \leq a \leq n/2$ , so that the calculations are similar to those we did for Algorithm P but with  $p$  now  $\leq 1/2$  instead of  $p = 1/2$ . Functions  $t_m$  and  $u_m$  can be defined as above; but  $g_a = (qz)^a t_{m-a} / t_m$ , the new recurrence is  $t_{m+1} = t_m - pqz^2 t_{m-1}$ , and  $t_0 = 1/q$ ,  $u_0 = 1/(qz)$ . These functions are polynomials in  $p$ ,  $q$ , and  $z$ , whose coefficients are binomial coefficients: In the notation of exercise 1.2.9-15, for  $m > 0$  we have  $t_m = G_{m-1}(-pqz^2) - pz^2 G_{m-2}(-pqz^2)$  and  $u_m = G_{m-1}(-pqz^2) - pz G_{m-2}(-pqz^2)$ , so

$$T(w) = \frac{1-pw}{q(1-w+pqz^2w^2)}; \quad U(w) = \frac{1-(1-qz)w}{qz(1-w+pqz^2w^2)}.$$

Consequently  $t'_m(1)/t_m(1) = 2pq(1-(p/q)^m)/(q-p)^2 - 2pm/(q-p)$  and  $u'_m(1)/u_m(1) = (2p-(p/q)^m q)/(q-p)^2 - 2p(m-\frac{1}{2})/(q-p)$ ;  $g'_a(1) = a/(q-p) - 2pq((q/p)^a - 1)/(q-p)^2$  for  $0 \leq a \leq n/2$  when  $n$  is even,  $a/(q-p) - q((q/p)^a - 1)/(q-p)^2$  when  $n$  is odd. The overall totals when  $n = 1000$  and  $p' = (.001, .01, .1, .5, .9, .99, .999)$  are respectively  $\approx (487.9, 492.3, 541.4, 973.7, 4853.4, 44688.2, 183063.4)$ .

**305.** That little additional clause reverses the picture! Now there's only one solution, and greediness fails badly when  $\nu x > n/2$  because it keeps trying to move  $x$  away from the solution. To analyze the new situation in detail, we need  $3(n-1)$  generating functions  $g_{ab}$ , where  $a = x_1 + x_2$  and  $b = x_3 + \cdots + x_n$ . The expected number of flips will be  $g'(1)$ , where  $g = 2^{-n} \sum_{a=0}^n \sum_{b=0}^{n-2} \binom{2}{a} \binom{n-2}{b} g_{ab}$ .

The behavior of Algorithm P is ambiguous, because the unsatisfied clause found in step P2 depends on the clause ordering. The most favorable case arises when  $a = 2$ , because we can decrease  $a$  to 1 by working on the special clause  $\bar{x}_1 \vee \bar{x}_2$ . Any other clause is equally likely to increase or decrease  $a+b$ . So the best-case generating functions maximize the chance of reaching  $a = 2$ :  $g_{00} = 1$ ,  $g_{01} = \frac{z}{2}(g_{00} + g_{11})$ ,  $g_{02} = \frac{z}{2}(g_{01} + g_{12})$ ,  $g_{10} = \frac{z}{2}(g_{00} + g_{20})$ ,  $g_{11} = \frac{z}{2}(g_{10} + g_{21})$ ,  $g_{12} = \frac{z}{2}(g_{11} + g_{22})$ , and  $g_{2b} = zg_{1b}$ . The solution has  $g_{1b} = (z/(2-z^2))^{b+1}$ ; and we find  $\text{mean}(g) = 183/32 = 5.71875$ .



The worst case arises whenever  $g_{20} \neq zg_{10}$  and  $g_{21} \neq zg_{11}$ ; for example we can take  $g_{20} = \frac{z}{2}(g_{10} + g_{21})$ ,  $g_{21} = \frac{z}{2}(g_{20} + g_{22})$ , together with the other seven equations from the best case. Then  $g_{01} = g_{10} = z(4 - 3z^2)/d$ ,  $g_{02} = g_{11} = g_{20} = z^2(2 - z^2)/d$ , and  $g_{12} = g_{21} = z^3/d$ , where  $d = 8 - 8z^2 + z^4$ . Overall,  $g = (1+z)^2(2-z^2)/(4d)$  and  $\text{mean}(g) = 11$ .

infinite loop  
geometric distribution  
memoryless

(This analysis can be extended to larger  $n$ : The worst case turns out to have  $g_{ab} = g_{a+b} = (z/2)^{a+b} t_{n-a-b}/t_n$ , in the notation of the previous exercise, giving  $n(3n-1)/4$  flips on average. The best case has  $g_{1b}$  as before; hence  $g'_{0b} = 3b+2-2^{1-b}$ ,  $g'_{1b} = 3b+3$ , and  $g'_{2b} = 3b+4$  when  $z = 1$ . The best average number of flips therefore turns out to be *linear*, with  $\text{mean}(g) = \frac{3}{2}n - \frac{8}{9}(3/4)^n$ .)

The analysis becomes more exciting, but trickier, when we use Algorithm W. Let  $p = p'/2$  and  $q = 1 - p$  as in the previous answer. Clearly  $g_{00} = 1$ ,  $g_{01} = g_{10} = zg_{00}$ ,  $g_{02} = \frac{z}{2}(g_{01} + g_{12})$ , and  $g_{22} = zg_{12}$ ; but the other four cases need some thought. We have

$$g_{11} = \frac{z}{4} \left( \left( \frac{1}{2} + q \right) (g_{01} + g_{10}) + g_{12} + 2pg_{21} \right),$$

since the costs for  $x_1x_2x_3x_4 = 1010$  are 1211 and the unsatisfied clauses are  $(\bar{x}_1 \vee x_4)$ ,  $(\bar{x}_3 \vee x_4)$ ,  $(\bar{x}_1 \vee x_2)$ ,  $(\bar{x}_3 \vee x_2)$ ; in the former two clauses we flip each literal equally often, but in the latter two we flip  $x_2$  with probability  $p$  and the other with probability  $q$ . A similar but simpler analysis shows that  $g_{21} = \frac{z}{4}(g_{11} + 3g_{22})$  and  $g_{20} = \frac{z}{5}(3g_{10} + 2g_{21})$ .

The most interesting case is  $g_{12} = \frac{z}{3}(pg_{02} + 2pg_{11} + 3gg_{22})$ , where the costs are 2122 and the problematic clauses are  $(\bar{x}_1 \vee x_2)$ ,  $(\bar{x}_3 \vee x_2)$ ,  $(\bar{x}_4 \vee x_2)$ . If  $p = 0$ , Algorithm W will always decide to flip  $x_2$ ; but then we'll be back in state 12 after the next flip.

Indeed, setting  $p = 0$  yields  $g_{00} = 1$ ,  $g_{01} = g_{10} = z$ ,  $g_{02} = \frac{1}{2}z^2$ ,  $g_{11} = \frac{3}{4}z^2$ ,  $g_{20} = \frac{3}{5}z^2 + \frac{3}{40}z^4$ ,  $g_{21} = \frac{3}{16}z^3$ , and  $g_{12} = g_{22} = 0$ . The weighted total therefore turns out to be  $g = (40 + 160z + 164z^2 + 15z^3 + 3z^4)/640$ . Notice that the greedy random walk never succeeds after making more than 4 flips, in this case; so we should set  $N = 4$  and restart after each failure. The probability of success is  $g(1) = 191/320$ . (This strategy is actually quite good: It succeeds after making an average of  $1577/382 \approx 4.13$  flips and choosing random starting values  $x_1x_2x_3x_4$  about  $320/191$  times.)

If  $p$  is positive, no matter how tiny, the success probability for  $N = \infty$  is  $g(1) = 1$ . But the denominator of  $g$  is  $48 - 48z^2 + 26pz^2 + 6pz^4 - 17p^2z^4$ , and we find that  $\text{mean}(g) = (1548 + 2399p - 255p^2)/(1280p - 680p^2) = (6192 + 4798p' - 255p'^2)/(2560p' - 680p'^2)$ . Taking  $p' = (.001, .01, .1, .5, .9, .99, .999)$  in this formula gives, respectively, the approximate values (2421.3, 244.4, 26.8, 7.7, 5.9, 5.7, 5.7).

(Calculations for  $n = 12$  show that  $g$  is a polynomial of degree 8 when  $p = 0$ , with  $g(1) \approx .51$  and  $g'(1) \approx 2.40$ . Thus, setting  $N = 8$  yields success after about 16.1 flips and 1.95 initializations. When  $p > 0$  we have  $g'(1) \approx 1.635p^{-5} + O(p^{-4})$  as  $p \rightarrow 0$ , and the seven values of  $p'$  considered above yield respectively  $(5 \times 10^{16}, 5 \times 10^{11}, 5 \times 10^6, 1034.3, 91.1, 83.89, 83.95)$  flips—surprisingly *not* monotone decreasing in  $p'$ . These WalkSAT statistics can be compared with 17.97 to 105 flips for Algorithm P.)

**306.** (a) Since  $l(N) = E_N + (1 - q_N)(N + l(N))$ , we have  $q_N l(N) = E_N + N - Nq_N = p_1 + 2p_2 + \dots + Np_N + Np_{N+1} + \dots + Np_\infty = N - (q_1 + \dots + q_{N-1})$ .

(b) If  $N = m + k$  and  $k \geq 0$  we have  $E_N = m^2/n$ ,  $q_1 + \dots + q_{N-1} = km/n$ , and  $q_N = m/n$ ; hence  $l(N) = n + k(n - m)/m$ .

(c) If  $N \leq n$ ,  $l(N) = (N - \binom{N}{2}/n)/(N/n) = n - \frac{N-1}{2}$ ; otherwise  $l(N) = l(n) = \frac{n+1}{2}$ .

(d) From  $q_N = p_1(N - q_1 - \dots - q_{N-1})$  and  $q_{N+1} = p_1(N + 1 - q_1 - \dots - q_N)$  we deduce  $p_{N+1} = p_1(1 - q_N)$ , hence  $1 - q_{N+1} = (1 - p_1)(1 - q_N)$ . So it's a geometric distribution, with  $p_t = p(1 - p)^{t-1}$  for  $t \geq 1$ . (The fact that  $l(1) = l(2) = \dots$  is called the "memoryless property" of the geometric distribution.)

(e) Choose  $p_1, \dots, p_n$  arbitrarily, with  $q_n = p_1 + \dots + p_n \leq 1$ . Then, arguing as in (d),  $p_{n+1}, p_{n+2}, \dots$  are defined by  $1 - q_N = (1 - 1/l(n))^{N-n}(1 - q_n)$  for  $N \geq n$ .

(f) Since  $l(n+1) - l(n) = (n - (q_1 + \dots + q_n))(1 - 1/q_n) \leq 0$ , we must have  $q_n = 1$  and  $l(n) = l(n+1)$ . (The case  $l(n) < l(n+1)$  is impossible.)

(g) Let  $x = p_1$  and  $y = p_2$ . By part (f), the conditions are equivalent to  $0 < x \leq x+y < 1$  and  $x(3-2x-y) > 1$ . Hence  $0 < (2x-1)(1-x) - xy \leq (2x-1)(1-x)$ ; we get the general solution by first choosing  $\frac{1}{2} < x < 1$ , then  $0 \leq y < (2x-1)(1-x)/x$ .

(h) If  $N^* = \infty$  and  $l(n) < \infty$ , we can find  $n'$  with  $q_{n'}l(n') = p_1 + 2p_2 + \dots + n'p_{n'} + n'p_{n'+1} + \dots + n'p_\infty > l(n)$ . Hence  $l(N) \geq q_N l(N) \geq q_{n'} l(n') > l(n)$  for all  $N \geq n'$ .

(i) We have  $q_{n+k} = k/(k+1)$  for  $k \geq 0$ ; hence  $l(n+k) = (k+1)(n+H_k)/k$ . The minimum occurs when  $l(n+k) \approx l(n+k-1)$ , namely when  $n \approx k - H_k$ ; thus  $k = n + \ln n + O(1)$ . For example, the optimum cutoff value when  $n = 10$  is  $N^* = 23$ . (Notice that  $E_\infty = \infty$ , yet  $l = l(N^*) \approx 14.194$  in this case.)

(j) Let  $p_t = \lfloor t > 1 \rfloor / 2^{t-1}$ . Then  $l(N) = (3 - 2^{2-N}) / (1 - 2^{1-N})$  decreases to 3.

(k) Clearly  $l \leq L$ . For  $N \leq L$  we have  $l(N) = (N - (q_1 + \dots + q_{N-1})) / q_N \geq (N - (1 + \dots + (N-1))) / L = (N-1) / L = L - (N-1) / 2 \geq (L+1) / 2$ . And for  $N = \lfloor L \rfloor + k + 1$ , similarly,  $l(N) \geq N - (1 + \dots + \lfloor L \rfloor + kL) / L = \lfloor L + 1 \rfloor (1 - \lfloor L \rfloor / (2L)) \geq (L+1) / 2$ .

**307.** (a)  $EX = E_{N_1} + (1 - q_{N_1})(N_1 + EX')$ , where  $X'$  is the number of steps for the sequence  $(N_2, N_3, \dots)$ . For numerical results, start with  $j \leftarrow 0, s \leftarrow 0, \alpha \leftarrow 1$ ; then, while  $\alpha > \epsilon$ , set  $j \leftarrow j + 1, \alpha \leftarrow (1 - q_{N_j})\alpha$ , and  $s \leftarrow s + E_{N_j} + \alpha N_j$ . (Here  $\epsilon$  is tiny.)

(b) Let  $P_j = (1 - q_{N_1}) \dots (1 - q_{N_{j-1}}) = \Pr(X > T_j)$ , and note that  $P_j \leq (1 - p_n)^{j-1}$  where  $n = \min\{t \mid p_t > 0\}$ . Since  $q_N l(N) = E_N + (1 - q_N)N$ , we have

$$\begin{aligned} EX &= q_{N_1} l(N_1) + (1 - q_{N_1})(q_{N_2} l(N_2) + (1 - q_{N_2})(q_{N_3} l(N_3) + \dots)) \\ &= \sum_{j=1}^{\infty} P_j q_{N_j} l(N_j) = \sum_{j=1}^{\infty} (P_j - P_{j+1}) l(N_j). \end{aligned}$$

(c)  $EX \geq \sum_{j=1}^{\infty} (P_j - P_{j+1}) l(N^*) = l$ .

(d) We can assume that  $N_j \leq n$  for all  $j$ ; otherwise the strategy would do even worse. For the hint, let  $\{N_1, \dots, N_r\}$  contain  $r_m$  occurrences of  $m$ , for  $1 \leq m \leq n$ , and suppose  $t_m = r_m + \dots + r_n$ . If  $t_m < n/(2m)$ , the probability of failure would be  $(1 - m/n)^{t_m} \geq 1 - t_m m/n > 1/2$ . Hence we have  $t_m \geq n/(2m)$  for all  $m$ , and  $N_1 + \dots + N_r = t_1 + \dots + t_n \geq nH_n/2$ .

Now there's some  $m$  such that the first  $r-1$  trials fail on  $p^{(m)}$  with probability  $> \frac{1}{2}$ . For this  $m$  we have  $EX > \frac{1}{2}(N_1 + \dots + N_{r-1}) \geq \frac{1}{2}(N_1 + \dots + N_r - n)$ .

**308.** (a)  $2^{a+1} - 1$ ; and we also have  $S_{2^a+b} = S_{b+1}$  for  $0 \leq b < 2^a - 1$  (by induction).

(b) The sequence  $(u_n, v_n)$  in (131) has  $1 + \rho k$  entries with  $u_n = k$ ; and  $\rho 1 + \dots + \rho n = n - \nu n$  by Eq. 7.1.3-(61). From the double generating function  $g(w, z) = \sum_{n \geq 0} w^{\nu n} z^n = (1+wz)(1+wz^2)(1+wz^4)(1+wz^8) \dots$  we deduce that  $\sum_{k \geq 0} z^{2k+1-\nu k} = zg(z^{-1}, z^2)$ .

(c)  $\{n \mid S_n = 2^a\} = \{2^{a+1}k + 2^{a+1} - 1 - \nu k \mid k \geq 0\}$ ; hence  $\sum_{n \geq 0} z^n [S_n = 2^a] = z^{2^{a+1}-1} g(z^{-1}, z^{2^{a+1}}) = z^{2^{a+1}-1} (1 + z^{2^{a+1}-1}) (1 + z^{2^{a+2}-1}) (1 + z^{2^{a+3}-1}) \dots$

(d) When  $2^a$  occurs for the  $2^b$ th time, we've had  $2^{a+b-c} - [c > a]$  occurrences of  $2^c$ , for  $0 \leq c \leq a+b$ . Consequently  $\Sigma(a, b, 1) = (a+b-1)2^{a+b} + 2^{a+1}$ .

(e) The exact value is  $\sum_{c=0}^{a+b} 2^{a+b-c} 2^c + \sum_{c=1}^{\rho k} 2^{a+b+c}$ ; and  $\rho k \leq \lambda k = \lfloor \lg k \rfloor$ .

(f) The stated formula is  $E \min_k \{\Sigma(a, b, k) \mid \Sigma(a, b, k) \geq X\}$ , if we penalize the algorithm so that it *never* succeeds unless it is run with the particular cutoff  $N = 2^a$ .

(g) We have  $Q \leq (1 - q_t)^{2^b} \leq (1 - q_t)^{1/q_t} < e^{-1}$ ; hence  $EX < (a + b - 1)2^{a+b} + 2^{a+1} + \sum_{k=1}^{\infty} (a + b + 2k - 1)2^{a+b}e^{-k} = 2^{a+b}((a + b)e/(e - 1) + e(3 - e)/(e - 1)^2 + 2^{1-b})$ . Furthermore we have  $2^{a+b} < 8l - 4l[b = 0]$ , by exercise 306(k).

flush  
Fibonacci ruler function  
ruler of Fibonacci  
Cohen

**309.** No—far from it. If Algorithm C were to satisfy the hypotheses of exercise 306, it would have to do *complete* restarts: It would not only have to flush *every* literal from the trail, it would also have to forget all the clauses that it has learned, and reinitialize the random heap. [But reluctant doubling appears to work well also outside of Vegas.]

**310.** A method analogous to (131) can be used: Let  $(u'_1, v'_1) = (1, 0)$ ; then define  $(u'_{n+1}, v'_{n+1}) = (u'_n \& -u'_n = 1 \ll v'_n? (\text{succ}(u'_n), 0): (u'_n, v'_n + 1))$ . Here ‘succ’ is the Fibonacci-code successor function that is defined by six bitwise operations in answer 7.1.3–158. Finally, let  $S'_n = F_{v'_n+2}$  for  $n \geq 1$ . (This sequence  $\langle S'_n \rangle$ , like  $\langle S_n \rangle$ , is “nicely balanced”; hence it is universal as in exercise 308. For example, when  $F_a$  appears for the first time, there have been exactly  $F_{a+2-c}$  occurrences of  $F_c$ , for  $2 \leq c \leq a$ .)

**311.** Because  $\langle R_n \rangle$  does surprisingly well in these tests, it seems desirable to consider also its Fibonacci analog: If  $f_n = \text{succ}(f_{n-1})$  is the binary Fibonacci code for  $n$ , we can call  $\langle \rho' n \rangle = \langle \rho f_n \rangle = (0, 1, 2, 0, 3, 0, 1, 4, 0, \dots)$  the “Fibonacci ruler function,” and let  $\langle R'_n \rangle = (1, 2, 3, 1, 5, 1, 2, 8, 1, \dots)$  be the “ruler of Fibonacci,” where  $R'_n = F_{2+\rho' n}$ .

The results  $(E_S, E_{S'}, E_R, E_{R'})$  for  $m = 1$  and  $m = 2$  are respectively  $(315.1, 357.8, 405.8, 502.5)$  and  $(322.8, 284.1, 404.9, 390.0)$ ; thus  $S$  beats  $S'$  beats  $R$  beats  $R'$  when  $m = 1$ , while  $S'$  beats  $S$  beats  $R'$  beats  $R$  when  $m = 2$ . The situation is, however, reversed for larger values of  $m$ :  $R$  beats  $R'$  beats  $S$  beats  $S'$  when  $m = 90$ , while  $R'$  beats  $R$  beats  $S'$  beats  $S$  when  $m = 89$ .

In general, the reluctant methods shine for small  $m$ , while the more “aggressive” ruler methods forge ahead as  $m$  grows: When  $n = 100$ ,  $S$  beats  $R$  if and only if  $m \leq 13$ , and  $S'$  beats  $R'$  if and only if  $m \leq 12$ . The doubling methods are best when  $m$  is a power of 2 or slightly less; the Fibonacci methods are best when  $m$  is a Fibonacci number or slightly less. The worst cases occur at  $m = 65 = 2^6 + 1$  for  $S$  and  $R$  (namely 1402.2 and 845.0); they occur at  $m = 90 = F_{11} + 1$  for  $S'$  and  $R'$  (namely 1884.8 and 805.9).

**312.**  $T(m, n) = m + b2^b h_0(\theta)/\theta + 2^b g(\theta)$ , where  $b = \lceil \lg m \rceil$ ,  $\theta = 1 - m/n$ ,  $h_a(z) = \sum_n z^n [S_n = 2^a]$ , and  $g(z) = \sum_{n \geq 1} S_n z^n = \sum_{a \geq 0} 2^a h_a(z)$ .

**313.** If  $l$  is flipped, the number of unsatisfied clauses increases by the cost of  $|l|$  and decreases by the number of unsatisfied clauses that contain  $l$ ; and the latter is at least 1.

Consider the following interesting clauses, which have the unique solution 0000:

$$x_1 \vee \bar{x}_2, \quad \bar{x}_1 \vee x_2, \quad x_2 \vee \bar{x}_3, \quad \bar{x}_2 \vee x_3, \quad x_3 \vee \bar{x}_4, \quad \bar{x}_3 \vee x_4, \quad \bar{x}_1 \vee \bar{x}_4.$$

“Uphill” moves  $1011 \mapsto 1111$  and  $1101 \mapsto 1111$  are forced; also  $0110 \mapsto 1110$  or  $0111$ .

**314.** (Solution by Bram Cohen, 2012.) Consider the 10 clauses  $\bar{1}234567$ ,  $\bar{1}2\bar{3}4567$ ,  $12345$ ,  $12346$ ,  $12347$ ,  $\bar{1}2\bar{3}4$ ,  $\bar{1}2\bar{3}5$ ,  $\bar{1}2\bar{3}6$ ,  $\bar{1}245$ ,  $\bar{1}246$ , and 60 more obtained by the cyclic permutation  $(1234567)$ . All binary  $x = x_1 \dots x_7$  with weight  $\nu x = 2$  have cost-free flips leading to weight 3, but no such flips to weight 1. Since the only solution has weight 0, Algorithm W loops forever whenever  $\nu x > 1$ . (Is there a smaller example?)

**315.** Any value with  $0 \leq p < 1/2$  works, since each graph component is either  $K_1$  or  $K_2$ .

**316.** No;  $\max \theta(1 - \theta)^d$  for  $0 \leq \theta < 1$  is  $d^d/(d + 1)^{d+1}$ , when  $\theta = 1/(d + 1)$ . [But Theorem J for  $d > 2$  is a consequence of the improved Theorem L in exercise 356(c).]

**317.** Number the vertices so that the neighbors of vertex 1 are  $2, \dots, d'$ , and let  $G_j = G \setminus \{1, \dots, j\}$ . Then  $\alpha(G) = \alpha(G_1) - \Pr(A_1 \cap \bar{A}_2 \cap \dots \cap \bar{A}_m)$ , and the latter probability is  $\leq \Pr(A_1 \cap \bar{A}_{d'+1} \cap \dots \cap \bar{A}_m) = \Pr(A_1 | \bar{A}_{d'+1} \cap \dots \cap \bar{A}_m) \alpha(G_{d'}) \leq p \alpha(G_{d'})$ .

Let  $\rho = (d - 1)/d$ . By induction we have  $\alpha(G_j) > \rho\alpha(G_{j+1})$  for  $1 \leq j < d'$ , because vertex  $j + 1$  has degree  $< d$  in  $G_j$ . If  $d' = 1$  then  $\alpha(G) \geq \alpha(G_1) - p\alpha(G_1) > \rho\alpha(G_1) > 0$ . Otherwise if  $d' \leq d$ ,  $\alpha(G) \geq \alpha(G_1) - p\alpha(G_{d'}) > \alpha(G_1) - p\rho^{1-d'}\alpha(G_1) \geq \alpha(G_1) - p\rho^{1-d}\alpha(G_1) = \rho\alpha(G_1) > 0$ . Otherwise we must have  $d' = d + 1$ , with vertex 1 of degree  $d$ , and  $\alpha(G) > \alpha(G_1) - p\rho^{-d}\alpha(G_1) = \frac{d-2}{d-1}\alpha(G_1) \geq 0$ .

Möbius polynomial  
Chebyshev polynomials  
convex hull

**318.** Let  $f_n = M_G(p)$  where  $G$  is the graph of a complete  $t$ -ary tree with  $t^n$  leaves; thus  $G$  has  $t^k$  vertices at distance  $k$  from the root, for  $0 \leq k \leq n$ . Then

$$f_0 = 1 - p, \quad f_1 = (1 - p)^t - p, \quad \text{and} \quad f_{n+1} = f_n^t - pf_{n-1}^{t^2} \text{ for } n > 1.$$

By Theorem S, it suffices to show that  $f_n \leq 0$  for some  $n$ .

The key idea is to let  $g_0 = 1 - p$  and  $g_{n+1} = f_{n+1}/f_n^t = 1 - p/g_n^t$ . Assuming that  $g_n > 0$  for all  $n$ , we have  $g_1 < g_0$  and  $g_n - g_{n+1} = p/g_n^t - p/g_{n+1}^t > 0$  when  $g_{n+1} < g_n$ . Hence  $\lim_{n \rightarrow \infty} g_n = \lambda$  exists, with  $0 < \lambda < 1$ . Furthermore  $\lambda = 1 - p/\lambda^t$ , so that  $p = \lambda^t(1 - \lambda)$ . But then  $p \leq t^t/(t + 1)^{t+1}$  (see answer 316 with  $\theta = 1 - \lambda$ ).

[One must admit, however, that the limit is not often reached until  $n$  is extremely large. For example, even if  $t = 2$  and  $p = .149$ , we don't have  $f_n < 0$  until  $n = 45$ . Thus  $G$  must have at least  $2^{45}$  vertices before this value of  $p$  is too large for Lemma L.]

**319.** Let  $x = 1/(d - 1)$ . Since  $e^x > 1 + x = d/(d - 1)$ , we have  $e > (d/(d - 1))^{d-1}$ .

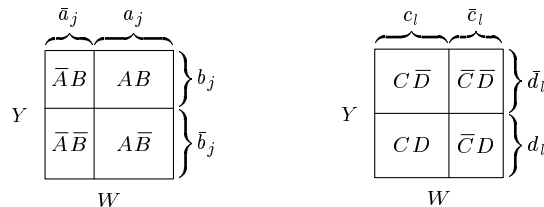
**320.** (a) Let  $f_m(p)$  be the Möbius polynomial when  $p_1 = \dots = p_m = p$ . Then we have  $f_m(p) = f_{m-1}(p) - pf_{m-2}(p)$ , and one can show by induction that  $f_m(1/(4 \cos^2 \theta)) = \sin((m + 2)\theta)/((2 \cos \theta)^{m+1} \sin \theta)$ . The threshold decreases to  $1/4$  as  $m \rightarrow \infty$ .

(b)  $1/(4 \cos^2 \frac{\pi}{2m})$ ; the Möbius polynomial  $g_m(p) = f_{m-1}(p) - pf_{m-3}(p)$  satisfies the same recurrence as  $f_m(p)$ , and equals  $2 \cos m\theta/(2 \cos \theta)^m$  when  $p = 1/(4 \cos^2 \theta)$ .

[In terms of the classical Chebyshev polynomials,  $g_m(p) = 2p^{m/2}T_m(1/(2\sqrt{p}))$  and  $f_m(p) = p^{(m+1)/2}U_{m+1}(1/(2\sqrt{p}))$ .]

**321.** Let  $\theta = (2 - \sqrt{2})/2$ ,  $\theta' = \theta(1 - \theta) = (\sqrt{2} - 1)/2$ , and  $c = (p - \theta)/(1 - \theta)$ . The method of answer 345 gives  $(\Pr(\overline{ABC}\overline{D}), \Pr(A\overline{BC}\overline{D}), \Pr(ABC\overline{D}), \Pr(A\overline{BC}D), \Pr(ABC\overline{D}), \Pr(AB\overline{C}D)) = (0, \theta'(1-c)^3, 2\theta'(1-c)^2c, \theta^2(1-c)^2 + 2\theta'(1-c)^3, \theta^2(1-c)c + 3\theta'(1-c)c^2, \theta^2c^2 + 4\theta'c^3)$ . Other cases are symmetric to these six. When  $p = 3/10$  the six probabilities are  $\approx (0, .20092, .00408, .08815, .00092, .00002)$ .

**322.** (a) Let  $a_j = \sum_i w_i [ij \in A]$ ,  $b_j = \sum_k y_k [jk \in B]$ ,  $c_l = \sum_k y_k [kl \in C]$ , and  $d_l = \sum_i w_i [li \in D]$ . Then when  $X = j$  and  $Z = l$ , the best way to allocate the events is



within  $W$  and  $Y$ . Hence  $\Pr(\overline{A} \cap \overline{B} \cap \overline{C} \cap \overline{D}) = \sum_{j,l} x_j z_l ((\overline{a}_j + \overline{d}_l) \div 1) ((\overline{b}_j + \overline{c}_l) \div 1)$ , which is zero if and only if we have  $a_j + d_l \geq 1$  or  $b_j + c_l \geq 1$  for all  $j$  and  $l$  with  $x_j z_l > 0$ .

(b) Since  $\sum_j x_j(a_j, b_j) = (p, p)$ , the point  $(p, p)$  lies in the convex hull of the points  $(a_j, b_j)$ . So there must be points  $(a, b) = (a_j, b_j)$  and  $(a', b') = (a_{j'}, b_{j'})$  such that the line from  $(a, b)$  to  $(a', b')$  intersects the region  $\{(x, y) \mid 0 \leq x, y \leq p\}$ ; in other words  $\mu a + (1 - \mu)a' \leq p$  and  $\mu b + (1 - \mu)b' \leq p$ . Similarly we can find  $c, d, c', d', v$ .

(c) Fact: If  $a \geq \frac{2}{3}$  and  $b' \geq \frac{2}{3}$ , then  $\mu = \frac{1}{2}$ ; hence  $a = b' = \frac{2}{3}$  and  $a' = b = 0$ . Notice also that there are 16 symmetries, generated by (i)  $a \leftrightarrow b, c \leftrightarrow d$ ; (ii)  $a \leftrightarrow a', b \leftrightarrow b', \mu \leftrightarrow 1 - \mu$ ; (iii)  $c \leftrightarrow c', d \leftrightarrow d', \nu \leftrightarrow 1 - \nu$ ; (iv)  $a \leftrightarrow d, b \leftrightarrow c, \mu \leftrightarrow \nu$ .

If  $c \leq c'$  and  $d \leq d'$ , or if  $c \leq \frac{1}{3}$  and  $d \leq \frac{1}{3}$ , we can assume (by symmetry) that the Fact applies; this gives a solution to all the constraints, with  $c = d = c' = d' = \frac{1}{3}$ .

For the remaining solutions we may assume that  $a, b' > \frac{1}{3} > a', b$ . Suppose the line from  $(a, b)$  to  $(a', b')$  intersects the line from  $(0, 0)$  to  $(1, 1)$  at the point  $(\alpha, \alpha)$ ; dividing  $a, b, a', b'$  by  $3\alpha$  gives a solution in which  $\mu a + (1 - \mu)a' = \mu b + (1 - \mu)b' = \frac{1}{3}$ . Similarly, we can assume that  $d, c' > \frac{1}{3} > d', c$  and that  $\nu c + (1 - \nu)c' = \nu d + (1 - \nu)d' = \frac{1}{3}$ . Consequently  $a + d \geq 1$  and  $b' + c' \geq 1$ . Symmetry also allows us to assume that  $a + d' \geq 1$ . In particular,  $a > \frac{2}{3}$ ; and, by the Fact,  $b' < \frac{2}{3}$ . So  $a' + d \geq 1, d > \frac{2}{3}, c' < \frac{2}{3}$ .

Now extend the lines that connect  $(a, b)$  to  $(a', b')$  and  $(c, d)$  to  $(c', d')$ , by increasing  $a, b', c', d$  while decreasing  $a', b, c, d'$ , until  $a' = 1 - d$  and  $a = 1 - d'$ , and until either  $a = 1$  or  $b = 0$ , and either  $d = 1$  or  $c = 0$ . The only solution of this kind with  $b' + c' \geq 1$  occurs when  $a = d = 1, a' = b = c = d' = 0, b' = c' = 1/2, \mu = \frac{1}{3}, \nu = \frac{2}{3}$ .

(d) For the first solution, we can let  $W, X, Y, Z$  be uniform on  $\{0, 1, 2\}, \{0, 1\}, \{0, 1, 2\}$ , and  $\{0\}$ , respectively; and let  $A = \{10, 20\}, B = \{11, 12\}, C = \{00\}, D = \{00\}$ . (For example,  $WXYZ = 1110$  gives event  $B$ .) The second solution turns out to be the same, but with  $(X, Y, Z, W)$  in place of  $(W, X, Y, Z)$ . Notice that the solution applies also to  $P_4$ , where the threshold is  $\frac{1}{3}$ . [See *STOC* **43** (2011), 242.]

**323.** *cbc*. In this simple case, we just eliminate all strings in which  $c$  is followed by  $a$ .

**324.** For  $1 \leq j \leq n$ , and for each  $v$  such that  $v = x_j$  or  $v \dashv x_j$ , let  $i \prec j$  for each  $i < j$  such that  $v = x_i$ . (If several values of  $i$  qualify, it suffices to consider only the largest one. Several authors have used the term “dependence graph” for this partial ordering.) The traces equivalent to  $\alpha$  correspond to the topological sortings with respect to  $\prec$ , because those arrangements of the letters are precisely the permutations that preserve the empilement.

In (136), for example, with  $x_1 \dots x_n = bcebfadc$ , we have  $1 \prec 2, 1 \prec 4, 2 \prec 4, 4 \prec 5, 3 \prec 6, 2 \prec 7, 3 \prec 7, 2 \prec 8, 4 \prec 8$ , and  $7 \prec 8$ . Algorithm 7.2.1.2V produces 105 solutions, 12345678 (*bcebfadc*) through 36127485 (*efcbdbca*).

**325.** Every such trace  $\alpha$  yields an acyclic orientation, if we let  $u \rightarrow v$  when  $u$  appears at a lower level in  $\alpha$ 's empilement. Conversely, the topological sortings of any acyclic orientation are all equivalent traces; so this correspondence is one-to-one. [See Ira M. Gessel, *Discrete Mathematics* **232** (2001), 119–130.]

**326.** True:  $x$  commutes with  $y$  if and only if  $y$  commutes with  $x$ .

**327.** Each trace  $\alpha$  is represented by its height  $h = h(\alpha) \geq 0$ , and by  $h$  linked lists  $L_j = L_j(\alpha)$  for  $0 \leq j < h$ . The elements of  $L_j$  are the letters on level  $j$  of  $\alpha$ 's empilement; these letters have disjoint territories, and we keep each list in alphabetic order so that the representation is unique. The canonical string representing  $\alpha$  is then  $L_0 L_1 \dots L_{h-1}$ . (For example, in (136) we have  $L_0 = be, L_1 = cf, L_2 = bd, L_3 = ac$ , and the canonical representation is *becfbdac*.) We also maintain the sets  $U_j = \bigcup \{T(a) \mid a \in L_j\}$  as bit vectors; in (136), for example, they are  $U_0 = \#36, U_1 = \#1b, U_2 = \#3c, U_3 = \#78$ .

To multiply  $\alpha$  by  $\beta$ , do the following for  $k = 0, 1, \dots, h(\beta) - 1$  (in that order), and for each letter  $b \in L_k(\beta)$  (in any order): Set  $j \leftarrow h(\alpha)$ ; then while  $j > 0$  and  $T(b) \& U_{j-1}(\alpha) = 0$ , set  $j \leftarrow j - 1$ . If  $j = h(\alpha)$ , set  $L_j(\alpha)$  empty,  $U_j(\alpha) \leftarrow 0$ , and  $h(\alpha) \leftarrow h(\alpha) + 1$ . Insert  $b$  into  $L_j(\alpha)$ , and set  $U_j(\alpha) \leftarrow U_j(\alpha) + T(b)$ .

adjacent pairs of letters, avoiding  
dependence graph  
partial ordering  
topological sortings  
empilement  
linear extensions, see Topological sorting  
Gessel  
canonical string

**328.** Do the following for  $k = h(\beta) - 1, \dots, 1, 0$  (in that order), and for each letter  $b \in L_k(\beta)$  (in any order): Set  $j \leftarrow h(\alpha) - 1$ ; while  $j > 0$  and  $T(b) \& U_j(\alpha) = 0$ , set  $j \leftarrow j - 1$ . Report failure if  $b$  isn't in  $L_j(\alpha)$ . Otherwise remove  $b$  from that list and set  $U_j(\alpha) \leftarrow U_j(\alpha) - T(b)$ ; if  $U_j(\alpha)$  is now zero, set  $h(\alpha) \leftarrow h(\alpha) - 1$ .

If there was no failure, the resulting  $\alpha$  is the answer.

**329.** Do the following for  $k = 0, 1, \dots, h(\alpha) - 1$  (in that order), and for each letter  $a \in L_k(\alpha)$  (in any order): Report failure if  $a$  isn't in  $L_0(\beta)$ . Otherwise remove  $a$  from that list, set  $U_0(\beta) \leftarrow U_0(\beta) - T(a)$ , and renormalize the representation of  $\beta$ .

Renormalization involves the following steps: Set  $j \leftarrow c - 1$ . While  $U_{j-1}(\beta) \neq 0$  and  $c \neq 0$ , terminate if  $j = h(\beta)$ ; otherwise set  $c \leftarrow 0$ ,  $j \leftarrow j + 1$ , and then, for each letter  $b$  in  $L_{j-1}(\beta)$  such that  $T(b) \& U_{j-2}(\beta) = 0$ , move  $b$  from  $L_{j-1}(\beta)$  to  $L_{j-2}(\beta)$  and set  $U_{j-2}(\beta) \leftarrow U_{j-2}(\beta) + T(b)$ ,  $U_{j-1}(\beta) \leftarrow U_{j-1}(\beta) - T(b)$ ,  $c \leftarrow 1$ . Finally, if  $U_{j-1}(\beta) = 0$ , set  $U_{i-1}(\beta) \leftarrow U_i(\beta)$  and  $L_{i-1}(\beta) \leftarrow L_i(\beta)$  for  $j \leq i < h(\beta)$ , then set  $h(\beta) \leftarrow h(\beta) - 1$ .

If there was no failure, the resulting  $\beta$  is the answer.

**330.** Let the territorial universe be  $V \cup E$ , the vertices plus edges of  $G$ , and let  $T(a) = \{a\} \cup \{\{a, b\} \mid a - b\}$ . [G. X. Viennot, in 1985, called this subgraph a *starfish*.] Alternatively, we can get by with just two elements in each set  $T(a)$  if and only if  $G = L(H)$  is the line graph of some other multigraph  $H$ . Then each vertex  $a$  of  $G$  corresponds to an edge  $u - v$  in  $H$ , and we can let  $T(a) = \{u, v\}$ .

[Notes: The smallest graph  $G$  that *isn't* a line graph is the "claw"  $K_{1,3}$ . Since sets of independent vertices in the line graph  $G$  are sets of disjoint edges in  $H$ , also called *matchings* of  $H$ , the Möbius polynomial of  $G$  is also known as the "matching polynomial" of  $H$ . Such polynomials are important in theoretical chemistry and physics. When all territories have  $|T(a)| \leq 2$ , all roots of the polynomial  $M_G^*(z)$  in (149) are real and positive, by exercise 341. But  $M_{\text{claw}}(z, z, z, z) = 1 - 4z + 3z^2 - z^3$  has complex roots  $\approx 0.317672$  and  $1.34116 \pm 1.16154i$ .]

**331.** If  $\alpha$  is a string with  $k > 0$  occurrences of the substring  $ac$ , there are  $2^k$  ways to decompose  $\alpha$  into factors  $\{a, b, c, ac\}$ , and the expansion includes  $+\alpha$  and  $-\alpha$  each exactly  $2^{k-1}$  times. Thus we're left with the sum of all strings that don't contain 'ac'.

**332.** No: If  $b$  commutes with  $a$  and  $c$ , but  $ac \neq ca$ , we're dealing with strings that contain no adjacent pairs  $ba$  or  $cb$ ; hence  $cab$  qualifies, but it's equivalent to the smaller string  $bca$ . [Certain graphs do define traces with the stated property, as we've seen in (135) and (136). Using the next exercise we can conclude that the property holds if and only if no three letters  $a < b < c$  have  $a \not\sim b$ ,  $b \not\sim c$ , and  $a - c$  in the graph  $G$  of clashes. Thus the letters can be arranged into a suitable linear order if and only if  $G$  is a cocomparability graph; see Section 7.4.2.]

**333.** To show that  $\sum_{\alpha \in A, \beta \in B} (-1)^{|\beta|} \alpha\beta = 1$ , let  $\gamma = a_1 \dots a_n$  be any nonempty string. If  $\gamma$  cannot be factored so that  $a_1 \dots a_k \in A$  and  $a_{k+1} \dots a_n \in B$ , then  $\gamma$  doesn't appear. Otherwise  $\gamma$  has exactly two such factorizations, one in which  $k$  has its smallest possible value and the other in which  $k$  is one greater; these factorizations cancel each other in the sum. [*Manuscripta Math.* **19** (1976), 239–241.]

**334.** Equivalently we want to generate all strings of length  $n$  on the alphabet  $\{1, \dots, m\}$  that satisfy the following criterion, which strengthens the adjacent-letter test of exercise 332: If  $1 \leq i < j \leq n$ ,  $x_i \not\sim x_j$ ,  $x_{i+1} \not\sim x_j$ ,  $\dots$ ,  $x_{j-1} \not\sim x_j$ , then  $x_i \leq x_j$ . [See A. V. Anisimov and D. E. Knuth, *Int. J. Comput. Inf. Sci.* **8** (1979), 255–260.]

**T1.** [Initialize.] Set  $x_0 \leftarrow 0$  and  $x_k \leftarrow 1$  for  $1 \leq k \leq n$ .

**T2.** [Visit.] Visit the trace  $x_1 \dots x_n$ .

Viennot  
starfish  
line graph  
claw  
matchings  
Möbius polynomial  
matching polynomial  
real roots of polynomials  
cocomparability graph  
Anisimov  
Knuth

- T3.** [Find  $k$ .] Set  $k \leftarrow n$ . While  $x_k = m$  set  $k \leftarrow k - 1$ . Terminate if  $k = 0$ .
- T4.** [Advance  $x_k$ .] Set  $x_k \leftarrow x_k + 1$  and  $j \leftarrow k - 1$ .
- T5.** [Is  $x_k$  valid?] If  $x_j > x_k$  and  $x_j \not\leftarrow x_k$ , return to T4. If  $j > 0$  and  $x_j < x_k$  and  $x_j \not\leftarrow x_k$ , set  $j \leftarrow j - 1$  and repeat this step.
- T6.** [Reset  $x_{k+1} \dots x_n$ .] While  $k < n$  do the following: Set  $k \leftarrow k + 1$ ,  $x_k \leftarrow 1$ ; while  $x_{k-1} > x_k$  and  $x_{k-1} \not\leftarrow x_k$ , set  $x_k \leftarrow x_k + 1$ . Then go back to T2. ■

MacMahon's Master Theorem  
 lexicographically smallest traces  
 multiset permutations  
 cocomparability graph  
 cograph  
 complete bipartite  
 complete  $k$ -partite graphs  
 convolution principle  
 binomial convolution  
 Bender  
 Goldman

**335.** Given such an ordering, we have  $M_G = \det(I - A)$ , where the entry in row  $u$  and column  $v$  of  $A$  is  $v[u \geq v$  or  $u \dashv v]$ . The determinant in the given example is

$$\det \begin{pmatrix} 1 & -b & -c & 0 & 0 & 0 \\ 0 & 1-b & 0 & -d & 0 & 0 \\ 0 & -b & 1-c & -d & -e & 0 \\ 0 & -b & -c & 1-d & 0 & -f \\ 0 & -b & -c & -d & 1-e & -f \\ 0 & -b & -c & -d & -e & 1-f \end{pmatrix} + \det \begin{pmatrix} -a & -b & -c & 0 & 0 & 0 \\ 0 & 1 & c & -d & 0 & 0 \\ 0 & 0 & 1 & -d & -e & 0 \\ 0 & 0 & 0 & 1-d & 0 & -f \\ 0 & 0 & 0 & -d & 1-e & -f \\ 0 & 0 & 0 & -d & -e & 1-f \end{pmatrix},$$

after expanding the first column, then subtracting the first row from all other rows in the right-hand determinant. Therefore this rule satisfies recurrence (142).

[The result also follows from MacMahon's Master Theorem, exercise 5.1.2-20, using the characterization of lexicographically smallest traces in answer 334. According to Theorem 5.1.2B, such traces are in one-to-one correspondence with multiset permutations whose two-line representation does not contain  $\begin{smallmatrix} v \\ u \end{smallmatrix}$  when  $v > u$  and  $v \dashv u$ . Is there a similar determinantal expression when  $G$  is *not* a cocomparability graph?]

**336.** (a) If  $\alpha$  is a trace for  $G$  and  $\beta$  is a trace for  $H$ , we have  $\mu_{G \oplus H}(\alpha\beta) = \mu_G(\alpha)\mu_H(\beta)$ . Hence  $M_{G \oplus H} = M_G M_H$ . (b) In this case  $\mu_{G-H}(\alpha\beta) = \mu_G(\alpha)$  if  $\beta = \epsilon$ ,  $\mu_H(\beta)$  if  $\alpha = \epsilon$ ; otherwise it's zero. Therefore  $M_{G-H} = M_G + M_H - 1$ .

[These rules determine  $M_G$  recursively whenever  $G$  is a cograph (see exercise 7-90). In particular, complete bipartite and  $k$ -partite graphs have simple Möbius series, exemplified by  $M_G = (1-a)(1-b)(1-c) + (1-d)(1-e) + (1-f) - 2$  when  $G = K_{3,2,1}$ .]

**337.** Substituting  $a_1 + \dots + a_k$  for  $a$  in  $M_G$  gives  $M_{G'}$ . (Each trace for  $G'$  is obtained by putting subscripts on the  $a$ 's of the traces for  $G$ .)

**338.** The proof of Theorem F needs only minor changes: We limit  $\alpha$  to traces that contain no elements of  $A$ , and we define  $\alpha'$  and  $\beta'$  by letting  $a$  be the smallest letter  $\notin A$  in the bottom level of  $\gamma$ 's emplacement. If  $\gamma$  has no such letter, there's only one factorization, with  $\alpha = \epsilon$ . Otherwise we pair up cancelling factorizations. [Incidentally, the sum of all traces whose *sinks* are in  $A$  must be written in the other order:  $M_G^{-1} M_{G \setminus A}$ .]

- 339.** (a) "Push down" on piece  $x_j$  and factor out what comes through the floor.
- (b) Factor out the pyramid for the smallest label, and repeat on what's left.
- (c) This is a general convolution principle for labeled objects [see E. A. Bender and J. R. Goldman, *Indiana Univ. Math. J.* **20** (1971), 753-765]. For example, when  $l = 3$  the number of ways to get a labeled trace of length  $n$  from three labeled pyramids is  $\sum_{i,j,k} \binom{n}{i,j,k} P_i P_j P_k / 3! = n! \sum_{i,j,k} (P_i / i!) (P_j / j!) (P_k / k!) / 3!$ , with  $i + j + k = n$  in both of these sums. We divide by  $3!$  so that the topmost pyramid labels will be increasing.
- (d) Sum the identity in (c) for  $l = 0, 1, 2, \dots$
- (e)  $T(z) = \sum_{n \geq 0} t_n z^n = 1 / M_G(z)$  by Theorem F, and  $P(z) = \sum_{n \geq 1} p_n z^n / n$ .

*Note:* If we retain the letter names, writing for example  $M_G(z) = 1 - (a + b + c)z + acz^2$  instead of  $M_G(z) = 1 - 3z + z^2$ , the formal power series  $-\ln M_G(z)$  exhibits the pyramids of length  $n$  in the coefficient of  $z^n$ , but only in the sense of *commutative* algebra (not

trace algebra). For example, the coefficient of  $z^3$  obtained from  $\sum_{k \geq 1} (1 - M_G(z))^k/k$  with trace algebra includes the nonpyramidal term  $bac/6$ .

**340.** Let  $w((i_1 \dots i_k)) = (-1)^{k-1} a_{i_1 i_2} a_{i_2 i_3} \dots a_{i_k i_1}$ ; thus  $w(\pi) = (-a_{13} a_{34} a_{42} a_{21}) (-a_{57} a_{75}) (a_{66})$  in the given example. The permutation polynomial is then  $\det A$ , by definition of the determinant. (And we get the *permanent*, if we omit the  $(-1)^{k-1}$ .)

**341.** The hint is true when  $n = 2$ , since the first involution polynomials are  $w_{11}x$  and  $w_{11}w_{22}x^2 - w_{12}$ . And there's a recurrence:  $W(S) = w_{ii}xW(S \setminus i) - \sum_{j \neq i} W(S \setminus \{i, j\})$ .

So we can prove the existence of  $n + 1$  roots  $s_1 < r_1 < \dots < r_n < s_{n+1}$  by induction: Let  $W_n(x)$  be the polynomial for  $\{1, \dots, n\}$ . Then  $W_{n+1}(x)$  is  $w_{(n+1)(n+1)}xW_n(x)$  minus  $n$  polynomials  $w_{(n+1)j}W(\{1, \dots, n\} \setminus j)$ , each with roots  $q_k^{(j)}$  that are nicely sandwiched between the roots of  $W_n$ . Furthermore  $q_{n-k}^{(j)} = -q_k^{(j)}$  and  $r_{n+1-k} = -r_k$ , for  $1 \leq k \leq n/2$ . It follows that  $W_{n+1}(r_n) < 0$ ,  $W_{n+1}(r_{n-1}) > 0$ , and so on, with  $(-1)^k W_{n+1}(r_{n+1-k}) > 0$  for  $1 \leq k \leq n/2$ . Moreover,  $W_{n+1}(0) = 0$  when  $n$  is even;  $(-1)^k W_{n+1}(0) > 0$  when  $n = 2k - 1$ ; and  $W_{n+1}(x) > 0$  for all large  $x$ . Hence the desired  $s_k$  exist. [See Heilmann and Lieb, *Physical Review Letters* **24** (1970), 1412.]

**342.** If we replace  $(i_1 \dots i_k)$  by  $a_{i_1 i_2} a_{i_2 i_3} \dots a_{i_k i_1}$  (as in answer 340, but without the  $(-1)^{k-1}$ ), then  $M_{G_n}$  becomes  $\det(I - A)$ . Replacing  $a_{ij}$  by  $a_{ij}x_j$  gives the determinant in MacMahon's Master Theorem. And if  $x_1 = \dots = x_n = x$ , we get the polynomial  $\det(I - xA)$ , whose roots are the reciprocals of the roots of  $A$ 's characteristic polynomial.

**343.** The formulas in answer 336 show that  $M_G(p_1, \dots, p_m)$  increases whenever any  $p_j$  decreases, with respect to a cograph  $G$ . The only graph on  $\leq 4$  vertices that isn't a cograph is  $P_4$  (see exercise 7-90); then  $M_G(p_1, p_2, p_3, p_4) = 1 - p_1 - p_2 - p_3 - p_4 + p_1p_3 + p_1p_4 + p_2p_4 = (1 - p_1)(1 - p_3 - p_4) - p_2(1 - p_4)$ . In this case also we can conclude that  $M_G(p_1, \dots, p_4) > 0$  implies  $(p_1, \dots, p_4) \in \mathcal{R}(G)$ . But when  $G = P_5$ , we find  $M_G(1 - \epsilon, 1 - \epsilon, \epsilon, 1 - \epsilon, 1 - \epsilon) > 0$  for  $0 \leq \epsilon < \phi^{-2}$ ; yet  $(1 - \epsilon, 1 - \epsilon, \epsilon, 1 - \epsilon, 1 - \epsilon)$  is never in  $\mathcal{R}(G)$  because  $M_G(0, 0, \epsilon, 1 - \epsilon, 1 - \epsilon) = -(1 - \epsilon)^2$ .

**344.** (a) If some minterm, say  $B_1 \overline{B_2} \overline{B_3} B_4$ , has negative "probability," then  $p_1 p_4 \times (1 - \pi_2 - \pi_3 + \pi_{23}) < 0$ ; hence  $M_G(0, p_2, p_3, 0) < 0$  violates the definition of  $\mathcal{R}(G)$ .

(b) In fact, more is true:  $\pi_{I \cup J} = \pi_I \pi_J$  when  $i \not\sim j$  for  $i \in I, j \in J$ , and  $I \cap J = \emptyset$ .

(c) It's  $M_G(p_1[1 \in J], \dots, p_m[m \in J])$ , by (140) and (141). This important fact, already implicit in the solution to part (a), implies that  $\beta(G|J) > 0$  for all  $J$ .

(d) Writing just ' $J$ ' for ' $G|J$ ', we shall prove that  $\alpha(i \cup J)/\beta(i \cup J) \geq \alpha(J)/\beta(J)$  for  $i \notin J$ , by induction on  $|J|$ . Let  $J' = \{j \in J \mid i \not\sim j\}$ . Then we have

$$\alpha(i \cup J) = \alpha(J) - \Pr\left(A_i \cap \bigcap_{j \in J} \overline{A}_j\right) \geq \alpha(J) - \Pr\left(A_i \cap \bigcap_{j \in J'} \overline{A}_j\right) \geq \alpha(J) - p_i \alpha(J'),$$

because of (133). Also  $\beta(i \cup J) = \beta(J) - p_i \beta(J')$ . Hence  $\alpha(i \cup J)\beta(J) - \alpha(J)\beta(i \cup J) \geq (\alpha(J) - p_i \alpha(J'))\beta(J) - \alpha(J)(\beta(J) - p_i \beta(J')) = p_i(\alpha(J)\beta(J') - \alpha(J')\beta(J))$ , which is  $\geq 0$  by induction since  $J' \subseteq J$ .

[This argument proves that Lemma L holds whenever  $(p_1, \dots, p_m)$  leads to a legitimate probability distribution with  $\beta(G) > 0$ ; hence such probabilities are in  $\mathcal{R}(G)$ .]

(e) By induction, we have  $\beta(i \cup J) = \beta(J) - \theta_i \beta(J') \prod_{i \sim j} (1 - \theta_j) \geq \beta(J) - \theta_i \beta(J') \prod_{j \in J \setminus J'} (1 - \theta_j) \geq (1 - \theta_i) \beta(J)$ , because  $\beta(J)/\beta(J') \geq \prod_{j \in J \setminus J'} (1 - \theta_j)$ .

**345.** (Solution by A. D. Scott and A. D. Sokal.) Set  $p'_j = (1 + \delta)p_j$  where  $\delta \leq 0$  is the slack of  $(p_1, \dots, p_m)$ . Then  $M_G(p'_1, \dots, p'_m) = 0$ , but it becomes positive if any  $p'_j$  is decreased. Define events  $B'_1, \dots, B'_m$  by the construction in exercise 344. Let  $C_1, \dots, C_m$  be independent binary random variables such that  $\Pr(C_j = 1) = q_j$ ,

permanent  
Heilmann  
Lieb  
determinant  
MacMahon's Master Theorem  
Scott  
Sokal  
slack



where  $(1 - p'_j)(1 - q_j) = 1 - p_j$ . Then the events  $B_j = B'_j \vee C_j$  satisfy the desired conditions:  $\Pr(B_i | \overline{B}_{j_1} \cap \dots \cap \overline{B}_{j_k}) = \Pr(B_i | \overline{B}'_{j_1} \cap \dots \cap \overline{B}'_{j_k}) = \Pr(B_i) = p_i$ ; and  $\Pr(B_1 \vee \dots \vee B_m) \geq \Pr(B'_1 \vee \dots \vee B'_m) = 1$ .

traces  
sources  
top-down algs  
bottom-up algs  
Scott  
Sokal

**346.** (a) By (144),  $K_{a,G}$  is the sum of all traces on the probabilities of  $G \setminus a$  whose sources are neighbors of  $a$ . Decreasing  $p_j$  doesn't decrease any trace.

(b) Suppose vertex  $a = 1$  has neighbors  $2, \dots, j$ . If we've recursively computed  $M_{G \setminus a^*}$  and  $M_{G \setminus a}$ , finding that  $(p_{j+1}, \dots, p_m) \in \mathcal{R}(G \setminus a^*)$  and  $(p_2, \dots, p_m) \in \mathcal{R}(G \setminus a)$ , then we know  $K_{a,G}$ ; and the monotonicity property in (a) implies that  $(p_1, \dots, p_m) \in \mathcal{R}(G)$  if and only if  $aK_{a,G} < 1$ .

The graph  $G = c \begin{array}{c} a \text{---} b \\ | \quad | \\ e \text{---} d \end{array}$  in exercise 335 can, for example, be processed as follows:

$$\begin{aligned} M_{abcdef} &= M_{bcdef} \left(1 - a \frac{M_{def}}{M_{bcdef}}\right) = (1 - a')(1 - b') \dots (1 - f'), & a' &= \frac{a}{(1 - b')(1 - c')}, \\ M_{bcdef} &= M_{cdef} \left(1 - b \frac{M_{cef}}{M_{cdef}}\right) = (1 - b')(1 - c') \dots (1 - f'), & b' &= \frac{b(1 - c'')}{(1 - c')(1 - d')}, \\ M_{cdef} &= M_{def} \left(1 - c \frac{M_{df}}{M_{def}}\right) = (1 - c')(1 - d')(1 - e')(1 - f'), & c' &= \frac{c}{(1 - d')(1 - e')}, \\ M_{cef} &= M_{ef} \left(1 - c \frac{M_{f}}{M_{ef}}\right) = (1 - c'')(1 - e')(1 - f'), & c'' &= \frac{c}{(1 - e')}, \\ M_{def} &= M_{ef} \left(1 - d \frac{M_e}{M_{ef}}\right) = (1 - d')(1 - e')(1 - f'), & d' &= \frac{d(1 - e'')}{(1 - e')(1 - f')}, \\ M_{ef} &= M_f \left(1 - e \frac{M_e}{M_f}\right) = (1 - e')(1 - f'), & e' &= \frac{e}{(1 - f')}, \\ M_e &= M_e \left(1 - e \frac{M_e}{M_e}\right) = (1 - e''), & e'' &= e, \\ M_f &= M_e \left(1 - f \frac{M_e}{M_e}\right) = (1 - f'), & f' &= f, \end{aligned}$$

with  $M_e = 1$ . (The equations on the left are derived top-down, then the equations on the right are evaluated bottom-up. We have  $(a, b, \dots, f) \in \mathcal{R}(G)$  if and only if  $f' < 1$ ,  $e'' < 1$ ,  $e' < 1$ ,  $\dots$ ,  $a' < 1$ .) Even better is to traverse this graph in another order, using the rule  $M_{G \oplus H} = M_G M_H$  (exercise 336) when subgraphs aren't connected:

$$\begin{aligned} M_{cdabef} &= M_{dabef} \left(1 - c \frac{M_b M_f}{M_{dabef}}\right) = (1 - c')(1 - d') \dots (1 - f'), & c' &= \frac{c}{(1 - a')(1 - d')(1 - e')}, \\ M_{dabef} &= M_{ab} M_{ef} \left(1 - d \frac{M_a M_e}{M_{ab} M_{ef}}\right) = (1 - d')(1 - a')(1 - b')(1 - e')(1 - f'), & & \text{(see below)} \\ M_{ab} &= M_b \left(1 - a \frac{M_e}{M_b}\right) = (1 - a')(1 - b'), & a' &= \frac{a}{(1 - b')}, \\ M_a &= M_e \left(1 - a \frac{M_e}{M_e}\right) = (1 - a''), & a'' &= a, \\ M_b &= M_e \left(1 - b \frac{M_e}{M_e}\right) = (1 - b'), & b' &= b, \end{aligned}$$

where  $d' = dM_a M_e / (M_{ab} M_{ef}) = d(1 - a'')(1 - e'') / ((1 - a')(1 - b')(1 - e')(1 - f'))$ , and  $M_{ef}, M_e, M_f, M_e$  are as before. In this way we can often solve the problem in linear time. [See A. D. Scott and A. D. Sokal, *J. Stat. Phys.* **118** (2005), 1151–1261, §3.4.]

**347.** (a) Suppose  $v_1 \text{---} v_2 \text{---} \cdots \text{---} v_k \text{---} v_1$  is an induced cycle. We can assume that  $v_1 \succ v_2$ . Then, by induction on  $j$ , we must have  $v_1 \succ \cdots \succ v_j$  for  $1 < j \leq k$ ; for if  $v_{j+1} \succ v_j$  we would have  $v_{j+1} \text{---} v_{j-1}$  by (\*). But now  $v_k \text{---} v_1$  implies that  $k = 3$ .

(b) Let the vertices be  $\{1, \dots, m\}$ , with territory sets  $T(a) \subseteq U$  for  $1 \leq a \leq m$ ; and let  $U$  be a tree such that each  $U \upharpoonright T(a)$  is connected. Let  $U_a$  be the least common ancestor of  $T(a)$  in  $U$ . (Thus the nodes of  $T(a)$  appear at the top of the subtree rooted at  $U_a$ .) Since  $U_a \in T(a)$ , we have  $a \text{---} b$  when  $U_a = U_b$ .

Writing  $s \succ_U t$  for the ancestor relation in  $U$ , we now define  $a \succ b$  if  $U_a \succ_U U_b$  or if  $U_a = U_b$  and  $a < b$ . Then (\*) is satisfied: If  $t \in T(a) \cap T(b)$ , we have  $U_a \succ_U t$  and  $U_b \succ_U t$ , hence  $U_a \succeq_U U_b$  or  $U_b \succeq_U U_a$ , hence  $a \succ b$  or  $b \succ a$ . And if  $a \succ b \succ c$  and  $t \in T(a) \cap T(c)$ , we have  $U_a \succeq_U U_b \succeq_U U_c$ ; consequently  $U_b \in T(a) \cap T(b)$ , because  $U_b$  lies on the unique path between  $t$  and  $U_a$  in  $U$  and  $T(a)$  is connected.

(c) Processing the nodes in any order such that  $a$  is eliminated before  $b$  whenever  $U_a$  is a proper ancestor of  $U_b$  will then lead only to subproblems in which the algorithm needs no “double-primed” variables.

For example, using  $(a, b, \dots, g)$  instead of  $(1, 2, \dots, 7)$  in order to match the notation in exercise 346, suppose  $U$  is the tree rooted at  $p$  having the edges  $p \text{---} q$ ,  $p \text{---} r$ ,  $r \text{---} s$ ,  $r \text{---} t$ , and let  $T(a) = \{p, q, r, t\}$ ,  $T(b) = \{p, r, s\}$ ,  $T(c) = \{p, q\}$ ,  $T(d) = \{q\}$ ,  $T(e) = \{r, s\}$ ,  $T(f) = \{s\}$ ,  $T(g) = \{t\}$ . Then  $a \succ b \succ c \succ d$ ,  $c \succ e \succ f$ ,  $e \succ g$ . The algorithm computes  $M_{abcdefg} = (1 - a')M_{bcdefg}$ ,  $M_{bcdefg} = (1 - b')M_{cdefg}$ , etc., where  $a' = aM_f/M_{bcdefg}$ ,  $b' = bM_{dfg}/M_{cdefg} = b(M_dM_fM_g)/(M_{cd}M_{ef}M_g)$ , etc.

In general, the tree ordering guarantees that no “double-primed” variables are needed. Thus the formulas reduce to  $v' = v / \prod_{u \prec v, v \succ u} (1 - u')$  for each vertex  $v$ .

(d) For example, we have  $p_1 = a, \dots, p_7 = g, \theta_1 = a', \dots, \theta_7 = g'$  in (c). The values of the  $\theta$ 's, which depend on the ordering  $\succ$ , are uniquely defined by the given equations; and we have  $M_G(p_1, \dots, p_m) = (1 - \theta_1) \cdots (1 - \theta_m)$  in any case. [W. Pegden, *Random Structures & Algorithms* 41 (2012), 546–556.]

**348.** There is at least one singularity at  $z = \rho e^{i\theta}$  for some  $\theta$ . If  $0 < r < \rho$ , the power series  $f(z) = \sum_{n=0}^{\infty} f^{(n)}(re^{i\theta})(z - re^{i\theta})^n/n!$  has radius of convergence  $\rho - r$ . If  $z = \rho$  isn't a singularity, the radius of convergence for  $\theta = 0$  would exceed  $\rho - r$ . But  $|f^{(n)}(re^{i\theta})| = |\sum_{m=0}^{\infty} m^2 a_n (re^{i\theta})^{m-n}| \leq f^{(n)}(r)$ . [*Mathematische Annalen* 44 (1894), 41–42.]

**349.** Typical generating functions are  $g_{0000001} = 1$ ;  $g_{01101110} = z(g_{01001110} + g_{01011110} + g_{01101110} + g_{01111110})/4$  (in Case 1) or  $g_{01101110} = z(g_{00001110} + g_{00101110} + g_{01001110} + g_{01101110})/4$  (in Case 2). These systems of 128 linear equations have solutions whose denominators involve one or more of the polynomials  $4 - z$ ,  $2 - z$ ,  $16 - 12z + z^2$ ,  $4 - 3z$ ,  $64 - 80z + 24z^2 - z^3$ ,  $8 - 8z + z^2$  in Case 1 (see exercise 320); the denominators in Case 2 are powers of  $4 - z$ .

Setting  $g(z) = \sum_x g_x(z)/128$  leads to  $g(z) = 1/((2 - z)(8 - 8z + z^2))$  in Case 1, with mean 7 and variance 42;  $g(z) = (1088 - 400z + 42z^2 - z^3)/(4 - z)^6$  in Case 2, with mean  $1139/729 \approx 1.56$  and variance  $1139726/729^2 \approx 2.14$ .

[The upper bound  $E_1 + \cdots + E_6$  is achieved by the distribution of Case 1, because it matches the extreme distribution (148) of the path graph  $P_6$ . Incidentally, if Case 1 is generalized from  $n = 7$  to arbitrary  $n$ , the mean is  $n(n - 1)/6$  and the variance is  $(n + 3)(n + 2)n(n - 1)/90$ .]

**350.** (a) The generating function for  $N$  is  $\prod_{k=1}^n (1 - \xi_k)/(1 - \xi_k z)$ ; so the mean and variance, in general, are  $\sum_{k=1}^n \xi_k/(1 - \xi_k)$  and  $\sum_{k=1}^n \xi_k/(1 - \xi_k)^2$ . In particular, the means are (i)  $n$ ; (ii)  $n/(2n - 1)$ ; (iii)  $n/(2^n - 1)$ ; (iv)  $H_{2n} - H_n + \frac{1}{2n} = \ln 2 + O(1/n)$ ; (v)  $\frac{1}{2}(\frac{1}{n+1} + \frac{1}{2n} - \frac{1}{2n+1}) = \frac{1}{2n} + O(1/n^2)$ . The variance in case (i) is 2; otherwise it's asymptotically the same as the mean, times  $1 + O(1/n)$ .

(b) In this case the mean and variance are  $\xi/(1 - \xi)$  and  $\xi/(1 - \xi)^2$ , where  $\xi = \Pr(A_m) = 1 - (1 - \xi_1) \dots (1 - \xi_n)$ . This value  $\xi$  is (i)  $1 - 2^{-n}$ ; (ii)  $1 - (1 - \frac{1}{2n})^n = 1 - e^{-1/2} + O(1/n)$ ; (iii)  $1 - (1 - 2^{-n})^n = n/2^n + O(n^2/4^n)$ ; (iv)  $1/2$ ; (v)  $1/(2n + 2)$ . Hence the respective means are (i)  $2^n - 1$ ; (ii)  $e^{1/2} - 1 + O(1/n)$ ; (iii)  $n/2^n + O(n^2/4^n)$ ; (iv)  $1$ ; (v)  $1/(2n + 1)$ . And the variances are (i)  $4^n - 2^n$ ; (ii)  $e - e^{1/2} + O(1/n)$ ; (iii)  $n/2^n + O(n^2/4^n)$ ; (iv)  $2$ ; (v)  $1/(2n + 1) + 1/(2n + 1)^2$ .

complete bipartite graph  
Moser  
Tardos  
Alon  
Spencer  
traces  
Fibonacci numbers  
consecutive ones  
chain rule

(c) Since  $G$  is  $K_{n,1}$ , exercises 336 and 343 imply that  $(\xi_1, \dots, \xi_n, \xi) \in \mathcal{R}(G)$  if and only if  $\xi < \frac{1}{2}$ . This condition holds in cases (ii), (iii), and (v).

**351.** (Solution by Moser and Tardos.) We require  $i - j$  if there's a setting of the variables such that  $A_i$  is false and  $A_j$  is true, provided that some change to the variables of  $\Xi_j$  might make  $A_i$  true. And vice versa with  $i \leftrightarrow j$ .

(The Local Lemma can be proved also for *directed* lopsided dependency graphs; see Noga Alon and Joel H. Spencer, *The Probabilistic Method* (2008), §5.1. But the theory of traces, which we use to analyze Algorithm M, is based on undirected graphs, and no algorithmic extension to the directed case is presently known.)

**352.** Answer 344(e), with  $M_G = \beta(i \cup J)$ ,  $M_{G \setminus i} = \beta(J)$ , proves that  $M_{G \setminus i}/M_G \geq 1 - \theta_i$ .

**353.** (a) There are  $n + 1$  sorted strings in Case 1, namely  $0^k 1^{n-k}$  for  $0 \leq k \leq n$ . There are  $F_{n+2}$  solutions in Case 2 (see, for example, exercise 7.2.1.1-91).

(b) At least  $2^n M_G(1/4)$ , where  $G$  is the path  $P_{n-1}$ . By exercise 320 we have  $M_G(1/4) = f_{n-1}(1/4) = (n + 1)/2^n$ ; so Case 1 matches the lower bound.

(c) There are *no* lopsided dependencies. Hence the relevant  $G$  is the empty graph on  $m = n - 1$  vertices;  $M_G(1/4) = (3/4)^{n-1}$  by exercise 336; and indeed,  $F_{n+2} \geq 3^{n-1} 2^{2-n}$ .

**354.** Differentiate (151) and set  $z \leftarrow 1$ .

**355.** If  $A = A_j$  is an isolated vertex of  $G$ , then  $1 - p_j z$  is a factor of the polynomial  $M_G^*(z)$  in (149), hence  $1 + \delta \leq 1/p_j$ ; and  $E_j = p_j/(1 - p_j) \leq 1/\delta$ . Otherwise  $M_G(p_1, \dots, p_{j-1}, p_j(1 + \delta), p_{j+1}, \dots, p_m) = M_G^*(1) - \delta p_j M_{G \setminus A}^*(1) > M_G^*(1 + \delta) = 0$ ; so  $E_j = p_j M_{G \setminus A}^*(1)/M_G^*(1) > 1/\delta$ .

**356.** (a) We prove the hint by induction on  $|S|$ . It's obvious when  $S = \emptyset$ ; otherwise let  $X = S \cap \bigcup_{i \in U_j} U_j$  and  $Y = S \setminus X$ . We have

$$\Pr(A_i | \overline{A}_S) = \frac{\Pr(A_i \cap \overline{A}_X \cap \overline{A}_Y)}{\Pr(\overline{A}_X \cap \overline{A}_Y)} \leq \frac{\Pr(A_i \cap \overline{A}_Y)}{\Pr(\overline{A}_X \cap \overline{A}_Y)} \leq \frac{\Pr(A_i) \Pr(\overline{A}_Y)}{\Pr(\overline{A}_X \cap \overline{A}_Y)} = \frac{\Pr(A_i)}{\Pr(\overline{A}_X | \overline{A}_Y)}$$

by (133). Suppose  $i$  belongs to the cliques  $U_{j_0}, \dots, U_{j_r}$  where  $j = j_0$ . Let  $X_0 = \emptyset$  and  $X_k = (S \cap U_{j_k}) \setminus X_{k-1}$ ,  $Y_k = Y \cup X_1 \cup \dots \cup X_{k-1}$  for  $1 \leq k \leq r$ . We have  $\Pr(A_i | \overline{A}_{Y_k}) \leq \theta_{i j_k}$  for all  $l \in X_k$ , since  $|Y_k| < |S|$  when  $X_k \neq \emptyset$ ; hence  $\Pr(\overline{A}_{X_k} | \overline{A}_{Y_k}) \geq (1 + \theta_{i j_k} - \Sigma_{j_k})$ . Thus  $\Pr(\overline{A}_X | \overline{A}_Y) = \Pr(\overline{A}_{X_1} | \overline{A}_{Y_1}) \Pr(\overline{A}_{X_2} | \overline{A}_{Y_2}) \dots \Pr(\overline{A}_{X_r} | \overline{A}_{Y_r}) \geq \prod_{k \neq j, i \in U_k} (1 + \theta_{ik} - \Sigma_k)$ , by the chain rule (exercise MPR-14); the hint follows.

Finally let  $W_k = U_1 \cup \dots \cup U_k$  for  $1 \leq k \leq t$ . The hint implies that

$$\begin{aligned} \Pr(\overline{A}_1 \cap \dots \cap \overline{A}_m) &= \Pr(\overline{A}_{W_1}) \Pr(\overline{A}_{W_2} | \overline{A}_{W_1}) \dots \Pr(\overline{A}_{W_t} | \overline{A}_{W_{t-1}}) \\ &\geq (1 - \Sigma_1)(1 - \Sigma_2) \dots (1 - \Sigma_t) > 0. \end{aligned}$$

(b) The extreme events  $B_1, \dots, B_m$  of Theorem S satisfy the hint of (a). Thus  $\Pr(B_i | \bigcap_{k \notin U_j} \overline{B}_k) \leq \theta_{ij}$  for all  $i \in U_j$ ; hence  $q_i = \Pr(B_i | \bigcap_{k \neq i} \overline{B}_k) \leq \theta_{ij}/(1 + \theta_{ij} - \Sigma_j)$ . Furthermore  $E_i = q_i/(1 - q_i)$  in (152), because  $q_i = p_i M_{G \setminus i}^*/M_{G \setminus i}$ .

(c) Let  $U_1, \dots, U_t$  be the edges of  $G$ , with  $\theta_{ik} = \theta_i$  when  $U_k = \{i, j\}$ . Then  $\Sigma_k = \theta_i + \theta_j < 1$ , and the sufficient condition in (a) is that  $\Pr(A_i) \leq \theta_i \prod_{j \neq k, i \rightarrow j} (1 - \theta_j)$  whenever  $i \rightarrow k$ . (But notice that Theorem M does not hold for such larger  $p_i$ .)

[K. Kolipaka, M. Szegedy, and Y. Xu, *LNCS 7408* (2012), 603–614.]

Kolipaka  
Szegedy  
Xu  
covering  
invariant property  
cufflink pattern  
 $L_7$   
lower semimodular  
modular

**357.** If  $r > 0$ , we have  $x = r/(1-p)$ ,  $y = r/(1-q)$ . But  $r = 0$  is possible only on the axes of Fig. 51: Either  $(p, q) = (0, 1)$ ,  $x = 0$ ,  $0 < y \leq 1$ , or  $(p, q) = (1, 0)$ ,  $0 < x \leq 1$ ,  $y = 1$ .

**358.** Suppose  $x \geq y$  (hence  $p \geq q$  and  $x > 0$ ). Then  $p \leq r$  if and only if  $1 - y \leq y$ .

**359.** Instead of computing  $\pi_l$  by formula (154), represent it as two numbers  $(\pi_l^+, \pi_l')$ , where  $\pi_l^+$  is the product of the nonzero factors and  $\pi_l'$  is the number of zero factors. Then the quantity  $\pi_l$  needed in (156) is  $\pi_l^+ [\pi_l' = 0]$ ; and the quantity  $\pi_l / (1 - \eta_{C \rightarrow l})$  is  $\pi_l^+ [\pi_l' = 1]$  if  $\eta_{C \rightarrow l} = 1$ , otherwise it's  $\pi_l^+ [\pi_l' = 0] / (1 - \eta_{C \rightarrow l})$ . A similar method can be used to separate out the zero factors of  $\prod_{l \in C} \gamma_{l \rightarrow C}$  in (157).

**360.** We may assume that  $\eta_3 = 0$ . Since  $\pi_l = 1$  implies that  $\eta_{C \rightarrow l} = \gamma_{\bar{l} \rightarrow C} = 0$ , we have  $\eta_{C \rightarrow 1} = \eta_{C \rightarrow 2} = \eta_{C \rightarrow 3} = \eta_{C \rightarrow 4} = \gamma_{\bar{1} \rightarrow C} = \gamma_{2 \rightarrow C} = \gamma_{3 \rightarrow C} = \gamma_{4 \rightarrow C} = 0$  for all  $C$ . Consequently, as in (159), all but three of the values  $\eta_{C \rightarrow l}$  are zero; let  $x, y, z$  denote the others. Also let  $\eta_{\bar{1}} = a$ ,  $\eta_2 = b$ ,  $\eta_4 = c$ ,  $\eta_{\bar{3}} = d$ . Then  $\pi_{\bar{1}} = (1-a)(1-x)$ ,  $\pi_2 = (1-b)(1-y)$ ,  $\pi_4 = (1-c)(1-z)$ , and  $\pi_{\bar{3}} = 1 - d$ . A fixed point is obtained if  $x = d(b + cd(1-b) + ad^2(1-b)(1-c)) / (1 - d^3(1-a)(1-b)(1-c))$ , etc. If  $d$  is 0 or 1 then  $x = y = z = d$ . [Are there any other fixed points, say with  $\pi_1 \neq 1$ ?]

**361.** The  $\pi$ 's and  $\gamma$ 's will also be either 0 or 1, and we exclude the case  $\pi_l = \pi_{\bar{l}} = 0$ ; thus each variable  $v$  is either 1, 0, or \*, depending on whether  $(\pi_v, \pi_{\bar{v}})$  is (0, 1), (1, 0), or (1, 1).

Any assignment of 1, 0, or \* to the variables is permissible, provided that every clause has at least one literal that's true or two that are \*. (Such partial assignments are called “covering,” and they're usually possible even with unsatisfiable clauses; see exercise 364.) All survey messages  $\eta'_{C \rightarrow l} = \eta_{C \rightarrow l}$  are zero except when clause  $C$  has  $l$  as its only non-false literal. The reinforcement message  $\eta_l$  can be either 0 or 1, except that it must be 1 if  $l$  is true ( $\pi_l = 0$ ) and all messages  $\eta_{C \rightarrow l}$  are 0.

If we also want  $\eta'_l = \eta_l$ , we take  $\kappa = 1$  in (158), and  $\eta_l = 1 - \pi_l$ .

**362.** Create a linked list  $L$ , containing all literals that are to be forced true, including all literals that are in 1-clauses of the original problem. Do the following steps while  $L$  is nonempty: Remove a literal  $l$  from  $L$ ; remove all clauses that contain  $l$ ; and remove  $\bar{l}$  from all the clauses that remain. If any of those clauses has thereby been reduced to a single literal, ( $l'$ ), check to see if  $l'$  or  $\bar{l}'$  is already present in  $L$ . If  $\bar{l}'$  is present, a contradiction has arisen; we must either terminate unsuccessfully or restart step S8 with increased  $\psi$ . But if  $\bar{l}'$  and  $l'$  are both absent, put  $l'$  into  $L$ .

**363.** (a) True; indeed, this is an important invariant property of Algorithm C.

(b)  $W(001) = 1$ ,  $W(***) = p_1 p_2 p_3$ , otherwise  $W(x) = 0$ .

(c) Statements (i) and (iii) are true, but not (ii); consider  $x = 10*$ ,  $x' = 00*$ , and the clause 123.

(d) All eight subsets of  $\{\bar{1}, \bar{2}, \bar{3}\}$  are stable except  $\{\bar{2}, \bar{3}\}$ , because  $x_1$  is constrained in 100. The other seven are partially ordered as shown. (This diagram illustrates  $L_7$ , the smallest lattice that is lower semimodular but not modular.)

|     |                |           |               |               |               |               |               |               |               |               |
|-----|----------------|-----------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| (e) | $x_2 x_3 = 00$ | 01        | 0*            | 10            | 11            | 1*            | *0            | *1            | **            |               |
|     | $x_1 = 0$      | 0         | $q_1 q_2$     | 0             | $q_1 q_3$     | $q_1 q_2 q_3$ | $q_1 q_2 p_3$ | 0             | $q_1 p_2 q_3$ | $q_1 p_2 p_3$ |
|     | $x_1 = 1$      | $q_2 q_3$ | $q_1 q_2 q_3$ | $q_1 q_2 p_3$ | $q_1 q_2 q_3$ | $q_1 q_2 q_3$ | $q_1 q_2 p_3$ | $q_1 p_2 q_3$ | $q_1 p_2 q_3$ | $q_1 p_2 p_3$ |
|     | $x_1 = *$      | 0         | $p_1 q_2 q_3$ | $p_1 q_2 p_3$ | $p_1 q_2 q_3$ | $p_1 q_2 q_3$ | $p_1 q_2 p_3$ | $p_1 p_2 q_3$ | $p_1 p_2 q_3$ | $p_1 p_2 p_3$ |

(f) One solution is  $\{\bar{1}\bar{2}3\bar{4}\bar{5}, \bar{1}4, \bar{2}5, \bar{3}4\bar{5}, \bar{3}45\}$ . (For these clauses the partial assignment  $\{3\}$  is stable, but it is “unreachable” below  $\{1, 2, 3, 4, 5\}$ .)

(g) If  $L = L' \setminus l$  and  $L' \in \mathcal{L}$  but  $L \notin \mathcal{L}$ , introduce the clause  $(x_l \vee \bigvee_{k \in L'} \bar{x}_k)$ .

(h) True, because  $L' = L \setminus l'$  and  $L'' = L \setminus l''$ , where  $|l'|$  and  $|l''|$  are unconstrained with respect to  $L$ . A variable that's unconstrained with respect to  $L$  is also unconstrained with respect to any subset of  $L$ .

(i) Suppose  $L' = L'^{(0)} \prec \dots \prec L'^{(s)} = \{1, \dots, n\}$  and  $L'' = L''^{(0)} \prec \dots \prec L''^{(t)} = \{1, \dots, n\}$ . Then  $L'^{(s-i)} \cap L''^{(t-j)}$  is stable for  $0 \leq i \leq s$  and  $0 \leq j \leq t$ , by induction on  $i + j$  using (h).

(j) It suffices to consider the case  $L = \{1, \dots, n\}$ . Suppose the unconstrained variables are  $x_1, x_2, x_3$ . Then, by induction, the sum is  $q_1 q_2 q_3 + p_1 + p_2 + p_3 - (p_1 p_2 + p_1 p_3 + p_2 p_3) + p_1 p_2 p_3 = 1$ , using “inclusion and exclusion” to compensate for terms that are counted more than once. A similar argument works with any number of unconstrained variables.

*Notes:* See F. Ardila and E. Maneva, *Discrete Mathematics* **309** (2009), 3083–3091. The sum in (j) is  $\leq 1$  when each  $p_k + q_k \leq 1$  for  $1 \leq k \leq n$ , because it is monotone. Because of (i), the stable sets below  $L$  form a lower semimodular lattice, with

$$L' \wedge L'' = L' \cap L'' \quad \text{and} \quad L' \vee L'' = \bigcap \{L''' \mid L''' \supseteq L' \cup L'' \text{ and } L''' \sqsubseteq L\}.$$

E. Maneva and A. Sinclair noted in *Theoretical Comp. Sci.* **407** (2008), 359–369 that a random satisfiability problem is satisfiable with probability  $\leq E \sum W(X)$ , the expected total weight of partial assignments having the given distribution, because of identity (j); this led them to sharper bounds than had previously been known.

**364.** (a) True if and only if all clauses have length 2 or more.

(b) 001 and \*\*\* are covering; these are the partial assignments of nonzero weight, when  $q_1 = \dots = q_n = 0$  in the previous exercise. Only 001 is a core.

(c) \*\*\* is the only covering and the only core;  $W(0101) = W(0111) = q_3$ .

(d) In fact, every stable partial assignment  $L'$  has a unique covering assignment  $L$  with  $L \sqsubseteq L'$ , namely  $L = \bigcap \{L'' \mid L'' \sqsubseteq L', \text{ obtained by successively removing unconstrained literals (in any order)}\}$ .

(e) If  $L'$  and  $L''$  are adjacent we have  $L' \cap L'' \sqsubseteq L'$  and  $L' \cap L'' \sqsubseteq L''$ .

(f) Not necessarily. For example, the clauses  $\{\bar{1}\bar{2}34, \bar{1}\bar{2}\bar{3}4, \bar{1}\bar{2}3\bar{4}, \bar{1}\bar{2}\bar{3}\bar{4}, \bar{1}\bar{2}3\bar{4}, \bar{1}\bar{2}\bar{3}\bar{4}\}$  define  $\bar{S}_2(x_1, x_2, x_3, x_4)$ ; there are two clusters but only an empty core.

[A. Braunstein and R. Zecchina introduced the notion of covering assignments in *J. Statistical Mechanics* (June 2004), P06007:1–18.]

**365.** If  $L$  is any of the six solutions in (8), and if  $q$  is odd, then  $qL - d$  is a covering assignment for  $0 \leq d < q$  and  $8q - d \leq n < 9q - d$ . (For example, if  $L = \{\bar{1}, \bar{2}, 3, 4, \bar{5}, \bar{6}, 7, 8\}$  the partial assignment  $3L - 1 = \{\bar{2}, \bar{5}, 8, 11, \bar{14}, \bar{17}, 20, 23\}$  works for  $n \in [23..25]$ .) Thus all  $n > 63$  are “covered.” [Do all nonempty coverings of *waerden*(3, 3;  $n$ ) have this form?]

**366.** Eliminating variable 1 ( $x_1$ ) by resolution yields the erp rule  $\bar{x}_1 \leftarrow (x_2 \vee \bar{x}_3) \wedge (x_3 \vee x_4)$ , and new clauses  $\{2\bar{3}4, 2\bar{3}\bar{4}, 234, \bar{2}34\}$ . Then eliminating 2 ( $x_2$ ) yields  $x_2 \leftarrow (x_3 \vee x_4) \wedge (\bar{x}_3 \vee x_4)$  and new clauses  $\{34, \bar{3}4\}$ . Now 4 ( $x_4$ ) is pure; so  $x_4 \leftarrow 1$ , and  $F' = \emptyset$  is satisfiable. (Going backwards in the erp rules will then make  $x_4 \leftarrow 1, x_2 \leftarrow 1, x_1 \leftarrow 0$ , regardless of  $x_3$ .)

**367.** (We can choose whichever of the two assignments is most convenient, for example by picking the shortest, since either one is a valid erp rule.) Any solution will either satisfy all the clauses on the right side of  $\bar{x}$  or all the clauses on the right side of  $x$ , or both. For if a solution falsifies both  $C_i \setminus x$  and  $C_j' \setminus \bar{x}$ , it falsifies  $C_i \diamond C_j'$ .

inclusion and exclusion  
Ardila  
semimodular lattice  
Maneva  
Sinclair  
symmetric Boolean functions  $S_k$   
Braunstein  
Zecchina  
Eliminating  
resolution  
pure

In either case the value of  $x$  will satisfy all of the clauses  $C_1, \dots, C_a, C'_1, \dots, C'_b$ .

**368.** If  $(l)$  is a clause, subsumption removes all other clauses that contain  $l$ . Then resolution (with  $p = 1$ ) will remove  $\bar{l}$  from all  $q$  of its clauses, and  $(l)$  itself.

**369.** Let  $C_i = (l \vee \alpha_i)$  and  $C'_j = (\bar{l} \vee \beta_j)$ . Each omitted clause  $C_i \diamond C'_j = (\alpha_i \vee \beta_j)$ , where  $1 < i \leq p$  and  $r < j \leq q$ , is redundant, because it is a consequence of the non-omitted clauses  $(\alpha_i \vee \bar{l}_1), \dots, (\alpha_i \vee \bar{l}_r), (l_1 \vee \dots \vee l_r \vee \beta_j)$  via hyperresolution. [This technique is called “substitution,” because we essentially replace  $|l|$  by its definition.]

**370.**  $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee b \vee c) = (a \vee \bar{c}) \wedge (b \vee c)$ . (See the discussion following 7.1.1-(27). In general, advanced preprocessors use the theory of DNF minimization, in its dual form, to find irredundant minimum forms for CNF. Such techniques are not implemented, however, in the examples of preprocessing considered in this section.)

**371.** One scenario starts by eliminating variable 1, replacing eight clauses by eight new ones: 234 $\bar{7}$ , 2 $\bar{3}$ 47, 235 $\bar{9}$ , 2 $\bar{3}$ 59, 345 $\bar{7}$ , 3 $\bar{4}$ 57, 457 $\bar{9}$ , 4 $\bar{5}$ 79. Then 8 is eliminated, replacing another eight by eight: 245 $\bar{6}$ , 2 $\bar{4}$ 56, 256 $\bar{7}$ , 2 $\bar{5}$ 67, 257 $\bar{9}$ , 2 $\bar{5}$ 79, 467 $\bar{9}$ , 4 $\bar{6}$ 79. Then come self-subsumptions: 234 $\bar{7}$   $\mapsto$  23 $\bar{7}$  (via 234), 3 $\bar{4}$ 5 $\bar{7}$   $\mapsto$  35 $\bar{7}$  (345), 357  $\mapsto$  35 (35 $\bar{7}$ ); and 35 subsumes 345, 35 $\bar{7}$ . Further self-subsumptions yield 235 $\bar{9}$   $\mapsto$  239, 2 $\bar{3}$ 59  $\mapsto$  2 $\bar{3}$ 9, 2579  $\mapsto$  279, 2 $\bar{4}$ 56  $\mapsto$  246, 246  $\mapsto$  46; and 46 subsumes 456, 467 $\bar{9}$ , 246. Similarly, 2567  $\mapsto$  267, 4579  $\mapsto$  459, 2 $\bar{3}$ 47  $\mapsto$  2 $\bar{3}$ 7, 3 $\bar{4}$ 57  $\mapsto$  357, 35 $\bar{7}$   $\mapsto$  35; and 35 subsumes 345, 357. Then 2456  $\mapsto$  246, 246  $\mapsto$  46; and 46 subsumes 456, 246, 4679. Also 2567  $\mapsto$  267, 4579  $\mapsto$  459, 2579  $\mapsto$  279.

Round 2 of variable elimination first gets rid of 4, replacing six clauses by just four using exercise 369: 23 $\bar{6}$ , 2 $\bar{3}$ 6, 569, 5 $\bar{6}$ 9. Then variable 3 goes away; ten clauses become eight, again via exercise 369: 25 $\bar{6}$ , 256, 25 $\bar{7}$ , 257, 25 $\bar{9}$ , 259, 56 $\bar{9}$ , 569. And the ten clauses that now contain 2 or  $\bar{2}$  resolve into just four: 567 $\bar{9}$ , 5 $\bar{6}$ 79, 567 $\bar{9}$ , 5 $\bar{6}$ 79.

After eliminating 7 and 9, only four clauses remain, namely 56, 5 $\bar{6}$ , 56, 5 $\bar{6}$ ; and they quickly produce a contradiction.

**372.** (This problem is surprisingly difficult.) Are the clauses  $\{\bar{1}\bar{5}, \bar{1}\bar{6}, \bar{2}\bar{5}, \bar{2}\bar{6}, \bar{3}\bar{7}, \bar{3}\bar{8}, \bar{4}\bar{7}, \bar{4}\bar{8}, 123, 124, 134, 234, 567, 568, 578, 678\}$  as “small” as possible?

**373.** Using the notation of (102), elimination of  $x_{1m}, x_{2m}, \dots, x_{mm}$  produces new clauses  $M'_{imk}$  for  $1 \leq i, k < m$  as well as  $M_{m(m-1)}$ . Then elimination of  $x_{m(m-1)}$  gives  $(M_{i(m-1)} \vee M_{m(m-2)})$  for  $1 \leq i < m$ . This clause self-subsumes to  $M_{i(m-1)}$ , using  $M'_{im1}, \dots, M'_{im(m-2)}$ . And  $M_{i(m-1)}$  subsumes each  $M'_{imk}$ , so we've reduced  $m$  to  $m-1$ .

**374.** As in (57), variables are numbered 1 to  $n$ , and literals from 2 to  $2n+1$ . But we will now number the clauses from  $2n+2$  to  $m+2n+1$ . The literals of clauses will be stored in cells, somewhat as in Algorithm A, but with additional links as in the exact cover algorithms of Section 7.2.2.1: Each cell  $p$  contains not only a literal  $L(p)$ , a clause number  $C(p)$ , and forward/backward pointers  $F(p)$  and  $B(p)$  to other cells with the same literal, but also left/right pointers  $S(p)$  and  $D(p)$  to other cells in the same clause. (Think “sinister” and “dexter.”) Cells 0 and 1 are reserved for special use; cell  $l$ , for  $2 \leq l < 2n+2$ , serves as the head of the doubly linked list of cells that contain the literal  $l$ ; cell  $c$ , for  $2n+2 \leq c < m+2n+2$ , serves as the head of the doubly linked list of cells that contain the elements of clause  $c$ ; and cell  $p$ , for  $m+2n+2 \leq p < M$ , either is available for future use or holds literal and clause data for a currently active clause.

Free cells are accessed via a global pointer AVAIL. To get a new  $p \leftarrow$  AVAIL when AVAIL  $\neq$  0, we set  $p \leftarrow$  AVAIL, AVAIL  $\leftarrow$  S(AVAIL); but if AVAIL = 0, we set  $p \leftarrow$  M and  $M \leftarrow$  M + 1 (assuming that M never gets too large). To free one or more cells from  $p'$  to  $p''$  that are linked together via left links, we set S( $p'$ )  $\leftarrow$  AVAIL and AVAIL  $\leftarrow$   $p''$ .

hyperresolution  
substitution  
DNF  
Two-level circuit minimization  
irredundant  
self-subsumptions  
self-subsumes  
exact cover  
doubly linked  
AVAIL

The number of active clauses containing literal  $l$ ,  $\text{TALLY}(l)$ , can therefore be computed as follows: Set  $t \leftarrow 0$ ,  $p \leftarrow \text{F}(l)$ ; while not  $\text{lit}(p)$ , set  $t \leftarrow t + 1$  and  $p \leftarrow \text{F}(p)$ ; set  $\text{TALLY}(l) \leftarrow t$ ; here ' $\text{lit}(p)$ ' stands for ' $p < 2n + 2$ '. The number of literals in clause  $c$ ,  $\text{SIZE}(c)$ , can be computed by a similar loop, using ' $\text{cls}(p)$ ' to stand for ' $p < m + 2n + 2$ ': Set  $t \leftarrow 0$ ,  $p \leftarrow \text{S}(c)$ ; while not  $\text{cls}(p)$ , set  $t \leftarrow t + 1$  and  $p \leftarrow \text{S}(p)$ ; set  $\text{SIZE}(c) \leftarrow t$ . After initialization, the  $\text{TALLY}$  and  $\text{SIZE}$  statistics can be updated dynamically as local changes are made. ( $\text{TALLY}(l)$  and  $\text{SIZE}(c)$  can be maintained in  $\text{L}(l)$  and  $\text{C}(c)$ .)

signature  
bitwise OR  
Bloom  
list merge  
tautology  
Biere  
False hits  
STAMP( $l$ )  
time stamp

To facilitate resolution, the literals of each clause are required to increase from left to right; in other words, we must have  $\text{L}(p) < \text{L}(q)$  whenever  $p = \text{S}(q)$  and  $q = \text{D}(p)$ , unless  $\text{cls}(p)$  or  $\text{cls}(q)$ . But the clauses within literal lists need not appear in any particular order. We might even have  $\text{C}(\text{F}(p)) > \text{C}(q)$  but  $\text{C}(\text{F}(p')) < \text{C}(q')$ , when  $\text{C}(p) = \text{C}(p')$  and  $\text{C}(q) = \text{C}(q')$ .

To facilitate subsumption, each literal  $l$  is assigned a 64-bit *signature*  $\text{SIG}(l) = (1 \ll U_1) \mid (1 \ll U_2)$ , where  $U_1$  and  $U_2$  are independently random 6-bit numbers. Then each clause  $c$  is assigned a signature that is the bitwise OR of the signatures of its literals: Set  $t \leftarrow 0$ ,  $p \leftarrow \text{S}(c)$ ; while not  $\text{cls}(p)$ , set  $t \leftarrow t \mid \text{SIG}(\text{L}(p))$  and  $p \leftarrow \text{S}(p)$ ; set  $\text{SIG}(c) \leftarrow t$ . (See the discussion of Bloom's superimposed coding in Section 6.5.)

(a) To resolve  $c$  with  $c'$ , where  $c$  contains  $l$  and  $c'$  contains  $\bar{l}$ , we essentially want to do a list merge. Set  $p \leftarrow 1$ ,  $q \leftarrow \text{S}(c)$ ,  $u \leftarrow \text{L}(q)$ ,  $q' \leftarrow \text{S}(c')$ ,  $u' \leftarrow \text{L}(q')$ , and do the following while  $u + u' > 0$ : If  $u = u'$ , copy( $u$ ) and bump( $q, q'$ ); if  $u = \bar{u}' = l$ , bump( $q, q'$ ); if  $u = \bar{u}' \neq l$ , terminate unsuccessfully; otherwise if  $u > u'$ , copy( $u$ ) and bump( $q$ ); otherwise copy( $u'$ ) and bump( $q'$ ). Here 'copy( $u$ )' means 'set  $p' \leftarrow p$ ,  $p \leftarrow \text{AVAIL}$ ,  $\text{S}(p') \leftarrow p$ ,  $\text{L}(p) \leftarrow u$ '; 'bump( $q$ )' means 'set  $q \leftarrow \text{S}(q)$ ; if  $\text{cls}(q)$  set  $u \leftarrow 0$ , otherwise set  $u \leftarrow \text{L}(q)$ '; 'bump( $q'$ )' is similar, but it uses  $q'$  and  $u'$ ; and 'bump( $q, q'$ )' means 'bump( $q$ ) and bump( $q'$ )'. Unsuccessful termination occurs when clauses  $c$  and  $c'$  resolve to a tautology; we set  $p \leftarrow 0$ , after first returning cells  $p$  through  $\text{S}(1)$  to free storage if  $p \neq 1$ . Successful termination with  $u = u' = 0$  means that the resolved clause consists of the literals in cells from  $p$  through  $\text{S}(1)$ , linked only via  $\text{S}$  pointers.

(b) Find a literal  $l$  with minimum  $\text{TALLY}(l)$ . Set  $p \leftarrow \text{F}(l)$ , and do the following while not  $\text{lit}(p)$ : Set  $c' \leftarrow \text{C}(p)$ ; if  $c' \neq c$  and  $\sim \text{SIG}(c') \ \& \ \text{SIG}(c) = 0$  and  $\text{SIZE}(c') \geq \text{SIZE}(c)$ , do a detailed subsumption test; then set  $p \leftarrow \text{F}(p)$ . The detailed test begins with  $q \leftarrow \text{S}(c)$ ,  $u \leftarrow \text{L}(q)$ ,  $q' \leftarrow \text{S}(c')$ ,  $u' \leftarrow \text{L}(q')$ , and does the following steps while  $u' \geq u > 0$ : bump( $q'$ ) while  $u' > u$ ; then bump( $q, q'$ ) if  $u' = u$ . When the loop terminates,  $c$  subsumes  $c'$  if and only if  $u \leq u'$ .

(c) Use (b) but with  $(\text{SIG}(c) \ \& \ \sim \text{SIG}(l))$  in place of  $\text{SIG}(c)$ . Also modify the detailed test, by inserting 'if  $u = l$  then  $u \leftarrow \bar{l}$ ' just after each occurrence of ' $u \leftarrow \text{L}(q)$ '.

[The algorithm in (b) was introduced by A. Biere, *LNCS 3542* (2005), 59–70, §4.2. "False hits," in which the detailed test is performed but no actual (self-)subsumption is detected, tend to occur less than 1% of the time in practice.]

**375.** Let each literal  $l$  have another field  $\text{STAMP}(l)$ , initially zero; and let  $s$  be a global "time stamp" that is initially zero. To make the test, set  $s \leftarrow s + 1$  and  $\sigma \leftarrow 0$ ; then set  $\text{STAMP}(u) \leftarrow s$  and  $\sigma \leftarrow \sigma \mid \text{SIG}(u)$  for all  $u$  such that  $(\bar{l}u)$  is a clause. If  $\sigma \neq 0$ , set  $\sigma \leftarrow \sigma \mid \text{SIG}(l)$  and run through all clauses  $c$  that contain  $l$ , doing the following: If  $\text{SIG}(c) \ \& \ \sim \sigma = 0$ , and if each of  $c$ 's literals  $u \neq l$  has  $\text{STAMP}(u) = s$ , exit with  $C_1 = c$  and  $r = \text{SIZE}(c) - 1$ . If  $C_1$  has thereby been found, set  $s \leftarrow s + 1$  and  $\text{STAMP}(\bar{u}) \leftarrow s$  for all  $u \neq l$  in  $c$ . Then a clause  $(\bar{l} \vee \beta_j)$  implicitly has  $j \leq r$  in the notation of exercise 369 if and only if  $\beta_j$  is a single literal  $u$  with  $\text{STAMP}(u) = s$ .

Given a variable  $x$ , test the condition first for  $l = x$ ; if that fails, try  $l = \bar{x}$ .

**376.** Highest priority is given to the common operations of unit conditioning and pure literal elimination, which are “low-hanging fruit.” Give each variable  $x$  two new fields,  $\text{STATE}(x)$  and  $\text{LINK}(x)$ . A “to-do stack,” containing all such easy pickings, begins at  $\text{TODO}$  and follows  $\text{LINK}$ s until reaching  $\Lambda$ . Variable  $x$  is on this stack only if  $\text{STATE}(x)$  is nonzero; the nonzero states are called  $\text{FF}$  (forced false),  $\text{FT}$  (forced true),  $\text{EQ}$  (eliminated quietly), and  $\text{ER}$  (eliminated by resolution).

Whenever a unit clause ( $l$ ) is detected, with  $\text{STATE}(|l|) = 0$ , we set  $\text{STATE}(|l|) \leftarrow (l \ \& \ 1? \text{FF}:\text{FT})$ ,  $\text{LINK}(|l|) \leftarrow \text{TODO}$ , and  $\text{TODO} \leftarrow |l|$ . But if  $\text{STATE}(|l|) = (l \ \& \ 1? \text{FT}:\text{FF})$ , we terminate, because the clauses are unsatisfiable.

Whenever a literal with  $\text{TALLY}(l) = 0$  is detected, we do the same thing if  $\text{STATE}(|l|) = 0$ . But if  $\text{STATE}(|l|) = (l \ \& \ 1? \text{FT}:\text{FF})$ , we simply set  $\text{STATE}(|l|) \leftarrow \text{EQ}$  instead of terminating.

To clear the to-do stack, we do the following while  $\text{TODO} \neq \Lambda$ : Set  $x \leftarrow \text{TODO}$  and  $\text{TODO} \leftarrow \text{LINK}(x)$ ; if  $\text{STATE}(x) = \text{EQ}$ , do nothing (no erp rule is needed to eliminate  $x$ ); otherwise set  $l \leftarrow (\text{STATE}(x) = \text{FT}? x:\bar{x})$ , output the erp rule  $l \leftarrow 1$ , and use the doubly linked lists to delete all clauses containing  $l$  and to delete  $\bar{l}$  from all clauses. (Those deletions update  $\text{TALLY}$  and  $\text{SIZE}$  fields, so they often contribute new entries to the to-do stack. Notice that if clause  $c$  loses a literal, we must recompute  $\text{SIG}(c)$ . If clause  $c$  disappears, we set  $\text{SIZE}(c) \leftarrow 0$ , and never use  $c$  again.)

Subsumption and strengthening are next in line. We give each clause  $c$  a new field  $\text{LINK}(c)$ , which is nonzero if and only if  $c$  appears on the “exploitation stack.” That stack begins at  $\text{EXP}$  and follows  $\text{LINK}$ s until reaching the nonzero sentinel value  $\Lambda'$ . All clauses are initially placed on the exploitation stack. Afterwards, whenever a literal  $\bar{l}$  is deleted from a clause  $c$ , either during unit conditioning or self-subsumption, we test if  $\text{LINK}(c) = 0$ ; if so, we put  $c$  back on the stack by setting  $\text{LINK}(c) \leftarrow \text{EXP}$  and  $\text{EXP} \leftarrow c$ .

To clear the subsumption stack, we first clear the to-do stack. Then, while  $\text{EXP} \neq \Lambda'$ , we set  $c \leftarrow \text{EXP}$ ,  $\text{EXP} \leftarrow \text{LINK}(c)$ , and do the following if  $\text{SIZE}(c) \neq 0$ : Remove clauses subsumed by  $c$ ; clear the to-do stack; and if  $\text{SIZE}(c)$  is still nonzero, strengthen clauses that  $c$  can improve, clear the to-do stack, and set  $\text{TIME}(c) \leftarrow T$  (see below).

All of this takes place before we even think about the elimination of variables. But rounds of variable elimination form the “outer level” of computation. Each variable  $x$  has yet another field,  $\text{STABLE}(x)$ , which is nonzero if and only if we need not attempt to eliminate  $x$ . This field is initially zero, but set nonzero when  $\text{STATE}(x) \leftarrow \text{EQ}$  or when an erp rule for  $x$  or  $\bar{x}$  is output. It is reset to zero whenever a variable is later “touched,” namely when  $x$  or  $\bar{x}$  appears in a deleted or self-subsumed clause. (In particular, every variable that appears in a new clause produced by resolution will be touched, because it will appear in at least one of the clauses that were replaced by new ones.)

If a round has failed to eliminate any variables, or if it has eliminated them all, we’re done. But otherwise there’s still work to do, because the new clauses can often be subsumed or strengthened. (Indeed, some of them might actually be duplicates.) Hence two *more* fields are introduced:  $\text{TIME}(l)$  for each literal and  $\text{TIME}(c)$  for each clause, initially zero. Let  $T$  be the number of the current elimination round. We set  $\text{TIME}(l) \leftarrow T$  for all literals  $l$  in all clauses that are replaced by resolution, and  $\text{TIME}(c) \leftarrow T$  is also set appropriately as mentioned above.

Introduce yet another field,  $\text{EXTRA}(c)$ , initially zero. It is reset to zero whenever  $\text{TIME}(c) \leftarrow T$ , and set to 1 whenever  $c$  is replaced by a new clause. For every literal  $l$  such that  $\text{STATE}(|l|) = 0$  and  $\text{TIME}(l) = T$  at the end of round  $T$ , set  $\text{EXTRA}(c) \leftarrow \text{EXTRA}(c) + 4$  for all clauses  $c$  that contain  $l$ , and  $\text{EXTRA}(c) \leftarrow \text{EXTRA}(c) \mid 2$  for all clauses  $c$  that contain  $\bar{l}$ . Then run through all clauses  $c$  for which  $\text{SIZE}(c) > 0$  and

unit conditioning  
pure literal elimination  
to-do stack  
erp rule  
doubly linked lists  
Subsumption  
strengthening  
exploitation stack  
sentinel value  
touched



$\text{TIME}(c) < T$ . If  $\text{SIZE}(c) = \text{EXTRA}(c) \gg 2$ , remove clauses subsumed by  $c$  and clear the exploitation stack. Also, if  $\text{EXTRA}(c) \& 3 \neq 0$ , we may be able to use  $c$  to strengthen other clauses—unless  $\text{EXTRA}(c) \& 1 = 0$  and  $\text{EXTRA}(c) \gg 2 < \text{SIZE}(c) - 1$ . Self-subsumption using  $l$  need not be attempted when  $\text{EXTRA}(c) \& 1 = 0$  unless  $\text{TIME}(\bar{l}) = T$  and  $\text{EXTRA}(c) \gg 2 = \text{SIZE}(c) - [\text{TIME}(l) = T]$ . Finally, reset  $\text{EXTRA}(c)$  to zero (even if  $\text{TIME}(c) = T$ ). [See Niklas Eén and Armin Biere, *LNCS* **3569** (2005), 61–75.]

**377.** Each vertex  $v$  of  $G$  corresponds to variables  $v_1, v_2, v_3$  in  $F$ ; each edge  $u - v$  corresponds to clauses  $(\bar{u}_1 \vee v_2), (\bar{u}_2 \vee v_3), (\bar{u}_3 \vee \bar{v}_1), (u_2 \vee \bar{v}_1), (u_3 \vee \bar{v}_2), (\bar{u}_1 \vee \bar{v}_3)$ . The longest paths in the dependency digraph for  $F$  have the form  $t_1 \rightarrow u_2 \rightarrow v_3 \rightarrow \bar{w}_1$  or  $t_1 \rightarrow \bar{u}_3 \rightarrow \bar{v}_2 \rightarrow \bar{w}_1$ , where  $t - u - v - w$  is a walk in  $G$ .

[A similar method reduces the question of finding an oriented cycle of length  $r$  in a given digraph to the question of finding a failed literal in some dependency digraph. The cycle detection problem has a long history; see N. Alon, R. Yuster, and U. Zwick, *Algorithmica* **17** (1997), 209–223. So any surprisingly fast algorithm to decide whether or not failed literals exist—that is, faster than  $n^{2\omega/(\omega+1)}$  when  $m = O(n)$  and matrix multiplication takes  $O(n^\omega)$ —would lead to surprisingly fast algorithms for other problems.]

**378.** The erp rule  $l \leftarrow l \vee (\bar{l}_1 \wedge \dots \wedge \bar{l}_q)$  will change any solution of  $F \setminus C$  into a solution of  $F$ . [See M. Järvisalo, A. Biere, and M. Heule, *LNCS* **6015** (2010), 129–144.]

(In practice it's sometimes possible to remove tens of thousands of blocked clauses. For example, all of the exclusion clauses (17) in the coloring problem are blocked, as are many of the clauses that arise in fault testing. Yet the author has yet to see a *single* example where blocked clause elimination is actually helpful in combination with transformations 1–4, which are already quite powerful by themselves.)

**379.** (Solution by O. Kullmann.) In general, *any* set  $F$  of clauses can be replaced by another set  $F'$ , whenever there's a variable  $x$  such that the elimination of  $x$  from  $F$  yields exactly the same clauses as the elimination of  $x$  from  $F'$ . In this case the elimination of  $a$  has this property. The erp rule  $a \leftarrow a \vee (\bar{b} \wedge \bar{c} \wedge d)$  is necessary and sufficient.

**380.** (a) Reverse self-subsumption weakens it to  $(a \vee b \vee c \vee d)$ , then to  $(a \vee b \vee c \vee d \vee e)$ , which is subsumed by  $(a \vee d \vee e)$ . [In general one can show that reverse self-subsumption from  $C$  leads to a subsumed clause if and only if  $C$  is certifiable from the other clauses.]

(b) Again we weaken to  $(a \vee b \vee c \vee d \vee e)$ ; but now we find this blocked by  $c$ .

(c) No erp rule is needed in (a), but we need  $c \leftarrow c \vee (\bar{a} \wedge \bar{b})$  in (b). [Heule, Järvisalo, and Biere, *LNCS* **6397** (2010), 357–371, call this “asymmetric elimination.”]

**381.** By symmetry, we'll remove the final clause. (Without it, the given clauses state that  $x_1 \leq x_2 \leq \dots \leq x_n$ ; with it, they state that all variables are equal.) Assume more generally that, for  $1 \leq j < n$ , every clause other than  $(\bar{x}_j \vee x_{j+1})$  that contains  $\bar{x}_j$  also contains either  $x_n$  or  $\bar{x}_i$  for some  $i < j$ . For  $1 \leq j < n - 1$  we can then weaken  $(x_1 \vee \dots \vee x_j \vee \bar{x}_n)$  to  $(x_1 \vee \dots \vee x_{j+1} \vee \bar{x}_n)$ . Finally,  $(x_1 \vee \dots \vee x_{n-1} \vee \bar{x}_n)$  can be eliminated because it is blocked by  $x_{n-1}$ .

Although we've eliminated only one clause,  $n - 1$  erp rules are actually needed to undo the process:  $x_1 \leftarrow x_1 \vee x_n$ ;  $x_2 \leftarrow x_2 \vee (\bar{x}_1 \wedge x_n)$ ;  $x_3 \leftarrow x_3 \vee (\bar{x}_1 \wedge \bar{x}_2 \wedge x_n)$ ;  $\dots$ ;  $x_{n-1} \leftarrow x_{n-1} \vee (\bar{x}_1 \wedge \dots \wedge \bar{x}_{n-2} \wedge x_n)$ . (Those rules, applied in reverse order, can however be simplified to  $x_j \leftarrow x_j \vee x_n$  for  $1 \leq j < n$ , because  $x_1 \leq \dots \leq x_n$  in any solution.)

[See Heule, Järvisalo, Biere, *EasyChair Proc. in Computing* **13** (2013), 41–46.]

**382.** See M. J. H. Heule, M. Järvisalo, and A. Biere, *LNCS* **6695** (2011), 201–215.

**383.** (a) In a learning step, let  $\Phi' = \Phi$  and  $\Psi' = \Psi \cup C$ . In a forgetting step, let  $\Phi' = \Phi$  and  $\Psi = \Psi' \cup C$ . In a hardening step, let  $\Phi' = \Phi \cup C$  and  $\Psi = \Psi' \cup C$ . In a softening

Eén  
Biere  
dependency digraph  
walk  
oriented cycle  
cycle detection problem  
Alon  
Yuster  
Zwick  
matrix multiplication  
Järvisalo  
Biere  
exclusion clauses  
coloring  
fault testing  
author  
Kullmann  
elimination of  $x$   
certifiable  
blocked  
Heule  
Järvisalo  
Biere  
asymmetric elimination  
blocked  
Heule  
Järvisalo  
Biere  
Heule  
Järvisalo  
Biere

step, let  $\Phi = \Phi' \cup C$  and  $\Psi' = \Psi \cup C$ . In all four cases it is easy to verify that  $(\text{sat}(\Phi) \iff \text{sat}(\Phi \cup \Psi))$  implies  $(\text{sat}(\Phi) \iff \text{sat}(\Phi') \iff \text{sat}(\Phi' \cup \Psi'))$ , where  $\text{sat}(G)$  means “ $G$  is satisfiable,” because  $\text{sat}(G \cup G') \implies \text{sat}(G)$ . Thus the assertions are invariant.

(b) Each erp rule allows us to go one step backward, until reaching  $F$ .

(c) The first (softening) step is fine, because both  $\Phi = (x)$  and  $\Phi \setminus (x) = 1$  are satisfiable, and because the erp rule unconditionally makes  $x$  true. But the second (learning) step is flawed, because  $\text{sat}(\Phi \cup \Psi)$  does not imply  $\text{sat}(\Phi \cup \Psi \cup C)$  when  $\Phi \cup \Psi = (x)$  and  $C = (\bar{x})$ . (This example explains why the criterion for learning is not simply ‘ $\text{sat}(\Phi) \implies \text{sat}(\Phi \cup C)$ ’ as it essentially is for softening.)

(d) Yes, because  $C$  is also certifiable for  $\Phi \cup \Psi$ .

(e) Yes, after softening it. No erp rule is needed, because  $\Phi \setminus C \vdash C$ .

(f) A soft clause can be discarded whether or not it is subsumed. To discard a hard clause that is subsumed by a soft clause, first harden the soft one. To discard a hard  $C$  that is subsumed by a hard  $C'$ , weaken  $C$  and then discard it. (The weakening step is clearly permissible, and no erp rule is needed.)

(g) If  $C$  contains  $\bar{x}$  and  $C'$  contains  $x$  and  $C \setminus \bar{x} \subseteq C' \setminus x$ , we can learn the soft clause  $C \diamond C' = C' \setminus x$ , then use it to subsume  $C'$  as in (f).

(h) Forget all soft clauses that contain  $x$  or  $\bar{x}$ . Then let  $C_1, \dots, C_p$  be the hard clauses containing  $x$ , and  $C'_1, \dots, C'_q$  those containing  $\bar{x}$ . Learn all the (soft) clauses  $C_i \diamond C'_j$ , and harden them, noting that they don’t involve  $x$ . Weaken each  $C_i$ , with erp rule  $x \leftarrow x \vee \bar{C}_i$ , and forget it; also weaken and forget each  $C'_j$ , with erp rule  $x \leftarrow x \wedge C'_j$ . (One can show that either of the erp rules in (161) would also suffice.)

(i) Whenever  $\Phi \cup \Psi$  is satisfiable, so is  $\Phi \cup \Psi \cup \{(x \vee z), (y \vee z), (\bar{x} \vee \bar{y} \vee \bar{z})\}$ , because we can always set  $z \leftarrow \bar{x} \vee \bar{y}$ .

[Reference: M. Järvisalo, M. Heule, and A. Biere, *LNCS 7364* (2012), 355–370. Notice that, by exercise 368, parts (f) and (h) justify the use of unit conditioning.]

**384.** Whenever we have a solution to  $\Phi \setminus C$  that falsifies  $C$ , we will show that  $\Phi$  is satisfied by making  $l$  true; hence softening  $C$  is permissible, with erp rule  $l \leftarrow l \vee \bar{C}$ .

To prove that claim, notice that a problem could arise only in a hard clause  $C'$  that contains  $\bar{l}$ . But if all other literals of  $C'$  are false in the given solution, then all literals of  $C \diamond C'$  are false, contradicting the assumption that  $(\Phi \setminus C) \wedge \overline{C \diamond C'} \vdash_1 \epsilon$ .

(Such clauses  $C$  are “resolution certifiable” with respect to  $\Phi \setminus C$ . Blocked clauses are a very special case. Similarly, we can safely learn any clause that is resolution certifiable with respect to  $\Phi \cup \Psi$ .)

**385.** (a) True, because  $\bar{C} \wedge l \vdash_1 \epsilon$ .

(b)  $\bar{1}$  is implied, not certifiable;  $\bar{1}2$  is certifiable, not absorbed;  $\bar{1}23$  is absorbed.

(c, d) If  $C$  is any clause and  $l$  is any literal, then  $F \wedge \bar{C} \vdash_1 l$  implies  $F' \wedge \bar{C} \vdash_1 l$ , because unit propagation in  $F$  carries over to unit propagation in  $F'$ .

**386.** (a) The trail contained exactly  $\text{score}(F, C, l)$  literals when decision  $\bar{l}$  was made at level  $d$ . The clause learned from the ensuing conflict causes at least one new literal to be implied at level  $d' < d$ .

(b) The score can’t decrease when  $F$  grows.

(c) Each  $l \in C$  needs at most  $n$  helpful rounds to make  $\text{score}(F, C, l) = \infty$ .

(d) Suppose, for example,  $F = (a \vee \bar{d}) \wedge (a \vee b \vee e \vee l) \wedge (\bar{a} \vee c) \wedge (\bar{b}) \wedge (c \vee d \vee \bar{e} \vee l)$  and  $C = (a \vee b \vee c \vee d \vee l)$ . The helpful sequences of decisions are  $(\bar{a}, \bar{c}, \bar{l})$ ,  $(\bar{c}, \bar{d}, \bar{l})$ ,  $(\bar{d}, \bar{a}, \bar{c}, \bar{l})$ ,  $(\bar{d}, \bar{c}, \bar{l})$ , and they occur with probabilities  $\frac{1}{10} \frac{1}{6} \frac{1}{4}$ ,  $\frac{1}{10} \frac{1}{6} \frac{1}{4}$ ,  $\frac{1}{10} \frac{1}{8} \frac{1}{6} \frac{1}{4}$ ,  $\frac{1}{10} \frac{1}{8} \frac{1}{4}$ .

invariant  
Järvisalo  
Heule  
Biere  
unit conditioning  
resolution certifiable  
Blocked clauses  
RAT, see resolution certifiable clauses

In general if a decision is to be made and  $j$  elements of  $\overline{C}$  are not yet in the trail, the probability that a suitable decision will be made at random is at least

$$f(n, j) = \min\left(\frac{j-1}{2n}f(n-1, j-1), \frac{j-2}{2(n-1)}f(n-2, j-2), \dots, \frac{1}{2(n-j+2)}f(n-j+1, 1), \frac{1}{2(n-j+1)}\right) = \frac{(j-1)!}{2^j n^j}.$$

(e) The waiting time to absorb each clause  $C_i$  is a geometric distribution whose mean is  $\leq 4n^{|C_i|}$ , repeated at most  $|C_i|n$  times.

References: K. Pipatsrisawat and A. Darwiche, *Artif. Intell.* **175** (2011), 512–525; A. Atserias, J. K. Fichte, and M. Thurley, *J. Artif. Intell. Research* **40** (2011), 353–373.

**387.** We may assume that  $G$  and  $G'$  have no isolated vertices. Letting variable  $vv'$  mean that  $v$  corresponds to  $v'$ , we need the clauses  $(\overline{uv'} \vee \overline{vv'})$  for  $u < v$  and  $(\overline{vv'} \vee \overline{v'v'})$  for  $u' < v'$ . Also, for each  $u < v$  with  $u - v$  in  $G$ , we introduce auxiliary variables  $uu'vv'$  for each edge  $u' - v'$  in  $G'$ , with clauses  $(\overline{uu'vv'} \vee \overline{uu'}) \wedge (\overline{uu'vv'} \vee \overline{vv'}) \wedge (\vee\{uu'vv' \mid u' - v' \text{ in } G'\})$ . The variables  $vv'$  and  $uu'vv'$  can be restricted to cases where  $\text{degree}(u) \leq \text{degree}(u')$  and  $\text{degree}(v) \leq \text{degree}(v')$ .

**388.** (a) Can the complete graph  $K_k$  be embedded in  $G$ ? (b) Can  $G$  be embedded in the complete  $k$ -partite graph  $K_{n, \dots, n}$ , where  $G$  has  $n$  vertices? (c) Can the cycle  $C_n$  be embedded in  $G$ ?

**389.** This is a graph embedding problem, with  $G'$  the  $4 \times 4$  (king U knight) graph and with  $G$  defined by edges T — H, H — E, . . . , N — G. The adjacent Ms can be avoided by changing 'PROGRAMMING' to either 'PROGRAMXING' or 'PROGRAXMING'.

Algorithm C needs less than 10 megamems to find the first solution below. Furthermore, if the blank space can also be moved, the algorithm will rather quickly also find solutions with just five knight moves (the minimum), or 17 of them (the max):

|   |   |   |   |
|---|---|---|---|
| U | P | C | F |
| M | M | O | □ |
| I | T | R | A |
| N | G | E | H |

|   |   |   |   |
|---|---|---|---|
| M | M | I | N |
| A | P | O | G |
| H | R | □ | F |
| U | T | E | C |

|   |   |   |   |
|---|---|---|---|
| H | N | U | F |
| E | M | O | I |
| G | T | □ | P |
| A | R | M | C |

**390.** Let  $d(u, v)$  be the distance between vertices  $u$  and  $v$ . Then  $d(v, v) = 0$  and  $d(u, v) \leq j + 1 \iff d(u, v) \leq j$  or  $d(u, w) \leq j$  for some  $w \in N(v) = \{w \mid w - v\}$ . (\*)

In parts (a), (d), we introduce variables  $v_j$  for each vertex  $v$  and  $0 \leq j \leq k$ . In part (c) we do this for  $0 \leq j < n$ . But parts (b), (e), (f) use just  $n$  variables,  $\{v \mid v \in V\}$ .

(a) Clauses  $(s_0) \wedge \vee_{v \in V \setminus s} (\overline{v_0}) \wedge \vee_{v \in V} (\overline{v_{j+1}} \vee v_j \vee \vee_{w - v} w_j)$  are satisfied only if  $v_j \leq [d(s, v) \leq j]$ ; hence the additional clause  $(t_k)$  is also satisfied only if  $d(s, t) \leq k$ . Conversely, if  $d(s, t) \leq k$ , all clauses are satisfied by setting  $v_j \leftarrow [d(s, v) \leq j]$ .

(b) There's a path from  $s$  to  $t$  if and only if there's a subset  $H \subseteq V$  such that  $s \in H, t \in H$ , and every other vertex of the induced graph  $G|H$  has degree 0 or 2. [The vertices on a shortest path from  $s$  to  $t$  yield one such  $H$ . Conversely, given  $H$ , we can find vertices  $v_j \in H$  such that  $s = v_0 - v_1 - \dots - v_k = t$ .]

We can represent that criterion via clauses on the binary variables  $v = [v \in H]$  by asserting  $(s) \wedge (t)$ , together with clauses to ensure that  $\Sigma(s) = \Sigma(t) = 1$ , and that  $\Sigma(v) \in \{0, 2\}$  for all  $v \in H \setminus \{s, t\}$ , where  $\Sigma(v) = \sum_{w \in N(v)} w$  is the degree of  $v$  in  $G|H$ . The number of such clauses for each  $v$  is at most  $6|N(v)|$ , because we can append  $\overline{v}$

geometric distribution  
 Pipatsrisawat  
 Darwiche  
 Atserias  
 Fichte  
 Thurley  
 isolated vertices  
 auxiliary variables  
 complete graph  
 complete  $k$ -partite graph  
 cycle  
 graph embedding  
 distance  $d(u, v)$  in a graph  
 induced graph  
 shortest path

to each clause of (18) and (19) when  $r = 2$ , and  $|N(v)|$  additional clauses will rule out  $\Sigma(v) < 2$ . Altogether there are  $O(m)$  clauses, because  $\sum_{v \in V} |N(v)| = 2m$ .

[Similar but simpler alternatives, such as (i) to require  $\Sigma(v) \in \{0, 2\}$  for all  $v \in V \setminus \{s, t\}$ , or (ii) to require  $\Sigma(v) \geq 2$  for all  $v \in H \setminus \{s, t\}$ , do *not* work: Counterexamples are (i)  $s \text{---} \textcircled{v} \text{---} t$  and (ii)  $s \text{---} \textcircled{v} \text{---} t$ . Another solution, more cumbersome, associates a Boolean variable with each *edge* of  $G$ .]

(c) Let  $s$  be any vertex; use (a), plus  $(v_{n-1})$  for all  $v \in V \setminus s$ .

(d) Clauses  $(s_0) \wedge \bigvee_{j=0}^{k-1} \bigvee_{v \in V} \bigvee_{w \in N(v)} (\bar{v}_k \vee w_{k+1})$  are satisfied only if we have  $v_j \geq [d(s, v) \leq j]$ ; hence the additional clause  $(\bar{t}_k)$  cannot also be satisfied when  $d(s, t) \leq k$ . Conversely, if  $d(s, t) > k$  we can set  $v_j \leftarrow [d(s, v) \leq j]$ .

(e)  $(s) \wedge (\bigvee_{v \in V} \bigvee_{w \in N(v)} (\bar{v} \vee w)) \wedge (\bar{t})$ .

(f) Letting  $s$  be any vertex, use  $(s) \wedge (\bigvee_{v \in V} \bigvee_{w \in N(v)} (\bar{v} \vee w)) \wedge (\bigvee_{v \in V \setminus s} \bar{v})$ .

[Similar constructions work with digraphs and strong connectivity. Parts (d)–(f) of this exercise were suggested by Marijn Heule. Notice that parts (a) and (c)–(f) construct renamed Horn clauses, which work very efficiently (see exercise 444).]

**391.** (a) Let  $d - 1 = (q_{l-1} \dots q_0)_2$ . To ensure that  $(x_{l-1} \dots x_0)_2 < d$  we need the clauses  $(\bar{x}_i \vee \bigvee \{\bar{x}_j \mid j > i, q_j = 1\})$  whenever  $q_i = 0$ . The same holds for  $y$ .

To enforce  $x \neq y$ , introduce the clause  $(a_{l-1} \vee \dots \vee a_0)$  in auxiliary variables  $a_{l-1} \dots a_0$ , together with  $(\bar{a}_j \vee x_j \vee y_j) \wedge (\bar{a}_j \vee \bar{x}_j \vee \bar{y}_j)$  for  $0 \leq j < l$  (see (172)).

(b) Now  $x \neq y$  is enforced via clauses of length  $2l$ , which state that we don't have  $x = y = k$  for  $0 \leq k < d$ . For example, the appropriate clause when  $l = 3$  and  $k = 5$  is  $(\bar{x}_2 \vee \bar{y}_2 \vee x_1 \vee y_1 \vee \bar{x}_0 \vee \bar{y}_0)$ .

(c) Use the clauses of (b) for  $0 \leq k < 2d - 2^l$ , plus clauses of length  $2l - 2$  for  $d \leq k < 2^l$  stating that we don't have  $(x_{l-1} \dots x_1)_2 = (y_{l-1} \dots y_1)_2 = k$ . (The encodings in (b) and (c) are identical when  $d = 2^l$ .)

[See A. Van Gelder, *Discrete Applied Mathematics* **156** (2008), 230–243.]

**392.** (a) [Puzzle (ii) was introduced by Sam Loyd in the *Boston Herald*, 13 November 1904; page 27 of his *Cyclopedia* (1914) states that he'd created a puzzle like (i) at age 9! Puzzle (iv) is by H. E. Dudeney, *Strand* **42** (1911), 108, slightly modified. Puzzle (iii) is from the Grabarchuks' *Big, Big, Big Book of Brainteasers* (2011), #196; puzzle (v) was designed by Serhiy A. Grabarchuk in 2015.]

|   |   |   |   |   |
|---|---|---|---|---|
| A | A | A | A | A |
| A | B | B | B | B |
| A | A | A | A | A |
| C | C | C | C | A |
| A | A | A | A | A |

(i)

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| A | A | A | D | D | D | D |   |
| A | D | D | D | E | E | D |   |
| A | A | A | A | A | E | D |   |
| C | C | C | C | A | E | D |   |
| C | A | A | A | C | A | E | D |
| C | A | B | A | A | A | E | B |
| C | A | B | B | B | E | E | B |
| C | A | A | A | B | B | B | B |

(ii)

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| A | A | A | B | B | B | B | B |
| A | C | A | A | A | C | C | B |
| A | C | C | A | C | B | B | B |
| A | A | D | C | C | B | B | E |
| F | A | D | D | D | E | B | E |
| F | A | A | A | D | E | B | E |
| F | D | D | D | D | E | B | E |
| F | F | F | F | F | E | E | E |

(iii)

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C | G | G | G | G | H | A | A | A | A | A | A | A | C | C | C | C |   |   |
| G | F | F | F | F | G | H | A | C | C | C | C | C | C | C | E | E | E | C |
| G | F | B | F | G | H | A | C | E | E | E | E | E | E | D | E | C |   |   |
| C | F | B | F | G | H | A | A | A | A | A | A | A | A | D | E | E |   |   |
| C | F | B | F | G | H | A | A | A | A | A | A | A | A | D | D | D | D |   |
| F | F | B | F | F | G | G | G | G | G | G | G | H | H | A | D |   |   |   |
| F | B | B | J | J | F | F | F | F | F | F | F | F | F | F | H | A | D |   |
| B | J | J | J | J | H | H | H | H | H | H | H | H | H | H | A | D |   |   |
| B | J | J | J | J | H | H | H | H | H | H | H | H | H | H | A | A | A | D |
| B | B | B | J | J | J | J | J | J | J | J | J | J | J | J | J | J | J | J |

(iv)

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| E | E | E | E | E | A | A | A | A | A | A | A | A | A | A | A | A | A | A |
| E | D | D | E | E | C | C | C | C | C | C | C | C | C | C | C | C | C | C |
| E | D | B | D | C | E | C | B | B | B | B | B | B | B | B | B | B | B | B |
| E | D | B | D | C | E | C | B | F | F | F | F | F | F | F | F | F | F | F |
| E | D | B | D | C | E | C | B | F | D | D | D | D | D | D | D | D | D | D |
| E | D | F | B | C | C | E | C | B | F | D | E | A | E | D | F | B | C | A |
| E | D | F | B | B | B | B | B | B | B | F | D | E | A | C | B | F | B | C |
| E | D | F | F | F | F | F | F | F | F | F | D | E | A | C | B | B | B | C |
| E | D | D | D | D | D | D | D | D | D | D | D | E | A | C | C | C | C | C |
| E | E | E | E | E | E | E | E | E | E | E | E | E | E | E | E | E | E | E |

(v)

(b) [Puzzle (vi) is an instance of the odd-even transposition sort, exercise 5.3.4–37. Eight order-reversing connections would be impossible with only eight columns, instead of the nine in (vii), because the permutation has too many inversions.]

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| A | B | B | D | D | F | F | H | H |
| B | A | D | B | F | D | H | F | G |
| C | D | A | F | B | H | D | G | F |
| D | C | F | A | H | B | G | D | E |
| E | F | C | H | A | G | B | E | D |
| F | E | H | C | G | A | E | B | C |
| G | H | E | G | C | E | A | C | B |
| H | G | E | E | C | C | A | A |   |

(vi)

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| A | B | B | D | D | F |   |   |
| B | A | D | B | F | D | F | G |
| C | D | A | F | B | D | G | F |
| D | C | F | A | B | G | D | E |
| E | F | C | A | G | B | E | D |
| F | E | C | G | A | E | B | C |
| G | E | G | C | E | A | C | B |
| G | E | E | C | C | A | A |   |

(vii)

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| A | B | B | C | C | D | D | A |
| D | A | C | B | D | C | A | B |
| D | C | A | B | D | A | C | B |
| C | D | B | A | A | D | B | C |
| C | B | D | B | D | A | D | B |
| B | C | C | D | B | A | B | D |
| B | A | D | C | A | B | A | D |
| A | D | A | A | C |   | B | A |

(viii)

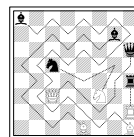
digraphs  
strong connectivity  
Heule  
renamed  
Horn clauses  
Van Gelder  
Loyd  
Dudeney  
Grabarchuks'  
odd-even transposition sort

(c) Let  $d_j = \sum_{i=1}^j (|T_i| - 1)$  and  $d = d_t$ . We introduce variables  $v_i$  for  $1 \leq i \leq d$ , and the following clauses for  $1 \leq j \leq t$  and  $d_{j-1} < i \leq d_j$ :  $(\bar{v}_{i'} \vee \bar{v}_i)$  for  $1 \leq i' \leq d_{j-1}$ ; the clauses of answer 390(b) on variables  $v_i$ , where  $s$  is the  $(i - d_{j-1})$ th element of  $T_j$  and  $t$  is the last element. These clauses ensure that the sets  $V_j = \{v \mid v_{d_{j-1}+1} \vee \dots \vee v_{d_j}\}$  are disjoint, and that  $V_j$  contains a connected component  $S_j \supseteq T_j$ .

We also assert  $(\bar{v}_i)$  for  $1 \leq i \leq d$ , whenever  $T_j$  is a singleton set  $\{v\}$ .

[For the more general “Steiner tree packing” problem, see M. Grötschel, A. Martin, and R. Weismantel, *Math. Programming* **78** (1997), 265–281.]

**393.** A construction somewhat like that of answer 392(c) can be used with five different  $8 \times 8$  graphs, one for the moves of each white-black pair  $S_j$ . But we need to keep track of the edges used, not vertices, in order to prohibit edges that cross each other. Additional clauses will rule that out.



Steiner tree packing  
Grötschel  
Martin  
Weismantel  
*langford'''(n)*  
Prestwich  
axiom clauses  
Tamura  
Taga  
Kitagawa  
Banbara  
channeling  
author  
direct encoding  
triangle

**394.** Call these clauses *langford'''(n)*. [Steven Prestwich described a similar method in *Trends in Constraint Programming* (Wiley, 2007), 269–274.] Typical results are:

|                        | variables | clauses | Algorithm D | Algorithm L | Algorithm C |         |
|------------------------|-----------|---------|-------------|-------------|-------------|---------|
| <i>langford'''(9)</i>  | 206       | 1157    | 131 Mμ      | 18 Mμ       | 22 Mμ       | (UNSAT) |
| <i>langford'''(13)</i> | 403       | 2935    | 1425 Gμ     | 44 Gμ       | 483 Gμ      | (UNSAT) |
| <i>langford'''(16)</i> | 584       | 4859    | 713 Kμ      | 42 Mμ       | 343 Kμ      | (SAT)   |
| <i>langford'''(64)</i> | 7352      | 120035  | (huge)      | (big)       | 71 Mμ       | (SAT)   |

**395.** The color of each vertex  $v$  gets binary axiom clauses  $(\bar{v}^{j+1} \vee v^j)$  for  $1 \leq j < d-1$ , as in (164). And for each edge  $u - v$  in the graph, we want  $d$  clauses  $(\bar{u}^{j-1} \vee u^j \vee \bar{v}^{j-1} \vee v^j)$  for  $1 \leq j \leq d$ , omitting  $\bar{u}^0$  and  $\bar{v}^0$  when  $j = 1$ ,  $u^d$  and  $v^d$  when  $j = d$ .

[The surprising usefulness of order encoding in graph coloring was first noticed by N. Tamura, A. Taga, S. Kitagawa, and M. Banbara in *Constraints* **14** (2009), 254–272.]

**396.** First we have  $(\bar{x}^{j+1} \vee x^j)$  and  $(\bar{\hat{x}}^{j+1} \vee \hat{x}^j)$  for  $1 \leq j < d$ . Then we have “channeling” clauses to ensure that  $j \leq x < j+1 \iff j\pi \leq x\pi < (j+1)\pi$  for  $0 \leq j < d$ :

$$(\bar{x}^j \vee x^{j+1} \vee \hat{x}^{j\pi}) \wedge (\bar{x}^j \vee x^{j+1} \vee \bar{\hat{x}}^{j\pi+1}) \wedge (\bar{\hat{x}}^{j\pi} \vee \hat{x}^{j\pi+1} \vee x^j) \wedge (\bar{\hat{x}}^{j\pi} \vee \hat{x}^{j\pi+1} \vee \bar{x}^{j+1}).$$

(These clauses should be either shortened or omitted in boundary cases, because  $x^0$  and  $\hat{x}^0$  are always true, while  $x^d$  and  $\hat{x}^d$  are always false. We obtain  $6d-8$  clauses for each  $x$ .)

With such clauses for every vertex of a graph, together with clauses based on adjacent vertices and cliques, we obtain encodings for  $n$ -coloring the  $n \times n$  queen graph that involve  $2(n^3 - n^2)$  variables and  $\frac{5}{3}n^4 + 4n^3 + O(n^2)$  clauses, compared to  $n^3 - n^2$  variables and  $\frac{5}{3}n^4 - n^3 + O(n^2)$  clauses with single cliques and (162) alone. Typical running times with Algorithm C and single cliques are 323 Kμ, 13.1 Mμ, 706 Gμ for  $n = 7, 8, 9$ ; with double clique-ing they become 252 Kμ, 1.97 Mμ, 39.8 Gμ, respectively.

The double clique hints turn out to be mysteriously ineffective when  $\pi$  is the standard organ-pipe permutation  $(0\pi, 1\pi, \dots, (d-1)\pi) = (0, 2, 4, \dots, 5, 3, 1)$  instead of its inverse. Random choices of  $\pi$  when  $n = 8$  yielded significant improvement almost half the time, in the author’s experiments; but they had negligible effect in 1/3 of the cases.

Notice that the example  $\pi$  for  $d = 4$  yields  $x^1 = \bar{x}_0, x^3 = x_3, \hat{x}^1 = \bar{x}_2, \hat{x}_3 = x_1$ . Hence the direct encoding is essentially present as part of this redundant representation, and the hints  $(\bar{u}^3 \vee \bar{v}^3) \wedge (u^1 \vee v^1) \vee (\bar{\hat{u}}^3 \vee \bar{\hat{v}}^3) \wedge (\hat{u}^1 \vee \hat{v}^1)$  for 2-cliques  $\{u, v\}$  are equivalent to (16). But the hints  $(u^2 \vee v^2 \vee w^2) \wedge (\bar{u}^2 \vee \bar{v}^2 \vee \bar{w}^2) \wedge (\hat{u}^2 \vee \hat{v}^2 \vee \hat{w}^2) \wedge (\bar{\hat{u}}^2 \vee \bar{\hat{v}}^2 \vee \bar{\hat{w}}^2)$  that apply when  $\{u, v, w\}$  is a triangle give additional logical power.

**397.** There are  $(p-2)d$  binary clauses  $(\bar{y}_j^{i+1} \vee y_j^i)$  for  $1 \leq i < p-1$ , together with the  $(2p-2)d$  clauses  $(\bar{x}_i^j \vee x_i^{j+1} \vee y_j^i) \wedge (\bar{x}_{i-1}^j \vee x_{i-1}^{j+1} \vee \bar{y}_j^i)$  for  $1 \leq i < p$ , all for  $0 \leq j < d$ . The hint clauses  $(x_0^{p-1} \vee \dots \vee x_{p-1}^{p-1}) \wedge (\bar{x}_0^{d-p+1} \vee \dots \vee \bar{x}_{p-1}^{d-p+1})$  are also valid.

(This setup corresponds to putting  $p$  pigeons into  $d$  holes, so we can usually assume that  $p \leq d$ . If  $p \leq 4$  it is better to use  $\binom{p}{2}d$  clauses as in exercise 395. Notice that we obtain an interesting representation of *permutations* when  $p = d$ . In that case  $y$  is the inverse permutation; hence  $(2d-2)p$  additional clauses corresponding to  $y_j = i \implies x_i = j$  are also valid, as well as two hint clauses for  $y$ .)

A related idea, but with direct encoding of the  $x$ 's, was presented by I. Gent and P. Nightingale in *Proceedings of the International Workshop on Modelling and Reformulating Constraint Satisfaction Problems 3* (2004), 95–110.

**398.** We could construct  $(3p-4)d$  binary clauses that involve  $y_j^i$ , as in exercise 397. But it's better just to have  $(3p-6)d$  clauses for the at-most-one constraints  $x_{0k} + x_{1k} + \dots + x_{(p-1)k} \leq 1$ ,  $0 \leq k < d$ .

**399.** (a)  $d^2 - t$  preclusion clauses (binary); or  $2d$  support clauses (total length  $2(d+t)$ ).

(b) If unit propagation derives  $\bar{v}_j$  from  $(\bar{u}_i \vee \bar{v}_j)$ , we knew  $u_i$ ; hence (17) gives  $\bar{u}_{i'}$  for all  $i' \neq i$ , and  $\bar{v}_j$  follows from the support clause that contains it.

(c) If unit propagation derives  $\bar{v}_j$  from its support clause, we knew  $\bar{u}_i$  for all  $i \neq j$ ; hence (15) gives  $u_j$ , and  $\bar{v}_j$  follows from (16). Or if unit propagation derives  $u_i$  from that support clause, we knew  $v_j$  and  $\bar{u}_{i'}$  for all  $i' \notin \{i, j\}$ ; hence  $\bar{u}_j$  from (16),  $u_i$  from (15).

(d) A trivial example has no legal pairs; then unit propagation never gets started from binary preclusions, but the (unit) support clauses deduce all. A more realistic example has  $d = 3$  and all pairs legal except (1,1) and (1,2), say; then we have  $(15) \wedge (17) \wedge (\bar{u}_1 \vee \bar{v}_1) \wedge (\bar{u}_1 \vee \bar{v}_2) \wedge (\bar{v}_3) \not\vdash_1 \bar{u}_1$  but  $(15) \wedge (17) \wedge (\bar{u}_1 \vee v_3) \wedge (\bar{v}_3) \vdash_1 \bar{u}_1$ .

[Preclusion was introduced by S. W. Golomb and L. D. Baumert, *JACM* **12** (1965), 521–523. The support encoding was introduced by I. P. Gent, *European Conf. on Artificial Intelligence* **15** (2002), 121–125, based on work of S. Kasif, *Artificial Intelligence* **45** (1990), 275–286.]

**400.** This problem has  $n$  variables  $q_1, \dots, q_n$  with  $n$  values each; thus there are  $n^2$  Boolean values, with  $q_{ij} = [q_i = j] = [\text{there's a queen in row } i \text{ and column } j]$ . The constraint between  $q_i$  and  $q_j$  is that  $q_i \notin \{q_j, q_j + i - j, q_j - i + j\}$ ; so it turns out that there are  $n$  at-least-one clauses, plus  $(n^3 - n^2)/2$  at-most-one clauses, plus either  $n^3 - n^2$  support clauses or  $n^3 - n^2 + \binom{n}{3}$  preclusion clauses. In this problem each support clause has at least  $n-2$  literals, so the support encoding is much larger.

Since the problem is easily satisfiable, it makes sense to try WalkSAT. When  $n = 20$ , Algorithm W typically finds a solution from the preclusion clauses after making fewer than 500 flips; its running time is about  $500 K\mu$ , including about  $200 K\mu$  just to read the input. With the support clauses, however, it needs about 10 times as many flips and consumes about 20 times as many mems, before succeeding.

Algorithm L is significantly worse: it consumes  $50 M\mu$  with preclusion clauses,  $11 G\mu$  with support clauses. Algorithm C is the winner, with about  $400 K\mu$  (preclusion) versus  $600 K\mu$  (support).

Of course  $n = 20$  is pretty tame; let's consider  $n = 100$  queens, when there are 10,000 variables and more than a million clauses. Algorithm L is out of the picture; in the author's experiments, it showed no indication of being even close to a solution after  $20 T\mu$ ! But Algorithm W solves that problem in  $50 M\mu$ , via preclusion, after making only about 5000 flips. Algorithm C wins again, polishing it off in  $29 M\mu$ . With

pigeons  
permutations  
inverse permutation  
direct encoding  
Gent  
Nightingale  
at-most-one constraints  
Golomb  
Baumert  
Gent  
Kasif  
WalkSAT  
author

the support clauses, nearly 100 million literals need to be input, and Algorithm W is hopelessly inefficient; but Algorithm C is able to finish after about 200 Mμ.

The preclusion clauses actually allow us to omit the at-most-one clauses in this problem, because two queens in the same row will be ruled out anyway. This trick improves the run time when  $n = 100$  to 35 Mμ for Algorithm W.

We can also append support clauses for the columns as well as the rows. This idea roughly halves the search space, but it gives no improvement because twice as many clauses must be handled. *Bottom line:* Support clauses don't support  $n$  queens well.

(However, if we seek *all* solutions to the  $n$  queens problem instead of stopping with the first one, using a straightforward extension of Algorithm D (see exercise 122), the support clauses proved to be definitely better in the author's experiments.)

401. (a)  $y^j = x^{2j-1}$ . (b)  $z^j = x^{3j-1}$ . In general  $w = \lfloor (x+a)/b \rfloor \iff w^j = x^{bj-a}$ .

402. (a)  $\bigwedge_{j=1}^{\lfloor d/2 \rfloor} (\bar{x}^{2j-1} \vee x^{2j})$ ; (b)  $\bigwedge_{j=1}^{\lfloor d/2 \rfloor} (\bar{x}^{2j-2} \vee x^{2j-1})$ ; omit  $\bar{x}^0$  and  $x^d$ .

403. (a)  $\bigwedge_{j=1}^{d-1} (\bar{x}^j \vee \bar{y}^j \vee z^j)$ ; (b)  $\bigwedge_{j=1}^{d-1} ((\bar{x}^j \vee z^j) \wedge (\bar{y}^j \vee z^j))$ ; (c)  $\bigwedge_{j=1}^{d-1} ((x^j \vee \bar{z}^j) \wedge (y^j \vee \bar{z}^j))$ ; (d)  $\bigwedge_{j=1}^{d-1} (x^j \vee y^j \vee \bar{z}^j)$ .

404. (a)  $\bigwedge_{j=0}^{d-1} (\bar{x}^j \vee x^{j+1} \vee \bar{y}^{j+1-a} \vee y^{j+a})$ . (As usual, omit literals with superscripts  $\leq 0$  or  $\geq d$ . If  $a > 1$  this encoding is unsymmetrical, with one clause for each value of  $x$ .)

(b)  $\bigwedge_{j=0}^{d-a} ((p \vee \bar{x}^j \vee y^{j+a}) \wedge (\bar{p} \vee x^{j+a} \vee \bar{y}^j))$ ;  $p$  is the auxiliary variable.

405. (a) If  $a < 0$  we can replace  $ax$  by  $(-a)\bar{x}$  and  $c$  by  $c + a - ad$ , where  $\bar{x}$  is given by (165). A similar reduction applies if  $b < 0$ . Cases with  $a, b$ , or  $c = 0$  are trivial.

(b) We have  $13x + 8\bar{y} \leq 63 \iff \text{not } 13x + 8\bar{y} \geq 64 \iff \text{not } (P_0 \text{ or } \dots \text{ or } P_{d-1}) \iff \text{not } P_0 \text{ and } \dots \text{ and not } P_{d-1}$ , where  $P_j = 'x \geq j \text{ and } \bar{y} \geq \lceil (64 - 13j)/8 \rceil'$ . This approach yields  $\bigwedge_{j=0}^7 (\bar{x}^j \vee y^{8 - \lceil (64 - 13j)/8 \rceil})$ , which simplifies to  $(\bar{x}^1 \vee y^1) \wedge (\bar{x}^2 \vee y^3) \wedge (\bar{x}^3 \vee y^4) \wedge (\bar{x}^4 \vee y^6) \wedge (\bar{x}^5)$ . (Notice that we could have defined  $P_j = '\bar{y} \geq j \text{ and } x \geq \lceil (64 - 8j)/13 \rceil'$  instead, thereby obtaining the less efficient encoding  $(\bar{x}^5) \wedge (y^7 \vee \bar{x}^5) \wedge (y^6 \vee \bar{x}^4) \wedge (y^5 \vee \bar{x}^4) \wedge (y^4 \vee \bar{x}^3) \wedge (y^3 \vee \bar{x}^2) \wedge (y^2 \vee \bar{x}^2) \wedge (y^1 \vee \bar{x}^1)$ ; it's better to discriminate on the variable with the larger coefficient.)

(c) Similarly,  $13\bar{x} + 8y \leq 90$  gives  $(x^5 \vee \bar{y}^7) \wedge (x^4 \vee \bar{y}^5) \wedge (x^3 \vee \bar{y}^4) \wedge (x^2 \vee \bar{y}^2) \wedge (x^1)$ .

(The  $(x, y)$  pairs legal for both (b) and (c) are (1, 1), (2, 3), (3, 4), (4, 6).)

(d)  $\bigwedge_{j=\max(0, \lceil (c+1-b(d-1))/a \rceil)}^{\min(d-1, \lceil (c+1)/a \rceil)} (\bar{x}^j \vee \bar{y}^{\lceil (c+1-a_j)/b \rceil})$ , when  $a \geq b > 0$  and  $c \geq 0$ .

406. (a)  $(\bigwedge_{j=\lceil (a+1)/(d-1) \rceil}^{\lfloor \sqrt{a+1} \rfloor} (\bar{x}^j \vee \bar{y}^{\lceil (a+1)/j \rceil})) \wedge (\bigwedge_{j=\lceil (a+1)/(d-1) \rceil}^{\lfloor \sqrt{a+1} \rfloor - 1} (\bar{x}^{\lceil (a+1)/j \rceil} \vee \bar{y}^j))$ .

(b)  $(\bigwedge_{j=l+1}^{\lfloor \sqrt{a-1} \rfloor + 1} (x^j \vee y^{\lfloor (a-1)/(j-1) \rfloor + 1})) \wedge (\bigwedge_{j=l+1}^{\lfloor \sqrt{a-1} \rfloor} (x^{\lfloor (a-1)/(j-1) \rfloor + 1} \vee y^j)) \wedge (x^l) \wedge (y^l)$ , where  $l = \lfloor (a-1)/(d-1) \rfloor + 1$ . [Both formulas belong to 2SAT.]

407. (a) We always have  $\lfloor x/2 \rfloor + \lceil x/2 \rceil = x$ ,  $\lfloor x/2 \rfloor + \lfloor y/2 \rfloor \leq \frac{x+y}{2} \leq \lfloor x/2 \rfloor + \lfloor y/2 \rfloor + 1$ , and  $\lceil x/2 \rceil + \lceil y/2 \rceil - 1 \leq \frac{x+y}{2} \leq \lceil x/2 \rceil + \lceil y/2 \rceil$ . (Similar reasoning proves the correctness of Batcher's odd-even merge network; see Eq. 5.3.4-(3).)

(b) Axiom clauses like (164) needn't be introduced for  $u$  and  $v$ , or even for  $z$ ; so they aren't counted here, although they could be added if desired. Let  $a_d = d^2 - 1$  be the number of clauses in the original method; then the new method has fewer clauses when  $a_{\lfloor d/2 \rfloor} + a_{\lfloor d/2 \rfloor + 1} + 3(d-2) < a_d$ , namely when  $d \geq 7$ . (The new method for  $d = 7$  involves 45 clauses, not 48; but it introduces 10 new auxiliary variables.) Asymptotically, we can handle  $d = 2^t + 1$  with  $3t2^t + O(2^t) = 3d \lg d + O(d)$  clauses and  $d \lg d + O(d)$  auxiliary variables.

(c)  $x + y \geq z \iff (d-1-x) + (d-1-y) \leq (2d-2-z)$ ; so we can use the same method, but complemented (namely with  $x^j \mapsto \bar{x}^{d-j}$ ,  $y^j \mapsto \bar{y}^{d-j}$ ,  $z^j \mapsto \bar{z}^{2d-1-j}$ ).

at-most-one  
author  
all solutions  
2SAT  
midpoint  
Batcher  
odd-even  
merge network  
Axiom clauses  
complemented

[See N. Tamura, A. Taga, S. Kitagawa, and M. Banbara, *Constraints* **14** (2009), 254–272; R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell, *Constraints* **16** (2011), 195–221.]

408. (a) No; makespan 11 is best, achievable as follows (or via left-right reflection):

|     |    |    |    |  |
|-----|----|----|----|--|
| M1: | J1 |    | J3 |  |
| M2: | J3 | J2 | J1 |  |
| M3: | J2 | J3 | J1 |  |

Tamura  
Taga  
Kitagawa  
Banbara  
Asín  
Nieuwenhuis  
Oliveras  
Rodríguez-Carbonell  
Shmoys  
Stein  
Wein  
Tamura  
Taga  
Kitagawa  
Banbara  
symmetry breaking  
Knuth  
Guéret  
Prins

(b) If  $j$  is the last job processed by machine  $i$ , that machine must finish at time  $\leq \sum_{k=1}^n w_{ik} + \sum_{k=1}^m w_{kj} - w_{ij}$ , because  $j$  uses some other machine whenever  $i$  is idle. [See D. B. Shmoys, C. Stein, and J. Wein, *SICOMP* **23** (1994), 631.]

(c) Clearly  $0 \leq s_{ij} \leq t - w_{ij}$ . And if  $ij \neq i'j'$  but  $i = i'$  or  $j = j'$ , we must have either  $s_{ij} + w_{ij} \leq s_{i'j'}$  or  $s_{i'j'} + w_{i'j'} \leq s_{ij}$  whenever  $w_{ij}w_{i'j'} \neq 0$ .

(d) When  $w_{ij} > 0$ , introduce Boolean variables  $s_{ij}^k$  for  $1 \leq k \leq t - w_{ij}$ , with the axiom clauses  $(\bar{s}_{ij}^{k+1} \vee s_{ij}^k)$  for  $1 \leq k < t - w_{ij}$ . Then include the following clauses for all relevant  $i, j, i', j'$  as in (c): For  $0 \leq k \leq t + 1 - w_{ij} - w_{i'j'}$ , assert  $(\bar{p}_{ij i'j'} \vee \bar{s}_{ij}^k \vee s_{i'j'}^{k+w_{ij}})$  if  $ij < i'j'$  or  $(p_{i'j' ij} \vee \bar{s}_{i'j'}^k \vee s_{ij}^{k+w_{i'j'}})$  if  $ij > i'j'$ , omitting  $\bar{s}_{ij}^0$  in the first of these ternary clauses and omitting  $s_{i'j'}^{t+1-w_{i'j'}}$  in the last.

[This method, introduced by N. Tamura, A. Taga, S. Kitagawa, and M. Banbara in *Constraints* **14** (2009), 254–272, was able to solve several open shop scheduling problems in 2008 that had resisted attacks by all other approaches.]

Since the left-right reflection of any valid schedule is also valid, we can also save a factor of two by arbitrarily choosing one of the  $p$  variables and asserting  $(p_{ij i'j'})$ .

(e) Any schedule for  $W$  and  $T$  yields a schedule for  $\lfloor W/k \rfloor$  and  $\lceil T/k \rceil$ , if we examine time slots  $0, k, 2k, \dots$ . [With this observation we can narrow down the search for an optimum makespan by first working with simpler problems; the number of variables and clauses for  $\lfloor W/k \rfloor$  and  $\lceil T/k \rceil$  is about  $1/k$  times the number for  $W$  and  $T$ , and the running time also tends to obey this ratio. For example, the author solved a nontrivial  $8 \times 8$  problem by first working with  $\lfloor W/8 \rfloor$  and getting the respective results (U, S, U) for  $t = (128, 130, 129)$ , where ‘U’ means “unsatisfiable” and ‘S’ means “satisfiable”; running times were about (75, 10, 1250) megamems. Then with  $\lfloor W/4 \rfloor$  it was (S, U, U) with  $t = (262, 260, 261)$  and runtimes (425, 275, 325); with  $\lfloor W/2 \rfloor$  it was (U, S, U) with  $t = (526, 528, 527)$  and runtimes (975, 200, 900). Finally with the full  $W$  it was (U, S, S) with  $t = (1058, 1060, 1059)$  and runtimes (2050, 775, 300), establishing 1059 as the optimum makespan while doing most of the work on small subproblems.]

*Notes:* Further savings are possible by noting that any clauses learned while proving that  $t$  is satisfiable are valid also when  $t$  is decreased. Difficult random problems can be generated by using the following method suggested by C. Guéret and C. Prins in *Annals of Operations Research* **92** (1999), 165–183: Start with work times  $w_{ij}$  that are as near equal as possible, having constant row and column sums  $s$ . Then choose random rows  $i \neq i'$  and random columns  $j \neq j'$ , and transfer  $\delta$  units of weight by setting  $w_{ij} \leftarrow w_{ij} - \delta, w_{i'j} \leftarrow w_{i'j} + \delta, w_{ij'} \leftarrow w_{ij'} + \delta, w_{i'j'} \leftarrow w_{i'j'} - \delta$ , where  $\delta \geq w_{ij}$  and  $\delta \geq w_{i'j'}$ ; this operation clearly preserves the row and column sums. Choose  $\delta$  at random between  $p \cdot \min\{w_{ij}, w_{i'j'}\}$  and  $\min\{w_{ij}, w_{i'j'}\}$ , where  $p$  is a parameter. The final weights are obtained after making  $r$  such transfers. Guéret and Prins suggested choosing  $r = n^3$ , and  $p = .95$  for  $n \geq 6$ ; but other choices give useful benchmarks too.



**409.** (a) If  $S \subseteq \{1, \dots, r\}$ , let  $\Sigma_S = \sum_{j \in S} a_j$ . We can assume that job  $n$  runs on machines 1, 2, 3 in that order. So the minimum makespan is  $2w_{2n} + x$ , where  $x$  is the smallest  $\Sigma_S$  that is  $\geq \lceil (a_1 + \dots + a_r)/2 \rceil$ . (The problem of finding such an  $S$  is well known to be NP-hard [R. M. Karp, *Complexity of Computer Computations* (New York: Plenum, 1972), 97–100]; hence the open shop scheduling problem is NP-complete.)

(b) Makespan  $w_{2n} + w_{4n}$  is achievable if and only if  $\Sigma_S = (a_1 + \dots + a_r)/2$  for some  $S$ . Otherwise we can achieve makespan  $w_{2n} + w_{4n} + 1$  by running jobs 1,  $\dots$ ,  $n$  in order on machine 1 and letting  $s_{3(n-1)} = 0$ ,  $s_{4n} = w_{2n}$ ; also  $s_{2j} = w_{2n} + w_{4n}$ , if machine 1 is running job  $j$  at time  $w_{2n}$ . The other jobs are easily scheduled.

(c)  $\lfloor 3n/2 \rfloor - 2$  time slots are clearly necessary and sufficient. (If all row and column sums of  $W$  are equal to  $s$ , can the minimum makespan be  $\geq \frac{3}{2}s$ ?)

(d) The “tight” makespan  $s$  is always achievable: By renumbering the jobs we can assume that  $a_j \leq b_j$  for  $1 \leq j \leq k$ ,  $a_j \geq b_j$  for  $k < j \leq n$ ,  $b_1 = \max\{b_1, \dots, b_k\}$ ,  $a_n = \max\{a_{k+1}, \dots, a_n\}$ . Then if  $b_n \geq a_1$ , machine 1 can run jobs  $(1, \dots, n)$  in order while machine 2 runs  $(n, 1, \dots, n-1)$ ; otherwise  $(2, \dots, n, 1)$  and  $(1, \dots, n)$  suffice.

If  $a_1 + \dots + a_n \neq b_1 + \dots + b_n$ , we can increase  $a_n$  or  $b_n$  to make them equal. Then we can add a “dummy” job with  $a_{n+1} = b_{n+1} = \max\{a_1 + b_1, \dots, a_n + b_n\} - s$ , and obtain an optimum schedule in  $O(n)$  steps as explained above.

Results (a), (b), (d) are due to T. Gonzalez and S. Sahni, who introduced and named the open shop scheduling problem in *JACM* **23** (1976), 665–679. Part (c) is a subsequent observation and open problem due to Gonzalez (unpublished).

**410.** Using half adders and full adders as we did in (23) allows us to introduce intermediate variables  $w_j$  such that  $(x_2x_1x_0)_2 + (x_2x_1x_00)_2 + (x_2x_1x_000)_2 + (\bar{y}_2\bar{y}_1\bar{y}_0000)_2 \leq (w_7w_6 \dots w_0)_2$ , and then to require  $(\bar{w}_7) \wedge (\bar{w}_6)$ . In slow motion, we successively compute  $(c_0z_0)_2 \geq x_0 + x_1$ ,  $(c_1z_1)_2 \geq x_0 + x_1 + \bar{y}_0$ ,  $(c_2z_2)_2 \geq c_0 + z_1$ ,  $(c_3z_3)_2 \geq x_1 + x_2 + \bar{y}_1$ ,  $(c_4z_4)_2 \geq c_1 + c_2 + z_3$ ,  $(c_5z_5)_2 \geq x_2 + \bar{y}_2 + c_3$ ,  $(c_6z_6)_2 \geq c_4 + z_5$ ,  $(c_7z_7)_2 \geq c_5 + c_6$ ; then  $w_7w_6 \dots w_0 = c_7z_7z_6z_4z_2z_0x_1x_0$ . In slower motion, each step  $(c_i z_i)_2 \geq u + v$  expands to  $z_i \geq u \oplus v$ ,  $c_i \geq u \wedge v$ ; each step  $(c_i z_i)_2 \geq t + u + v$  expands to  $s_i \geq t \oplus u$ ,  $p_i \geq t \wedge u$ ,  $z_i \geq v \oplus s$ ,  $q_i \geq v \wedge s$ ,  $c_i \geq p_i \vee q_i$ . And at the clause level,  $t \geq u \wedge v \iff (t \vee \bar{u} \vee \bar{v})$ ;  $t \geq u \vee v \iff (t \vee \bar{u}) \wedge (t \vee \bar{v})$ ;  $t \geq u \oplus v \iff (t \vee \bar{u} \vee v) \wedge (t \vee u \vee \bar{v})$ . [Only about half of (24) is needed when inequalities replace equalities. Exercise 42 offers improvements.]

We end up with 44 binary and ternary clauses; 10 of them can be omitted, because  $z_0, z_2, z_4, z_6$ , and  $z_7$  are pure literals, and the clause for  $c_7$  can be omitted if we simply require  $c_5 = c_6 = 0$ . But the order encoding of exercise 405 is clearly much better. The log encoding becomes attractive only with larger integers, as in the following exercise. [See J. P. Warners, *Information Processing Letters* **68** (1998), 63–69.]

**411.** Use  $m + n$  new variables to represent an auxiliary number  $w = (w_{m+n} \dots w_1)_2$ . Form clauses as in exercise 41 for the product  $xy = w$ ; but retain only about half of the clauses, as in answer 410. The resulting  $9mn - 5m - 10n$  clauses are satisfiable if  $w = xy$ ; and we have  $w \geq xy$  whenever they are satisfiable. Now add  $3m + 3n - 2$  further clauses as in (169) to ensure that  $z \geq w$ . The case  $z \leq xy$  is similar.

**412.** Mixed-radix representations are also of interest in this connection. See, for example, N. Eén and N. Sörensson, *J. Satisfiability, Bool. Modeling and Comp.* **2** (2006), 1–26; T. Tanjo, N. Tamura, and M. Banbara, *LNCS* **7317** (2012), 456–462.

**413.** Eliminating first  $a_{n-1}$ , then  $a_{n-2}$ , etc., yields  $2^n - 1$  clauses. (The analogous result for  $x_1 \dots x_n < y_1 \dots y_n$  is  $2^n + 2^{n-1} + 1$ . A preprocessor will probably eliminate  $a_{n-1}$ .)

subset sum problem  
Karp  
NP-complete  
Gonzalez  
Sahni  
half adders  
full adders  
Tseytin encoding, half  
pure literals  
order encoding  
Warners  
Mixed-radix  
Eén  
Sörensson  
Tanjo  
Tamura  
Banbara  
preprocessor

414. Construct clauses for  $1 \leq k \leq n$  that represent ' $a_{k-1}$  implies  $x_k < y_k + a_k$ ':

$$\left( \bar{a}_{k-1} \vee \bigvee_{j=1}^{d-1} (\bar{x}_k^j \vee y_k^j) \right) \wedge \left( \bar{a}_{k-1} \vee a_k \vee \bigvee_{j=0}^{d-1} (\bar{x}_k^j \vee y_k^{j+1}) \right), \quad \text{omitting } \bar{x}_k^0 \text{ and } y_k^d;$$

also omit  $\bar{a}_0$ . For the relation  $x_1 \dots x_n \leq y_1 \dots y_n$  we can omit the  $d$  clauses that contain the (pure) literal  $a_n$ . But for  $x_1 \dots x_n < y_1 \dots y_n$ , we want  $a_n = 0$ ; so we omit  $a_n$  and the  $d-1$  clauses  $(\bar{a}_{n-1} \vee \bar{x}_n^j \vee y_n^j)$ .

415. There's only one, namely  $\bigwedge_{\sigma_1, \dots, \sigma_n \in \{-1, 1\}} (\sigma_1 x_1 \vee \sigma_1 y_1 \vee \dots \vee \sigma_n x_n \vee \sigma_n y_n)$ . *Proof:* Some clause must contain only positive literals, because  $f(0, \dots, 0) = 0$ . This clause must be  $(x_1 \vee y_1 \vee \dots \vee x_n \vee y_n)$ ; otherwise it would be false in cases where  $f$  is true. A similar argument shows that *every* clause  $(\sigma_1 x_1 \vee \sigma_1 y_1 \vee \dots \vee \sigma_n x_n \vee \sigma_n y_n)$  must be present. And no clause for  $f$  can contain both  $x_j$  and  $\bar{y}_j$ , or both  $\bar{x}_j$  and  $y_j$ .

416. The other clauses are  $\bigwedge_{i=1}^m ((u_i \vee \bar{v}_i \vee \bar{a}_0) \wedge (\bar{u}_i \vee v_i \vee \bar{a}_0))$  and  $(a_0 \vee a_1 \vee \dots \vee a_n)$ . [See A. Biere and R. Brummayer, *Proceedings, International Conference on Formal Methods in Computer Aided Design 8* (IEEE, 2008), 4 pages [FMCAD 08].]

417. The four clauses  $(\bar{s} \vee \bar{t} \vee u) \wedge (\bar{s} \vee t \vee v) \wedge (s \vee \bar{t} \vee \bar{u}) \wedge (s \vee t \vee \bar{v})$  ensure that  $s$  is true if and only if  $t? u: v$  is true. But we need only the first two of these, as in (174), when translating a branching program, because the other two are blocked in the initial step. Removing them makes the other two blocked on the second step, etc.

418. A suitable branching program for  $h_n$  when  $n = 3$ , beginning at  $I_{11}$ , is  $I_{11} = (\bar{1}? 21: 22)$ ,  $I_{21} = (\bar{2}? 31: 32)$ ,  $I_{22} = (\bar{2}? 32: 33)$ ,  $I_{31} = (\bar{3}? 0: 42)$ ,  $I_{32} = (\bar{3}? 42: 43)$ ,  $I_{33} = (\bar{3}? 43: 1)$ ,  $I_{42} = (\bar{1}? 0: 1)$ ,  $I_{43} = (\bar{2}? 0: 1)$ . It leads via (174) to the following clauses for row  $i$ ,  $1 \leq i \leq m$ :  $(r_{i,1,1})$ ;  $(\bar{r}_{i,k,j} \vee x_{ik} \vee r_{i,k+1,j}) \wedge (\bar{r}_{i,k,j} \vee \bar{x}_{ik} \vee r_{i,k+1,j+1})$ , for  $1 \leq j \leq k \leq n$ ;  $(\bar{r}_{i,n+1,1}) \wedge (r_{i,n+1,n+1})$  and  $(\bar{r}_{i,n+1,j+1} \vee x_{ij})$  for  $1 \leq j < n$ . Also the following clauses for column  $j$ ,  $1 \leq j \leq n$ :  $(c_{i,1,1})$ ;  $(\bar{c}_{j,k,i} \vee x_{kj} \vee c_{j,k+1,i}) \wedge (\bar{c}_{j,k,i} \vee \bar{x}_{kj} \vee c_{j,k+1,i+1})$ , for  $1 \leq i \leq k \leq m$ ;  $(\bar{c}_{j,m+1,1}) \wedge (c_{j,m+1,m+1})$  and  $(\bar{c}_{j,m+1,i+1} \vee x_{ij})$  for  $1 \leq i < m$ .

419. (a) There are exactly  $n-2$  solutions:  $x_{ij} = [j=1][i \neq m-1] + [j=2][i=m-1] + [j=k][i=m-1]$ , for  $2 < k \leq n$ .

(b) There are exactly  $m-2$  solutions:  $\bar{x}_{ij} = [j > 1][i=m-1] + [j=1][i=m-2] + [j=1][i=k]$ , for  $1 \leq k < m-2$  or  $k=m$ .

420. Start via (24) with  $(\bar{x}_1 \vee x_2 \vee s) \wedge (x_1 \vee \bar{x}_2 \vee s) \wedge (x_1 \vee x_2 \vee \bar{s}) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{s})$ ;  $(x_1 \vee \bar{c}) \wedge (x_2 \vee \bar{c}) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee c)$ ;  $(\bar{s} \vee x_3 \vee t) \wedge (s \vee \bar{x}_3 \vee t) \wedge (s \vee x_3 \vee \bar{t}) \wedge (\bar{s} \vee \bar{x}_3 \vee \bar{t})$ ;  $(s \vee \bar{c}') \wedge (x_3 \vee \bar{c}') \wedge (\bar{s} \vee \bar{x}_3 \vee c')$ ;  $(\bar{c}) \wedge (\bar{c}')$ . Propagate  $(\bar{c})$  and  $(\bar{c}')$ , obtaining  $(\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{s} \vee \bar{x}_3)$ ; remove subsumed clauses  $(\bar{x}_1 \vee \bar{x}_2 \vee \bar{s})$ ,  $(\bar{s} \vee \bar{x}_3 \vee \bar{t})$ ; remove blocked clause  $(s \vee x_3 \vee \bar{t})$ ; remove clauses containing the pure literal  $t$ ; rename  $s$  to  $a_1$ .

421. Start via (173) with  $(\bar{a}_5 \vee x_1 \vee a_4) \wedge (\bar{a}_5 \vee \bar{x}_1 \vee a_3) \wedge (\bar{a}_4 \vee \bar{x}_2 \vee a_2) \wedge (\bar{a}_3 \vee x_2 \vee a_2) \wedge (\bar{a}_3 \vee \bar{x}_2) \wedge (\bar{a}_2 \vee \bar{x}_3) \wedge (a_5)$ . Propagate  $(a_5)$ .

422. (a)  $x_1$  implies  $\bar{x}_2$ , then  $a_1$ , then  $\bar{x}_3$ ;  $x_2$  implies  $\bar{x}_1$ , then  $a_1$ , then  $\bar{x}_3$ .

(b)  $x_1$  implies  $a_3$ , then  $\bar{x}_2$ , then  $a_2$ , then  $\bar{x}_3$ ;  $x_2$  implies  $\bar{a}_3$ , then  $\bar{x}_1$ ,  $a_4$ ,  $a_2$ ,  $\bar{x}_3$ .

423. No; consider  $x_1? (x_2? x_3: x_4): (x_2? x_4: x_3)$  with  $L = (\bar{x}_3) \wedge (\bar{x}_4)$ . [But a *forcing* encoding *can* always be constructed, via the extra clauses defined in exercise 436. Notice that, in the presence of failed literal tests, weak forcing corresponds to forcing.]

424. The clause  $\bar{1}\bar{3}\bar{4}$  is redundant (in the presence of  $\bar{1}\bar{2}\bar{3}$  and  $2\bar{3}\bar{4}$ ); it cannot be omitted, because  $\{\bar{2}\bar{3}, 2\bar{3}, 12\} \not\vdash_1 \bar{3}$ . The clause  $2\bar{3}\bar{4}$  is also redundant (in the presence of  $\bar{1}\bar{3}\bar{4}$  and  $12$ ); it *can* be omitted, because  $\{\bar{1}\bar{4}, 3\bar{4}, 1\} \vdash_1 \bar{4}$ ,  $\{\bar{1}\bar{3}, 3\bar{4}, 1\} \vdash_1 \bar{3}$ , and  $\{\bar{1}\bar{2}, \bar{1}, 12\} \vdash_1 2$ .

(pure) literal  
Biere  
Brummayer  
blocked  
subsumed clauses  
blocked clause  
pure literal  
failed literal tests

**425.** If  $x$  is in the core,  $F \vdash_1 x$ , because Algorithm 7.1.1C does unit propagation. Otherwise  $F$  is satisfied when all core variables are true and all noncore variables are false.

**426.** (a) True. Suppose the clauses involving  $a_m$  are  $(a_m \vee \alpha_i)$  for  $1 \leq i \leq p$  and  $(\bar{a}_m \vee \beta_j)$  for  $1 \leq j \leq q$ ; then  $G$  contains the  $pq$  clauses  $(\alpha_i \vee \beta_j)$  instead. If  $F|L \vdash_1 l$  we want to prove that  $G|L \vdash_1 l$ . This is clear if unit propagation from  $F|L$  doesn't involve  $a_m$ . Otherwise, if  $F|L \vdash_1 a_m$ , unit propagation has falsified some  $\alpha_i$ ; every subsequent propagation step from  $F|L$  that uses  $(\bar{a}_m \vee \beta_j)$  can use  $(\alpha_i \vee \beta_j)$  in a propagation step from  $G|L$ . A similar argument applies when  $F|L \vdash_1 \bar{a}_m$ .

(Incidentally, variable elimination also preserves "honesty.")

(b) False. Let  $F = (x_1 \vee x_2 \vee a_1) \wedge (x_1 \vee x_2 \vee \bar{a}_1)$ ,  $L = \bar{x}_1$  or  $\bar{x}_2$ .

**427.** Suppose  $n = 3m$ , and let  $f$  be the symmetric function  $[\nu x < m$  or  $\nu x > 2m]$ . The prime clauses of  $f$  are the  $N = \binom{n}{m, m, m} \sim 3^{n+3/2}/(2\pi n)$  ORs of  $m$  positive literals and  $m$  negative literals. There are  $N^l = \binom{n}{m-1, m, m+1} = \frac{m}{m+1}N$  ways to specify that  $x_{i_1} = \dots = x_{i_m} = 1$  and  $x_{i_{m+1}} = \dots = x_{i_{2m-1}} = 0$ ; and this partial assignment implies that  $x_j = 1$  for  $j \notin \{i_1, \dots, i_{2m-1}\}$ . Therefore at least one of the  $m+1$  clauses  $(\bar{x}_{i_1} \vee \dots \vee \bar{x}_{i_m} \vee x_{i_{m+1}} \vee \dots \vee x_{i_{2m-1}} \vee x_j)$  must be present in any set of prime clauses that forces  $f$ . By symmetry, any such set must include at least  $N^l/m$  prime clauses.

On the other hand,  $f$  is characterized by  $O(n^2)$  forcing clauses (see answer 436).

**428.** (a)  $(y \vee z_{j1} \vee \dots \vee z_{jd})$  for  $1 \leq j \leq n$ ;  $(\bar{x}_{ij} \vee \bar{z}_{ik} \vee \bar{z}_{jk})$  for  $1 \leq i < j \leq n$ ,  $1 \leq k \leq d$ .

(b) Imagine a circuit with  $2N(N+1)$  gates  $g_{it}$ , one for each literal  $l$  of  $G_{nd}$  and for each  $0 \leq t \leq N$ , meaning that literal  $l$  is known to be true after  $t$  rounds of unit propagation, if we start with given values of the  $x_{ij}$  variables only. Thus we set  $g_{l0} \leftarrow 1$  if  $l = x_{ij}$  and  $x_{ij}$  is true, or if  $l = \bar{x}_{ij}$  and  $x_{ij}$  is false; otherwise  $g_{l0} \leftarrow 0$ . And

$$g_{l(t+1)} \leftarrow g_{lt} \vee \bigvee \{g_{\bar{l}_1 t} \wedge \dots \wedge g_{\bar{l}_k t} \mid (l \vee l_1 \vee \dots \vee l_k) \in G_{nd}\}, \quad \text{for } 1 \leq t < N.$$

Given values of the  $x_{ij}$ , the literal  $y$  is implied if and only if the graph has no  $d$ -coloring; and at most  $N$  rounds make progress. Thus there's a monotone chain for  $g_{yN} = f_{nd}$ .

[This exercise was suggested by S. Buss and R. Williams in 2014, based on a similar construction by M. Gwynne and O. Kullmann.]

**429.** Let  $\Sigma_k$  be the sum of the assigned  $x$ 's in leaves descended from node  $k$ . Unit propagation will force  $b_j^k \leftarrow 1$  for  $1 \leq j \leq \Sigma_k$ , moving from leaves toward the root. Then it will force  $b_j^k \leftarrow 0$  for  $j = \Sigma_k + 1$ , moving downwards from the root, because  $r = \Sigma_2 + \Sigma_3$  and because (21) starts this process when  $k = 2$  or  $3$ .

**430.** Imagine boundary conditions as in answer 26, and assume that  $x_{j_1}, \dots, x_{j_r}$  have been assigned 1, where  $j_1 < \dots < j_r$ . Unit propagation forces  $s_{j_k+1-k}^k \leftarrow 1$  for  $1 \leq k \leq r$ ; then it forces  $s_{j_k-k}^k \leftarrow 0$  for  $r \geq k \geq 1$ . So unassigned  $x$ 's are forced to zero.

**431.** Equivalently  $x_1 + \dots + x_m + \bar{y}_1 + \dots + \bar{y}_n \leq n$ ; so we can use (18)-(19) or (20)-(21).

**432.** The clauses of answer 404(b) can be shown to be forcing. But not those of 404(a) when  $a > 1$ ; for example, if  $a = 2$  and we assume  $\bar{x}^2$ , unit propagation doesn't yield  $y^2$ .

**433.** Yes. Imagine, for example, the partial assignment  $x = 1***10**1$ ,  $y = 10*00*1**$ . Then  $y_3$  must be 1; otherwise we'd have  $10010001 \leq x \leq y \leq 1000011111$ . In this situation unit propagation from the clauses that correspond to  $1 \leq \langle a_1 01 \rangle$ ,  $a_1 \leq \langle a_2 \bar{x}_2 0 \rangle$ ,  $a_2 \leq \langle a_3 \bar{x}_3 y_3 \rangle$ ,  $a_3 \leq \langle a_4 \bar{x}_4 0 \rangle$ ,  $a_4 \leq \langle a_5 00 \rangle$  forces  $a_1 = 1$ ,  $a_2 = 1$ ,  $a_4 = 0$ ,  $a_3 = 0$ ,  $y_3 = 1$ .

In general if a given partial assignment is consistent with  $x \leq y$ , we must have  $x \downarrow \leq y \uparrow$ , where  $x \downarrow$  and  $y \uparrow$  are obtained from  $x$  and  $y$  by changing all unassigned variables to 0 and 1, respectively. If that partial assignment forces some  $y_j$  to a

unit propagation  
honesty  
symmetric Boolean functions  
prime clauses  
Buss  
Williams  
Gwynne  
Kullmann

particular value, the value must be 1; and we must in fact have  $x \downarrow > y' \uparrow$ , where  $y'$  is like  $y$  but with  $y_j = 0$  instead of  $y_j = *$ . If  $x_j \neq 1$ , unit propagation will force  $a_1 = \dots = a_{j-1} = 1$ ,  $a_k = \dots = a_j = 0$ ,  $y_j = 1$ , for some  $k \geq j$ .

Similar remarks apply when  $x_i$  is forced, because  $x \leq y \iff \bar{y} \leq \bar{x}$ .

**434.** (a) Clearly  $p_k$  is equivalent to  $\bar{x}_1 \wedge \dots \wedge \bar{x}_k$ ,  $q_k$  is equivalent to  $\bar{x}_k \wedge \dots \wedge \bar{x}_n$ , and  $r_k$  implies that a run of exactly  $l$  1s begins at  $x_k$ .

(b) When  $l = 1$ , if  $x_k = 1$  unit propagation will imply  $\bar{p}_j$  for  $j \geq k$  and  $\bar{q}_j$  for  $j \leq k$ , hence  $\bar{r}_j$  for  $j \neq k$ ; then  $r_k$  is forced, making  $x_j = 0$  for all  $j \neq k$ . Conversely,  $x_j = 0$  forces  $\bar{r}_j$ ; if this holds for all  $j \neq k$ , then  $r_k$  is forced, making  $x_k = 1$ .

But when  $l = 2$  and  $n = 3$ , the clauses fail to force  $x_2 = 1$  by unit propagation. They also fail to force  $x_1 = 0$  when we have  $l = 2$ ,  $n = 4$ , and  $x_3 = 1$ .

**435.** The following construction with  $O(nl)$  clauses is satisfactory when  $l$  is small: Begin with the clauses for  $p_k$  and  $q_k$  (but not  $r_k$ ) in exercise 434(a); include also  $(\bar{x}_k \vee p_{k-l})$  for  $l < k \leq n$ , and  $(\bar{x}_k \vee q_{k+l})$  for  $1 \leq k \leq n-l$ . Append  $(\bar{p}_{k-l} \vee \bar{q}_{k+l} \vee x_k)$  for  $1 \leq k \leq n$ , omitting  $\bar{p}_j$  for  $j < 1$  and omitting  $\bar{q}_j$  for  $j > n$ . Finally, append

$$(x_k \vee \bar{x}_{k+1} \vee x_{k+d}) \quad \text{for } 0 \leq k < n \text{ and } 1 < d < l, \quad (*)$$

omitting  $x_j$  when  $j < 1$  or  $j > n$ .

To reduce to  $O(n \log l)$  clauses, suppose  $2^{e+1} < l \leq 2^{e+2}$ , where  $e \geq 0$ . The clauses  $(*)$  can be replaced by  $(\bar{x}_k \vee \bar{y}_k^{(e)} \vee \bar{z}_k^{(e)})$  for  $1 \leq k \leq n$ , if  $\bar{x}_{k-d}$  implies  $y_k^{(e)}$  for  $1 \leq d \leq \lfloor l/2 \rfloor$  and  $\bar{x}_{k+d}$  implies  $z_k^{(e)}$  for  $1 \leq d \leq \lceil l/2 \rceil$ . And to achieve the latter, we introduce clauses  $(\bar{y}_k^{(t)} \vee y_k^{(t+1)})$ ,  $(\bar{y}_{k-2^t}^{(t)} \vee y_k^{(t+1)})$ ,  $(\bar{z}_k^{(t)} \vee z_k^{(t+1)})$ ,  $(\bar{z}_{k+2^t}^{(t)} \vee z_k^{(t+1)})$ ,  $(x_{k-1} \vee y_k^{(0)})$ ,  $(x_{k+2^e-1-\lfloor l/2 \rfloor} \vee y_k^{(0)})$ ,  $(x_{k+1} \vee z_k^{(0)})$ ,  $(x_{k-2^e+1+\lceil l/2 \rceil} \vee z_k^{(0)})$ , for  $1 \leq k \leq n$  and  $0 \leq t < e$ , always omitting  $x_j$  or  $\bar{y}_j$  or  $\bar{z}_j$  when  $j < 1$  or  $j > n$ .

**436.** Let the variables  $q_k$  for  $0 \leq k \leq n$  and  $q \in Q$  represent the sequence of states, and let  $t_{kaq}$  represent a transition when  $1 \leq k \leq n$  and when  $T$  contains a triple of the form  $(q', a, q)$ . The clauses,  $F$ , are the following, for  $1 \leq k \leq n$ : (i)  $(\bar{t}_{kaq} \vee x_k^a) \wedge (\bar{t}_{kaq} \vee q_k)$ , where  $x_k^0$  denotes  $\bar{x}_k$  and  $x_k^1$  denotes  $x_k$ ; (ii)  $(\bar{q}_{k-1} \vee \bigvee \{t_{kaq'} \mid (q', a, q) \in T\})$ , for  $q \in Q$ ; (iii)  $(\bar{q}_k \vee \bigvee \{t_{kaq'} \mid (q', a, q) \in T\})$ ; (iv)  $(\bar{x}_k^a \vee \bigvee \{t_{kaq} \mid (q', a, q) \in T\})$ ; (v)  $(\bar{t}_{kaq'} \vee \bigvee \{q_{k-1} \mid (q, a, q') \in T\})$ ; together with (vi)  $(\bar{q}_0)$  for  $q \in Q \setminus I$  and  $(\bar{q}_n)$  for  $q \in Q \setminus O$ .

It is clear that if  $F \vdash_1 \bar{x}_k^a$ , no string  $x_1 \dots x_n \in L$  can have  $x_k = a$ . Conversely, assume that  $F \not\vdash_1 \bar{x}_k^a$ , and in particular that  $F \not\vdash_1 \epsilon$ . To prove the forcing property, we want to show that some string of  $L$  has  $x_k = a$ . It will be convenient to say that a literal  $l$  is 'n.f.' (not falsified) if  $F \not\vdash_1 \bar{l}$ ; thus  $x_k^a$  is assumed to be n.f.

By (iv), there's a  $(q', a, q) \in T$  such that  $t_{kaq}$  is n.f. Hence  $q_k$  is n.f., by (i). If  $k = n$  we have  $q \in O$  by (vi); otherwise some  $t_{(k+1)aq'}$  is n.f., by (ii), hence  $x_{k+1}^b$  is n.f. Moreover, (v) tells us that there's  $(q'', a, q) \in T$  with  $q_{k-1}''$  n.f. If  $k = 1$  we have  $q'' \in I$ ; otherwise some  $t_{(k-1)aq''}$  is n.f., by (iii), and  $x_{k-1}^c$  is n.f. Continuing this line of reasoning yields  $x_1 \dots x_n \in L$  with  $x_k = a$  (and with  $x_{k+1} = b$  if  $k < n$ ,  $x_{k-1} = c$  if  $k > 1$ ).

The same proof holds even if we add unit clauses to  $F$  that assign values to one or more of the  $x$ 's. Hence  $F$  is forcing. [See F. Bacchus, *LNCS 4741* (2007), 133–147.]

The language  $L_2$  of exercise 434 yields  $17n + 4$  clauses:  $F = \bigwedge_{1 \leq k \leq n} ((\bar{t}_{k00} \vee \bar{x}_k) \wedge (\bar{t}_{k00} \vee 0_k) \wedge (\bar{t}_{k11} \vee x_k) \wedge (\bar{t}_{k11} \vee 1_k) \wedge (\bar{t}_{k12} \vee x_k) \wedge (\bar{t}_{k12} \vee 2_k) \wedge (\bar{t}_{k02} \vee \bar{x}_k) \wedge (\bar{t}_{k02} \vee 2_k) \wedge (\bar{0}_{k-1} \vee t_{k00} \vee t_{k11}) \wedge (\bar{1}_{k-1} \vee t_{k12}) \wedge (\bar{2}_{k-1} \vee t_{k02}) \wedge (x_k \vee t_{k00} \vee t_{k02}) \wedge (\bar{x}_k \vee t_{k11} \vee t_{k12}) \wedge (\bar{t}_{k00} \vee 0_{k-1}) \wedge (\bar{t}_{k11} \vee 0_{k-1}) \wedge (\bar{t}_{k12} \vee 1_{k-1}) \wedge (\bar{t}_{k02} \vee 2_{k-1})) \wedge (\bar{1}_0) \wedge (\bar{2}_0) \wedge (\bar{0}_n) \wedge (\bar{1}_n)$ . (Unit propagation will immediately assign values to 10 of the  $8n + 3$  variables, thereby satisfying 22 of these clauses, when  $n \geq 3$ . For example,  $\bar{t}_{112}$ ,  $\bar{t}_{n11}$ ,  $\bar{0}_{n-1}$  are forced.)

The clauses produced by this general-purpose construction can often be significantly simplified by preprocessing to eliminate auxiliary variables. (See exercise 426.)

preprocessing  
eliminate  
automaton  
unit propagation  
Quimper  
Walsh  
Bailleux  
Boufkhad  
Roussel

**437.** Each variable  $x_k$  now becomes a set of  $|A|$  variables  $x_{ka}$  for  $a \in A$ , with clauses like (15) and (17) to ensure that exactly one value is assigned. The same construction is then valid, with the same proof, if we simply replace ' $x_k^a$ ' by ' $x_{ka}$ ' throughout. (Notice that unit propagation will often derive partial information such as  $\bar{x}_{ka}$ , meaning that  $x_k \neq a$ , although the precise value of  $x_k$  may not be known.)

**438.** Let  $l_{\leq j} = l_1 + \dots + l_j$ . Exercise 436 does the job via the following automaton:  $Q = \{0, 1, \dots, l_{\leq t} + t - 1\}$ ,  $I = \{0\}$ ,  $O = \{l_{\leq t} + t - 1\}$ ;  $T = \{(l_{\leq j} + j, 0, l_{\leq j} + j) \mid 0 \leq j < t\} \cup \{(l_{\leq j} + j + k, 1, l_{\leq j} + j + k + 1) \mid 0 \leq j < t, 0 \leq k < l_{j+1}\} \cup \{(l_{\leq j} + j - 1, 0, l_{\leq j} + j - [j=t]) \mid 1 \leq j \leq t\}$ .

**439.** We obviously want the clauses  $(\bar{x}_j \vee \bar{x}_{j+1})$  for  $1 \leq j < n$ ; and we can use, say, (18) and (19) with  $r = t$ , to force 0s whenever the number of 1s reaches  $t$ . The difficult part is to force 1s from partial patterns of 0s; for example, if  $n = 9$  and  $t = 4$ , we can conclude that  $x_4 = x_6 = 1$  as soon as we know that  $x_3 = x_7 = 0$ .

An interesting modification of (18) and (19) turns out to work beautifully, namely with the clauses  $(\bar{t}_j^k \vee t_{j+1}^k)$  for  $1 \leq j < 2t - 1$  and  $1 \leq k \leq n - 2t + 1$ , together with  $(x_{2j+k-1} \vee \bar{t}_{2j-1}^k \vee t_{2j-1}^{k+1})$  for  $1 \leq j \leq t$  and  $0 \leq k \leq n - 2t + 1$ , omitting  $\bar{t}_{2j-1}^0$  and  $t_{2j-1}^{n-2t+2}$ .

**440.** It's convenient to introduce  $\binom{n+1}{2}|N|$  variables  $P_{ik}$  for all  $P \in N$  and for  $1 \leq i \leq k \leq n$ , as well as  $\binom{n+1}{3}|N|^2$  variables  $QR_{ijk}$  for  $Q, R \in N$  and for  $1 \leq i < j \leq k \leq n$ , although almost all of them will be eliminated by unit propagation. The clauses are: (i)  $(\bar{Q}R_{ijk} \vee Q_{i(j-1)}) \wedge (\bar{Q}R_{ijk} \vee R_{jk})$ ; (ii)  $(\bar{P}_{kk} \vee \bigvee \{x_k^a \mid P \rightarrow a \in U\})$ ; (iii)  $(\bar{P}_{ik} \vee \bigvee \{QR_{ijk} \mid i < j \leq k, P \rightarrow QR \in W\})$ , if  $i < k$ ; (iv)  $(\bar{x}_k^a \vee \bigvee \{P_{kk} \mid P \rightarrow a \in U\})$ ; (v)  $(\bar{P}_{ik} \vee \bigvee \{PR_{i(k+1)l} \mid k < l \leq n, R \in N\} \vee \bigvee \{QR_{hik} \mid 1 \leq h < i, Q \in N\})$ , if  $i > 1$  or  $k < n$ ; (vi)  $(\bar{Q}R_{ijk} \vee \bigvee \{P_{ik} \mid P \rightarrow QR \in W\})$ ; (vii)  $(\bar{P}_{1n})$  for  $P \in N \setminus S$ .

The forcing property is proved by extending the argument in answer 436: Assume that  $x_k^a$  is n.f.; then some  $P_{kk}$  with  $P \rightarrow a$  is also n.f. Whenever  $P_{ik}$  is n.f. with  $i > 1$  or  $k < n$ , some  $PR_{i(k+1)l}$  or  $QR_{hik}$  is n.f.; hence some "larger"  $P'_{il}$  or  $P'_{hk}$  is also n.f. And if  $P_{1n}$  is n.f., we have  $P \in S$ .

Furthermore we can go "downward": Whenever  $P_{ik}$  is n.f. with  $i < k$ , there's  $QR_{ijk}$  such that  $Q_{i(j-1)}$  and  $R_{jk}$  are n.f.; on the other hand if  $P_{kk}$  is n.f., there's  $a \in A$  such that  $x_k^a$  is n.f. Our assumption that  $x_k^a$  is n.f. has therefore shown the existence of  $x_1 \dots x_n \in L$  with  $x_k = a$ .

[See C.-G. Quimper and T. Walsh, *LNCS 4741* (2007), 590–604].

**441.** See O. Bailleux, Y. Boufkhad, and O. Roussel, *LNCS 5584* (2009), 181–194.

**442.** (a)  $F \mid L_q^- = F \mid l_1 \dots \mid l_{q-1} \mid \bar{l}_q$  contains  $\epsilon$  if and only if  $F \mid l_1 \dots \mid l_{q-1}$  contains  $\epsilon$  or the unit clause  $(l_q)$ .

(b) If  $F \not\vdash_1 l$  and  $F \mid \bar{l} \vdash_1 \epsilon$ , the failed literal elimination technique will reduce  $F$  to  $F \mid l$  and continue looking for further reductions. Thus we have  $F \vdash_2 l$  if and only if unit propagation plus failed literal elimination will deduce either  $\epsilon$  or  $l$ .

(c) Use induction on  $k$ ; both statements are obvious when  $k = 0$ . Suppose we have  $F \vdash_{k+1} \bar{l}$  via  $l_1, \dots, l_p = \bar{l}$ , with  $F \mid L_q^- \vdash_k \epsilon$  for  $1 \leq q \leq p$ . If  $p > 1$  we have  $F \mid l \mid L_q^- \vdash_k \epsilon$  for  $1 \leq q < p$ ; it follows that  $F \mid l \vdash_{k+1} l_{p-1}$  and  $F \mid l \vdash_{k+1} \bar{l}_{p-1}$ . If  $p = 1$  we have  $F \mid l \vdash_k \epsilon$ . Hence  $F \mid l \vdash_{k+1} \epsilon$  in both cases.

Now we want to prove that  $F \mid l \vdash_{k+1} \epsilon$  and  $F \vdash_{k+2} \epsilon$ , given  $F \vdash_{k+1} l'$  and  $F \vdash_{k+1} \bar{l}'$ . If  $F \mid L_q^- \vdash_k \epsilon$  for  $1 \leq q \leq p$ , with  $l_p = l'$ , we know that  $F \mid L_q^- \vdash_{k+1} \epsilon$ . Furthermore we can assume that  $F \not\vdash_{k+1} \bar{l}$ ; hence  $l \neq \bar{l}_q$  for  $1 \leq q \leq p$ , and  $l \neq l_p$ . If

$l = l_q$  for some  $q < p$ , then  $F | l | L_r^- \vdash_k \epsilon$  for  $1 \leq r < q$  and  $F | L_r^- \vdash_k \epsilon$  for  $q < r \leq p$ ; otherwise  $F | l | L_q^- \vdash_k \epsilon$  for  $1 \leq q \leq p$ . In both cases  $F | l \vdash_{k+1} l'$  and  $F \vdash_{k+2} l'$ . Essentially the same proof shows that  $F | l \vdash_{k+1} \bar{l}'$  and  $F \vdash_{k+2} \bar{l}'$ .

(d) True, by the last relation in part (c).

(e) If all clauses of  $F$  have more than  $k$  literals,  $L_k(F)$  is empty; hence  $L_0(R') = L_1(R') = L_2(R') = \emptyset$ . But  $L_k(R') = \{\bar{1}, 2, 4\}$  for  $k \geq 3$ ; for example,  $R' \vdash_3 \bar{1}$  because  $R' | 1 \vdash_2 \epsilon$ , because  $R' | 1 \vdash_2 3$  and  $R' | 1 \vdash_2 \bar{3}$ .

(f) Unit propagation can be done in  $O(N)$  steps if  $N$  is the total length of all clauses; this handles the case  $k = 1$ .

For  $k \geq 2$ , procedure  $P_k(F)$  calls  $P_{k-1}(F | x_1)$ ,  $P_{k-1}(F | \bar{x}_1)$ ,  $P_{k-1}(F | x_2)$ , etc., until either finding  $P_{k-1}(F | \bar{l}) = \{\epsilon\}$  or trying both literals for each variable of  $F$ . In the latter case,  $P_k$  returns  $F$ . In the former case, if  $P_{k-1}(F | l)$  is also  $\{\epsilon\}$ ,  $P_k$  returns  $\{\epsilon\}$ ; otherwise it returns  $P_k(F | l)$ . The set  $L_k$  contains all literals for which we've reduced  $F$  to  $F | l$ , unless  $P_k(F) = \{\epsilon\}$ . (In the latter case, every literal is in  $L_k$ .)

To justify this procedure we must verify that the order of testing literals doesn't matter. If  $F | \bar{l} \vdash_k \epsilon$  and  $F | \bar{l}' \vdash_k \epsilon$ , we have  $F | l | \bar{l}' \vdash_k \epsilon$  and  $F | l' | \bar{l} \vdash_k \epsilon$  by (c); hence  $P_k(F | l) = P_k(F | l | l') = P_k(F | l' | l) = P_k(F | l')$ .

[See O. Kullmann, *Annals of Math. and Artificial Intell.* **40** (2004), 303–352.]

**443.** (a) If  $F | L \vdash \epsilon$  then  $F | L \vdash l$  for all literals  $l$ ; so if  $F \in \text{PC}_k$  we have  $F | L \vdash_k l$  and  $F | L \vdash_k \bar{l}$  and  $F | L \vdash_k \epsilon$ , proving that  $\text{PC}_k \subseteq \text{UC}_k$ .

Suppose  $F \in \text{UC}_k$  and  $F | L \vdash l$ . Then  $F | L | \bar{l} \vdash \epsilon$ , and we have  $F | L | \bar{l} \vdash_k \epsilon$ . Consequently  $F | L \vdash_{k+1} l$ , proving that  $\text{UC}_k \subseteq \text{PC}_{k+1}$ .

The satisfiable clause sets  $\emptyset$ ,  $\{1\}$ ,  $\{1, \bar{1}2\}$ ,  $\{12, \bar{1}2\}$ ,  $\{12, \bar{1}2, \bar{1}2\bar{3}\}$ ,  $\{123, \bar{1}23, \bar{1}2\bar{3}\}$ ,  $\{123, \bar{1}23, \bar{1}2\bar{3}, \bar{1}2\bar{3}\bar{4}\}$ ,  $\{123, \bar{1}23, \bar{1}2\bar{3}, \bar{1}2\bar{3}\bar{4}, \bar{1}2\bar{3}\bar{4}\bar{5}\}$ ,  $\dots$ , show that  $\text{PC}_k \neq \text{UC}_k \neq \text{PC}_{k+1}$ .

(b)  $F \in \text{PC}_0$  if and only if  $F = \emptyset$  or  $\epsilon \in F$ . (This can be proved by induction on the number of variables in  $F$ , because  $\epsilon \notin F$  implies that  $F$  has no unit clauses.)

(c) If  $F$  has only one clause, it is in  $\text{UC}_0$ . More interesting examples are  $\{\bar{1}2, \bar{1}2\}$ ;  $\{1234, \bar{1}2\bar{3}\bar{4}\}$ ;  $\{123\bar{4}, 12\bar{3}4, 1234, \bar{1}234\}$ ;  $\{12, \bar{1}2, 34\bar{5}, \bar{3}4\bar{5}\}$ ; etc. In general,  $F$  is in  $\text{UC}_0$  if and only if it contains all of its prime clauses.

(d) True, by induction on  $n$ : If  $F | L \vdash l$  then  $F | L | \bar{l} \vdash \epsilon$ , and  $F | L | \bar{l}$  has  $\leq n - 1$  variables; so  $F | L | \bar{l} \in \text{PC}_{n-1} \subseteq \text{UC}_{n-1}$ . Hence we have  $F | L | \bar{l} \vdash_{n-1} \epsilon$  and  $F | L \vdash_n l$ .

(e) False, by the examples in (c).

(f)  $R' \in \text{UC}_2 \setminus \text{PC}_2$ . For example, we have  $R' | 1 \vdash_2 2$  and  $R' | 1 \vdash_2 \bar{2}$ .

[See M. Gwynne and O. Kullmann, *arXiv:1406.7398* [cs.CC] (2014), 67 pages.]

**444.** (a) Complementing a variable doesn't affect the algorithm's behavior, so we can assume that  $F$  consists of unrenamed Horn clauses. Then all clauses of  $F$  will be Horn clauses of length  $\geq 2$  whenever step E2 is reached. Such clauses are always satisfiable, by setting all remaining variables false; so step E3 cannot find both  $F \vdash_1 l$  and  $F \vdash_1 \bar{l}$ .

(b) For example,  $\{12, \bar{2}3, \bar{1}2\bar{3}, \bar{1}23\}$ .

(c) Every unsatisfiable  $F$  recognized by SLUR must be in  $\text{UC}_1$ . Conversely, if  $F \in \text{UC}_1$ , we can prove that  $F$  is satisfiable and in  $\text{UC}_1$  whenever step E2 is reached.

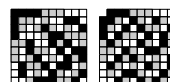
[Essentially the same argument proves that a generalized algorithm, which uses  $\vdash_k$  instead of  $\vdash_1$  in steps E1 and E3, always classifies  $F$  if and only if  $F \in \text{UC}_k$ . See M. Gwynne and O. Kullmann, *Journal of Automated Reasoning* **52** (2014), 31–65.]

(d) If step E3 interleaves unit propagation on  $F | l$  with unit propagation on  $F | \bar{l}$ , stopping when either branch is complete and  $\epsilon$  was not detected in the other, the running time is proportional to the number of cells used to store  $F$ , using data structures like those of Algorithm L. (This is an unpublished idea of Klaus Truemper.)

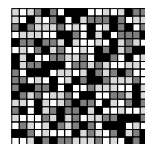
Kullmann  
prime clauses  
Gwynne  
Kullmann  
propagation,  $k$ th order  
 $\text{UC}_k$   
Gwynne  
Kullmann  
data structures  
Truemper



**451.** One of the colors can be placed uniquely, by the previous exercise. So we're left with the simple problem of two-coloring the remaining 66 squares and avoiding both 0-quads and 1-quads. That problem is unsatisfiable with  $\sum x_{ij}$  odd. The author then constructed a  $33 + 33 + 33$  solution by hand, using the fact that each color class must be unable to use the deleted square. [See M. Beresin, E. Levine, and J. Winn, *The College Mathematics Journal* **20** (1989), 106–114 and the cover; J. L. Lewis, *J. Recreational Math.* **28** (1997), 266–273.]



**452.** Any such solution must have exactly 81 cells of each color, because R. Nowakowski proved in 1978 that  $Z(18, 18) = 82$ . The solution exhibited here was found by B. Steinbach and C. Posthoff [*Multiple-Valued Logic and Soft Computing* **21** (2013), 609–625], exploiting  $90^\circ$  rotational symmetry.



**453.** (a) If  $R \subseteq \{1, \dots, m\}$  and  $C \subseteq \{1, \dots, n\}$ , let  $V(R, C) = \{u_i \mid i \in R\} \cup \{v_j \mid j \in C\}$ . If  $X$  is decomposable, there's no path from a vertex in  $V(R, C)$  to a vertex not in  $V(R, C)$ ; hence the graph isn't connected. Conversely, if the graph isn't connected, let  $V(R, C)$  be one of its connected components. Then  $0 < |R| + |C| < m + n$ , and we've decomposed  $X$ .

(b) False in general, unless every row and column of  $X'$  contains a positive element. Otherwise, clearly true by the definition of lexicographic order.

(c) True: A direct sum is certainly decomposable. Conversely, let  $X$  be decomposable via  $R$  and  $C$ . We may assume that  $1 \in R$  or  $1 \in C$ ; otherwise we could replace  $R$  by  $\{1, \dots, m\} \setminus R$  and  $C$  by  $\{1, \dots, n\} \setminus C$ . Let  $i \geq 1$  and  $j \geq 1$  be minimal such that  $i \notin R$  and  $j \notin C$ . Then  $x_{i'j} = 0$  for  $1 \leq i' < i$  and  $x_{ij'} = 0$  for  $1 \leq j' < j$ . The lexicographic constraints now force  $x_{i'j'} = 0$  for  $1 \leq i' < i, j' \geq j$ ; also for  $i' \geq i, 1 \leq j' < j$ . Consequently  $X = X' \oplus X''$ , where  $X'$  is  $(i-1) \times (j-1)$  and  $X''$  is  $(m+1-i) \times (n+1-j)$ . (Degenerate cases where  $i = 1$  or  $j = 1$  or  $i = m+1$  or  $j = n+1$  need to be considered, but they work fine. This result allows us to “read off” the block decomposition of a lexicographically ordered matrix.)

*Reference:* A. Mader and O. Mutzbauer, *Ars Combinatoria* **61** (2001), 81–95.

**454.** We have  $f(x) \leq f(x\tau) \leq f(x\tau\tau) \leq \dots \leq f(x\tau^k) \leq \dots$ ; eventually  $x\tau^k = x$ .

**455.** (a) Yes, because  $C$  only causes 1001 and 1101 to be nonsolutions. (b) No, because  $F$  might have been satisfied only by 0011. (c) Yes as in (a), although (187) might no longer be an endomorphism of  $F \wedge C$  as it was in that case. (d) Yes; if 0110 is a solution, so are 0101 and 1010. [Of course this exercise is highly artificial: We're unlikely to know that a weird mapping such as (187) is an endomorphism of  $F$  unless we know a lot more about the set of solutions.]

**456.** Only  $(1 + 2 \cdot 7)(1 + 2)(1 + 8) = 405$ , out of 65536 possibilities (about 0.06%).

**457.** We have  $\min_{0 \leq k \leq 16} (k^k 16^{16-k}) = 6^6 16^{10} \approx 51.3 \times 10^{16}$ . For general  $n$ , the minimum occurs when  $k = 2^n/e + O(1)$ ; and it is  $2^{2^n(n-x)}$  where  $x = 1/(e \ln 2) + O(2^{-n}) < 1$ .

**458.** The operation of assigning values to each variable of an autarky, so that all clauses containing those variables are satisfied, while leaving all other variables unchanged, is an endomorphism. (For example, consider the operation that makes a pure literal true.)

**459.**  $\text{sweep}(X_{ij}) = 0$  when  $i = 0$  or  $j = 0$ . And for  $1 \leq i \leq m$  and  $1 \leq j \leq n$  we have  $\text{sweep}(X_{ij}) = \max(x_{ij} + \text{sweep}(X_{(i-1)(j-1)}), \text{sweep}(X_{(i-1)j}), \text{sweep}(X_{i(j-1)}))$ .

[Let the 1s in the matrix be  $x_{i_1 j_1}, \dots, x_{i_r j_r}$ , with  $1 \leq i_1 \leq \dots \leq i_r \leq m$  and with  $j_{q+1} < j_q$  when  $i_{q+1} = i_q$ . Richard Stanley has observed (unpublished) that  $\text{sweep}(X)$  is the number of rows that occur when the Robinson–Schensted–Knuth algorithm is used to insert the sequence  $n - j_1, \dots, n - j_r$  into an initially empty tableau.]

Knuth  
Beresin  
Levine  
Winn  
Lewis  
Nowakowski  
Steinbach  
Posthoff  
rotational symmetry  
block decomposition  
Mader  
Mutzbauer  
pure literal  
Stanley  
Robinson  
Schensted  
Knuth  
tableau



**460.** We introduce auxiliary variables  $s_{ij}^t$  that will become true if  $\text{sweep}(X_{ij}) > t$ . They are implicitly true when  $t < 0$ , false when  $t = k$ . The clauses are as follows, for  $1 \leq i \leq m$ ,  $1 \leq j \leq n$ , and  $0 \leq t \leq \min(i - 1, j - 1, k)$ :  $(\bar{s}_{(i-1)j}^t \vee s_{ij}^t)$ , if  $i > 1$  and  $t < k$ ;  $(\bar{s}_{i(j-1)}^t \vee s_{ij}^t)$ , if  $j > 1$  and  $t < k$ ; and  $(\bar{x}_{ij} \vee \bar{s}_{(i-1)(j-1)}^{t-1} \vee s_{ij}^t)$ . Omit  $\bar{s}_{0(j-1)}^{t-1}$  and  $\bar{s}_{(i-1)0}^{t-1}$  and  $\bar{s}_{(i-1)(j-1)}^0$  and  $\bar{s}_{ij}^k$  from that last clause, if present.

Durfee square  
disjoint shortest paths  
unit propagation

**461.**  $\bigvee_{i=1}^{m-1} \bigvee_{j=1}^{n-1} (x_{ij} \vee \bar{c}_{(i-1)j} \vee c_{ij}) \wedge \bigvee_{i=1}^m \bigvee_{j=1}^{n-1} (\bar{c}_{(i-1)j} \vee \bar{x}_{ij} \vee x_{i(j+1)})$ , omitting  $\bar{c}_{0j}$ . These clauses take care of  $\tau_1$ ; interchange  $i \leftrightarrow j$ ,  $m \leftrightarrow n$  for  $\tau_2$ .

**462.** Let  $\tilde{X}_{ij}$  denote the last  $m + 1 - i$  rows and the last  $n + 1 - j$  columns of  $X$ ; and let  $t_{ij} = \text{sweep}(X_{(i-1)(j-1)}) + \text{sweep}(\tilde{X}_{(i+1)(j+1)})$ . For  $\tau_1$  we must prove  $1 + t_{i(j+1)} \leq k$ , given that  $1 + t_{ij} \leq k$ . It's true because  $\text{sweep}(X_{(i-1)j}) = \text{sweep}(X_{(i-1)(j-1)})$  when column  $j$  begins with  $i - 1$  zeros, and we have  $\text{sweep}(\tilde{X}_{(i+1)(j+2)}) \leq \text{sweep}(\tilde{X}_{(i+1)(j+1)})$ .

Let  $X' = X\tau_3$  have the associated sweep sums  $t'_{ij}$ . We must prove that  $t'_{ij} \leq k$  and  $1 + t'_{(i+1)(j+1)} \leq k$ , if  $1 + t_{ij} \leq k$ ,  $1 + t_{i(j+1)} \leq k$ ,  $1 + t_{(i+1)j} \leq k$ , and  $t_{(i+1)(j+1)} \leq k$ . The key point is that  $\text{sweep}(X'_{ij}) = \max(\text{sweep}(X_{(i-1)j}), \text{sweep}(X_{i(j-1)}))$ , since  $x'_{ij} = 0$ . Also  $\text{sweep}(\tilde{X}'_{(i+1)(j+1)}) = 1 + \text{sweep}(\tilde{X}_{(i+2)(j+1)})$ .

(Notice that  $\tau_1$  and  $\tau_2$  might actually *decrease* the sweep, but  $\tau_3$  preserves it.)

**463.** If row  $i + 1$  is entirely zero but row  $i$  isn't,  $\tau_2$  will apply. Therefore the all-zero rows occur at the top. And by  $\tau_1$ , the first nonzero row has all its 1s at the right.

Suppose rows 1 through  $i$  have  $r_1, \dots, r_i$  1s, all at the right, with  $r_i > 0$ . Then  $r_1 \leq \dots \leq r_i$ , by  $\tau_2$ . If  $i < n$  we can increase  $i$  to  $i + 1$ , since we can't have  $x_{(i+1)j} > x_{(i+1)(j+1)}$  when  $j \leq n - r_i$ , by  $\tau_1$ ; and we can't have it when  $j > n - r_i$ , by  $\tau_3$ .

Thus all the 1s are clustered at the right and the bottom, like the diagram of a partition but rotated  $180^\circ$ ; and the sweep is the size of its "Durfee square" (see Fig. 48 in Section 7.2.1.4). Hence the maximum number of 1s, given sweep  $k$ , is  $k(m + n - k)$ .

**464.** By answer 462,  $\tau_1$  can be strengthened to  $\tau'_1$ , which sets  $x_{i(j+1)} \leftarrow 1$  but leaves  $x_{ij} = 1$ . Similarly,  $\tau_2$  can be strengthened to  $\tau'_2$ . These endomorphisms preserve the sweep but increase the weight, so they can't apply to a matrix of maximum weight. [One can prove, in fact, that max-weight binary matrices of sweep  $k$  are precisely equivalent to  $k$  disjoint shortest paths from the leftmost cells in row  $m$  to the rightmost cells in row 1. Hence every integer matrix of sweep  $k$  is the sum of  $k$  matrices of sweep 1.]

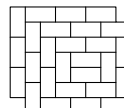
**465.** If not, there's a cycle  $x_0 \rightarrow x_1 \rightarrow \dots \rightarrow x_p = x_0$  of length  $p > 1$ , where  $x_i \tau_{uv_i} \mapsto x_{i+1}$ . Let  $uv$  be the largest of  $\{uv_1, \dots, uv_{p-1}\}$ . Then none of the other  $\tau$ 's in the cycle can change the status of edge  $uv$ . But that edge must change status at least twice.

**466.** Notice first that  $v_{11}$  must be true, if  $m \geq 2$ . Otherwise  $h_{11}, v_{21}, h_{22}, v_{32}, \dots$  would successively be forced by unit propagation, until reaching a contradiction at the edge of the board. And  $v_{31}$  must also be true, if  $m \geq 4$ , by a similar argument. Thus the entire first column must be filled with verticals, except the bottom row when  $m$  is odd.

Then we can show that the remainder of row 1 is filled with horizontals, except for the rightmost column when  $n$  is even. And so on.

The unique solution when  $m$  and  $n$  are both even uses  $v_{ij}$  if and only if  $i + j$  is even and  $\max(i, m - i) \leq j \leq n/2$ , or  $i + j$  is odd and  $v_{i(n+1-j)}$  is used. When  $m$  is odd, add a row of horizontals below the  $(m - 1) \times n$  solution. When  $n$  is odd, remove the rightmost column of verticals in the  $m \times (n + 1)$  solution.

**467.** The  $8 \times 7$  covering is obtained by reflection of the  $7 \times 8$  covering (shown here) about its southwest-to-northeast diagonal. Both solutions are unique.



**468.** (a) Typical running times with Algorithm C for sizes  $6 \times 6$ ,  $8 \times 8$ ,  $\dots$ ,  $16 \times 16$  are somewhat improved: 39 K $\mu$ , 368 K $\mu$ , 4.3 M $\mu$ , 48 M $\mu$ , 626 M $\mu$ , 8 G $\mu$ .

(b) Now they're even better, but still growing exponentially: 30 K $\mu$ , 289 K $\mu$ , 2.3 M $\mu$ , 22 M $\mu$ , 276 M $\mu$ , 1.7 G $\mu$ .

even-length cycle  
truth table  
orbits  
cycle

**469.** For instance  $(v_{11}), (v_{31}), (v_{51}), (h_{12}), (h_{14}), (v_{22}), (v_{42}), (h_{23}), (v_{33}), \epsilon$ .

**470.** There can't be a cycle  $x_0 \rightarrow x_1 \rightarrow \dots \rightarrow x_p = x_0$  of length  $p > 1$ , because the largest vertex whose mate is changed always gets smaller and smaller mates.

**471.** We must pair  $2n$  with 1, then  $2n - 1$  with 2,  $\dots$ , then  $n + 1$  with  $n$ .

**472.** We can number the vertices from 1 to  $mn$  in such a way that every 4-cycle switches as desired. For example, we can make  $(i, j) < (i, j + 1) \iff (i, j) < (i + 1, j) \iff (i, j) \bmod 4 \in \{(0, 0), (0, 1), (1, 1), (1, 2), (2, 2), (2, 3), (3, 3), (3, 0)\}$ . One such numbering in the  $4 \times 4$  case is shown here.

|    |    |    |    |
|----|----|----|----|
| 16 | 15 | 1  | 2  |
| 4  | 14 | 13 | 3  |
| 5  | 6  | 12 | 11 |
| 9  | 7  | 8  | 10 |

**473.** For every even-length cycle  $v_0 \text{---} v_1 \text{---} \dots \text{---} v_{2r-1} \text{---} v_0$  with  $v_0 = \max v_i$  and  $v_1 > v_{2r-1}$ , assert  $(\bar{v}_0 \bar{v}_1 \vee v_1 v_2 \vee \bar{v}_2 \bar{v}_3 \vee \dots \vee v_{2r-1} v_0)$ .

**474.** (a)  $(2n) \cdot (2n - 2) \cdot \dots \cdot 2 = 2^n n!$ . (b)  $(17\bar{3})(\bar{1}\bar{7}3)(25\bar{2}\bar{5})(4\bar{4})(6)(\bar{6})$ .

(c) Using 0, 1,  $\dots$ , f for the 4-tuples 0000, 0001,  $\dots$ , 1111, we must have  $f(0) = f(9) = f(5)$ ,  $f(2) = f(b) = f(7)$ ;  $f(4) = f(8) = f(d)$ ; and  $f(6) = f(a) = f(f)$ ; in other words, the truth table of  $f$  must have the form  $abcdeagcagcfchg$ , where  $a, b, c, d, e, f, g, h \in \{0, 1\}$ . So there are  $2^8$   $f$ 's.

(d) Change '=' to ' $\neq$ ' in (c). There are no such truth tables, because (191) contains odd cycles; all cycles of an antisymmetry must have even length.

(e) The 128 binary 7-tuples are partitioned into sixteen "orbits"  $\{x, x\sigma, x\sigma^2, \dots\}$ , with eight of size 12 and eight of size 4. For example, one of the 4s is  $\{0011010, 0010110, 0111110, 0110010\}$ ; one of the 12s is  $\{0000000, 0011101, \dots, 1111000\}$ . Hence there are  $2^{16}$  functions with this symmetry, and  $2^{16}$  others with this antisymmetry.

**475.** (a)  $2^{n+1} n!$ . (There are  $2^{n+1} n! / a$ , if  $f$  has  $a$  automorphisms+antiautomorphisms.)

(b)  $(x\bar{z})(\bar{x}z)$ , because (surprisingly)  $(x \vee y) \wedge (x \oplus z) = (\bar{z} \vee y) \wedge (\bar{z} \oplus \bar{x})$ .

(c) In general if  $\sigma$  is any permutation having a cycle of length  $l$ , and if  $p$  is a prime divisor of  $l$ , some power of  $\sigma$  will have a cycle of length  $p$ . (Repeatedly raise  $\sigma$  to the  $q$ th power for all primes  $q \neq p$ , until all cycle lengths are powers of  $p$ . Then, if the longest remaining cycle has length  $p^e$ , compute the  $p^{e-1}$ st power.)

(d) Suppose  $f(x_1, x_2, x_3)$  has the symmetry  $(x_1 \bar{x}_2 x_3)(\bar{x}_1 x_2 \bar{x}_3)$ . Then  $f(0, 0, 0) = f(1, 1, 0) = f(0, 1, 1)$ ,  $f(1, 1, 1) = f(0, 0, 1) = f(1, 0, 0)$ , so  $(x_1 \bar{x}_2)(\bar{x}_1 x_2)$  is a symmetry.

(e) A similar argument shows that  $(ux)(vw)(\bar{u}\bar{x})(\bar{v}\bar{w})$  is a symmetry.

(f) If  $\sigma$  is an antisymmetry of  $f$ , then  $\sigma^2$  is a symmetry. If  $f$  has a nontrivial symmetry, it has a symmetry of prime order  $p$ , by (c). And if  $p \neq 2$ , it has one of order 2, by (d) and (e), unless  $n > 5$ .

(g) Let  $f(x_1, \dots, x_6) = 1$  only when  $x_1 \dots x_6 \in \{001000, 001001, 001011, 010000, 010010, 010110, 100000, 100100, 100101\}$ . (Another interesting example, for  $n = 7$ , has  $f = 1 \iff x_1 \dots x_7$  is a cyclic shift of 0000001, 0001101, or 0011101; 21 symmetries.)

**476.** We want clauses that specify  $r$ -step chains in  $n$  variables, having a single output  $x_{n+r}$ . For  $0 < t < t' < 2^n$ , introduce new variables  $\Delta_{tt'} = x_{(n+r)t} \oplus x_{(n+r)t'}$ . (See (24).) Then for each signed involution  $\sigma$ , not the identity, we want a clause that says " $\sigma$  is not a symmetry of  $f$ ," namely  $(\bigvee \{\Delta_{tt'} \mid t < t' \text{ and } t' = t\sigma\})$ . (Here  $t$  is considered to be the same as its binary representation  $(t_1 \dots t_n)_2$ , as in exercise 477.)

Also, if  $\sigma$  has no fixed points—this is true if and only if  $\sigma$  takes  $x_i \mapsto \bar{x}_i$  for at least one  $i$ —we have further things to do: In case (b), we want a clause that says " $\sigma$

is not an antisymmetry,” namely  $(\bigvee\{\bar{\Delta}_{tt'} \mid t < t' \text{ and } t' = t\sigma\})$ . But in case (a), we need further variables  $a_j$  for  $1 \leq j \leq T$ , where  $T$  is the number of signed involutions that are fixedpoint-free. We append the clause  $(a_1 \vee \dots \vee a_T)$ , and also  $(\bar{a}_j \vee \Delta_{tt'})$  for all  $t < t'$  such that  $t' = t\sigma$  when  $\sigma$  corresponds to index  $j$ . Those clauses say, “there’s at least one signed involution that is an antisymmetry.”

normal chain  
preprocessor  
colexicographic order  
author  
full-adder

There are no solutions when  $n \leq 3$ . Answers for (a) are  $((x_1 \oplus x_2) \vee x_3) \wedge x_4 \oplus x_1$  and  $((\bar{x}_1 \oplus x_2) \wedge x_3) \oplus x_4 \wedge x_5 \oplus x_1$ ; in both cases the signed involution  $(1\bar{1})(2\bar{2})$  is obviously an antisymmetry. Answers for (b) are  $((x_1 \oplus x_2) \vee x_3) \wedge (x_4 \vee x_1)$  and  $((x_1 \wedge x_2) \oplus x_3) \wedge x_4 \oplus (x_5 \vee x_1)$ . [Is there a simple formula that works for all  $n$ ?]

**477.** Use the following variables for  $1 \leq h \leq m$ ,  $n < i \leq n + r$ , and  $0 < t < 2^n$ :  $x_{it}$  = ( $t$ th bit of truth table for  $x_i$ );  $g_{hi} = [g_h = x_i]$ ;  $s_{ijk} = [x_i = x_j \circ_i x_k]$ , for  $1 \leq j < k < i$ ;  $f_{ipq} = \circ_i(p, q)$  for  $0 \leq p, q \leq 1$ ,  $p + q > 0$ . (We don’t need  $f_{i00}$ , because every operation in a normal chain takes  $(0, 0) \mapsto 0$ .) The main clauses for truth table computations are

$$(\bar{s}_{ijk} \vee (x_{it} \oplus a) \vee (x_{jt} \oplus b) \vee (x_{kt} \oplus c) \vee (f_{ibc} \oplus \bar{a})), \text{ for } 0 \leq a, b, c \leq 1 \text{ and } 1 \leq j < k < i.$$

Simplifications arise in special cases: For example, if  $b = c = 0$ , the clause is omitted if  $a = 0$ , and the term  $f_{i00}$  is omitted if  $a = 1$ . Furthermore if  $t = (t_1 \dots t_n)_2$ , and if  $j \leq n$ , the (nonexistent) variable  $x_{jt}$  actually has the known value  $t_j$ ; again we omit either the whole clause or the term  $(x_{jt} \oplus b)$ , depending on  $b$  and  $t$ . For example, there usually are eight main clauses that involve  $s_{ijk}$ ; but there’s only one that involves  $s_{i12}$  when  $t < 2^{n-2}$ , namely  $(\bar{s}_{i12} \vee \bar{x}_{i1})$ , because the truth tables for  $x_1$  and  $x_2$  begin with  $2^{n-2}$  0s. (All such simplifications would be done by a preprocessor if we had defined additional variables  $f_{i00}$  and  $x_{jt}$ , and fixed their values with unit clauses.)

There also are more mundane clauses, namely  $(\bar{g}_{hi} \vee \bar{x}_{it})$  or  $(\bar{g}_{hi} \vee x_{it})$  according as  $g_h(t_1, \dots, t_n) = 0$  or 1, to fix the outputs; also  $(\bigvee_{i=n+1}^{n+r} g_{hi})$  and  $(\bigvee_{k=1}^{i-1} \bigvee_{j=1}^{k-1} s_{ijk})$ , to ensure that each output appears in the chain and that each step has two operands.

Additional clauses are optional, but they greatly shrink the space of possibilities:  $(\bigvee_{k=1}^m g_{ki} \vee \bigvee_{i'=i+1}^{n+r} \bigvee_{j=1}^{i'-1} s_{i'ji} \vee \bigvee_{i'=i+1}^{n+r} \bigvee_{j=i+1}^{i'-1} s_{i'ij})$  ensures that step  $i$  is used at least once;  $(\bar{s}_{ijk} \vee \bar{s}_{i'ji})$  and  $(\bar{s}_{ijk} \vee \bar{s}_{i'ki})$  for  $i < i' \leq n + r$  avoid reapplying an operand.

Finally, we can rule out trivial binary operations with the clauses  $(f_{i01} \vee f_{i10} \vee f_{i11})$ ,  $(f_{i01} \vee f_{i10} \vee \bar{f}_{i11})$ ,  $(\bar{f}_{i01} \vee f_{i10} \vee \bar{f}_{i11})$ . (But beware: These clauses, for  $n < i \leq n + r$ , will make it impossible to compute the trivial function  $g_1 = 0$  in fewer than three steps!)

Further clauses such as  $(\bar{s}_{ijk} \vee f_{i01} \vee \bar{x}_{it} \vee x_{jt})$  are true, but unhelpful in practice.

**478.** We can insist that the  $(j, k)$  pairs in steps  $n+1, \dots, n+r$  appear in colexicographic order; for example, a chain step like  $x_8 = x_4 \oplus x_5$  need never follow  $x_7 = x_2 \wedge x_6$ . The clauses, for  $n < i < n + r$ , are  $(\bar{s}_{ijk} \vee \bar{s}_{(i+1)j'k'})$  if  $1 \leq j' < j < k = k' < i$  or if  $1 \leq j < k$  and  $1 \leq j' < k' < k < i$ . (If  $(j, k) = (j', k')$ , we could insist further that  $f_{i01}f_{i10}f_{i11}$  is lexicographically less than  $f_{(i+1)01}f_{(i+1)10}f_{(i+1)11}$ . But the author didn’t go that far.)

Furthermore, if  $p < q$  and if each output function is unchanged when  $x_p$  is swapped with  $x_q$ , we can insist that  $x_p$  is used before  $x_q$  as an operand. Those clauses are

$$(\bar{s}_{ijq} \vee \bigvee_{n < i' < i} \bigvee_{1 \leq j' < k' < i'} [j' = p \text{ or } k' = p] s_{i'j'k'}) \text{ whenever } j \neq p.$$

For example, when answer 477 is applied to the full-adder problem, it yields  $M_r$  clauses in  $N_r$  variables, where  $(M_4, M_5) = (942, 1662)$  and  $(N_4, N_5) = (82, 115)$ . The symmetry-breaking strategy above, with  $(p, q) = (1, 2)$  and  $(2, 3)$ , raises the number of clauses to  $M'_r$ , where  $(M'_4, M'_5) = (1025, 1860)$ . Algorithm C reported ‘unsat’ after  $(1015, 291)$  kilomems using  $(M_4, M'_4)$  clauses; ‘sat’ after  $(250, 268)$  kilomems using  $(M_5, M'_5)$ . With larger problems, such symmetry breakers give significant speedup when proving unsatisfiability, but they’re often a handicap in satisfiable instances.

**479.** (a) Using the notation of the previous answer, we have  $(M_8, M'_8, N_8) = (14439, 17273, 384)$  and  $(M_9, M'_9, N_9) = (19719, 24233, 471)$ . The running times for the ‘sat’ cases with  $M_9$  and  $M'_9$  clauses were respectively  $(16, 645, 1259)$  and  $(66, 341, 1789)$  megamems—these stats are the (min, median, max) of nine runs with different random seeds. The ‘unsat’ cases with  $M_8$  and  $M'_8$  were dramatically different:  $(655631, 861577, 952218)$  and  $(8858, 10908, 13171)$ . Thus  $s(4) = 9$  in 7.1.2–(28) is optimum.

(b) But  $s(5) = 12$  is *not* optimum, despite the beauty of 7.1.2–(29)! The  $M_{11} = 76321$  clauses in  $N_{11} = 957$  variables are ‘sat’ in 680  $G\mu$ , yielding an amazing chain:

$$\begin{array}{lll} x_6 = x_1 \oplus x_2, & x_{10} = x_6 \vee x_7, & x_{14} = \bar{x}_8 \wedge x_{11}, \\ x_7 = x_1 \oplus x_3, & x_{11} = x_4 \oplus x_9, & z_1 = x_{15} = x_{10} \oplus x_{14}, \\ x_8 = x_4 \oplus x_5, & x_{12} = x_9 \oplus x_{10}, & z_2 = x_{16} = x_{12} \wedge \bar{x}_{15}. \\ x_9 = x_3 \oplus x_6, & z_0 = x_{13} = x_5 \oplus x_{11}, & \end{array}$$

And  $(M'_{10}, N_{10}) = (68859, 815)$  turns out to be ‘unsat’ in 1773 gigamems; this can be reduced to 309 gigamems by appending the unit clause  $(g_{3(15)})$ , since  $C(S_{4,5}) = 10$ .

Hence we can evaluate  $x_1 + \dots + x_7$  in only  $5 + 11 + 2 + 1 = 19$  steps, by computing  $(u_1 u_0)_2 = x_5 + x_6 + x_7$ ,  $(v_2 v_1 z_0)_2 = x_1 + x_2 + x_3 + x_4 + u_0$ ,  $(w_2 z_1)_2 = u_1 + v_1$ ,  $z_2 = v_2 \oplus w_2$ .

(c) The solver finds an elegant 8-step solution for  $(M_8, N_8) = (6068, 276)$  in 6  $M\mu$ :

$$\begin{array}{llll} x_4 = x_1 \vee x_2, & x_6 = x_3 \oplus x_4, & x_8 = x_3 \oplus x_5, & S_1 = x_{10} = x_6 \wedge x_8, \\ x_5 = x_1 \oplus x_2, & \bar{S}_0 = x_7 = x_3 \vee x_4, & S_3 = x_9 = \bar{x}_6 \wedge x_8, & S_2 = x_{11} = x_7 \oplus x_8. \end{array}$$

The corresponding  $(M'_7, N_7) = (5016, 217)$  problem is ‘unsat’ in 97  $M\mu$ .

(d) The total cost of evaluating the  $S$ ’s independently is  $3 + 7 + 6 + 7 + 3 = 26$ , using the optimum computations of Fig. 9 in Section 7.1.2. Therefore the author was surprised to discover a 9-step chain for  $S_1, S_2$ , and  $S_3$ , using the footprint heuristic:

$$\begin{array}{lll} x_5 = x_1 \oplus x_2, & x_8 = x_5 \oplus x_7, & S_3 = x_{11} = \bar{x}_8 \wedge x_9, \\ x_6 = x_1 \oplus x_3, & x_9 = x_6 \vee x_7, & S_2 = x_{12} = x_8 \wedge \bar{x}_{10}, \\ x_7 = x_3 \oplus x_4, & x_{10} = x_2 \oplus x_9, & S_1 = x_{13} = x_8 \wedge x_{10}. \end{array}$$

This chain can solve problem (d) in 13 steps; but SAT technology does it in 12(!):

$$\begin{array}{lll} x_5 = x_1 \oplus x_2, & x_9 = x_6 \vee x_7, & S_1 = x_{13} = x_8 \wedge x_{10}, \\ x_6 = x_1 \oplus x_3, & x_{10} = x_2 \oplus x_9, & S_4 = x_{14} = x_1 \wedge \bar{x}_{11}, \\ x_7 = x_3 \oplus x_4, & x_{11} = x_5 \vee x_9, & \bar{S}_0 = x_{15} = x_4 \vee x_{11}, \\ x_8 = x_5 \oplus x_7, & S_3 = x_{12} = x_8 \wedge \bar{x}_{10}, & S_2 = x_{16} = \bar{x}_8 \wedge x_{11}. \end{array}$$

The nonexistence of an 11-step solution can be proved via Algorithm C by a long computation (11034 gigamems), during which 99,999,379 clauses are learned(!).

(e) This solution (found in 342  $G\mu$ ) matches the lower bound in exercise 7.1.2–80:

$$\begin{array}{lll} x_7 = x_1 \oplus x_2, & x_{11} = x_4 \oplus x_{10}, & x_{15} = \bar{x}_9 \wedge x_{12}, \\ x_8 = x_3 \oplus x_4, & x_{12} = x_5 \oplus x_{10}, & x_{16} = x_{13} \oplus x_{15}, \\ x_9 = x_1 \oplus x_5, & x_{13} = x_8 \vee x_{11}, & x_{17} = x_{14} \wedge x_{16}. \\ x_{10} = x_6 \oplus x_8, & x_{14} = x_7 \oplus x_{12}, & \end{array}$$

(f) This solution (found in 7471  $G\mu$ ) also matches that lower bound:

$$\begin{array}{lll} x_7 = x_1 \wedge x_2, & x_{11} = x_5 \oplus x_6, & x_{15} = x_8 \oplus x_{13}, \\ x_8 = x_1 \oplus x_2, & x_{12} = x_4 \oplus x_{11}, & x_{16} = x_{10} \oplus x_{14}, \\ x_9 = x_3 \oplus x_4, & x_{13} = x_9 \oplus x_{11}, & x_{17} = x_7 \oplus x_{16}, \\ x_{10} = x_5 \wedge x_6, & x_{14} = x_9 \vee x_{12}, & x_{18} = x_{15} \vee x_{17}. \end{array}$$

Here  $x_{18}$  is the normal function  $\bar{S}_{0,4} = S_{1,2,3,5,6}$ . We beat exercise 7.1.2–28 by one step.

(g) A solution in  $t(3) = 12$  steps is found almost instantaneously (120 megamems); but 11 steps are too few (‘unsat’ in 301 gigamems).

sideways sum  
author  
footprint heuristic  
normal function

**480.** (a) Let  $x_1x_2x_3x_4 = x_1x_r y_l y_r$ . The truth tables for  $z_l$  and  $z_r$  are 0011010010001000 and 01\*\*1\*00\*011\*011, where the \*s (“don’t-cares”) are handled by simply *omitting* the corresponding clauses ( $\bar{g}_{hi} \vee \pm x_{it}$ ) in answer 477.

Less than 1 gigamem of computation proves that a six-step circuit is ‘unsat’. Here’s a seven-stepper, found in just 30 M $\mu$ :  $x_5 = x_2 \oplus x_3$ ,  $x_6 = x_3 \vee x_4$ ,  $x_8 = x_1 \oplus x_6$ ,  $x_7 = x_1 \vee x_5$ ,  $x_9 = x_6 \oplus x_7$ ,  $z_l = x_{10} = x_7 \wedge x_8$ ,  $z_r = x_{11} = x_3 \oplus x_9$ . (See exercise 7.1.2–60 for a six-step solution that is based on a *different* encoding.)

(b) Now we have the truth tables  $z_l = 0011010001001000100010000011$ ,  $z_r = 01**1*001*00*0111*00*011*01101**$ , if  $x_4x_5 = y_l y_r$ . One of many 9-step solutions is found in 6.9 gigamems:  $x_6 = x_1 \oplus x_2$ ,  $x_7 = x_2 \oplus x_5$ ,  $x_8 = x_4 \oplus x_6$ ,  $x_9 = \bar{x}_4 \wedge x_7$ ,  $x_{10} = x_1 \oplus x_9$ ,  $x_{11} = x_8 \vee x_9$ ,  $x_{12} = x_3 \oplus x_{10}$ ,  $z_r = x_{13} = x_3 \oplus x_{11}$ ,  $z_l = x_{14} = x_{11} \wedge \bar{x}_{12}$ .

The corresponding clauses for only 8 steps are proved ‘unsat’ after 190 G $\mu$  of work. (Incidentally, the encoding of exercise 7.1.2–60 does *not* have a 9-step solution.)

(c) Let  $c_n$  be the minimum cost of computing the representation  $z_l z_r$  of  $(x_1 + \dots + x_n) \bmod 3$ . Then  $(c_1, c_2, c_3, c_4) = (0, 2, 5, 7)$ , and  $c_{n-3} \leq c_n + 9$ . Hence  $c_n \leq 3n - 4$  for all  $n \geq 2$ . [This result is due to A. Kojevnikov, A. S. Kulikov, and G. Yaroslavtsev, whose paper in *LNCS 5584* (2009), 32–44, also inspired exercises 477–480.]

*Conjecture:* For  $n \geq 3$  and  $0 \leq a \leq 2$ , the minimum cost of evaluating the (single) function  $[(x_1 + \dots + x_n) \bmod 3 = a]$  is  $3n - 5 - [(n + a) \bmod 3 = 0]$ . (It’s true for  $n \leq 5$ . Here’s a 12-step computation when  $n = 6$  and  $a = 0$ , found in 2014 by Armin Biere:  $x_7 = x_1 \oplus x_2$ ,  $x_8 = x_3 \oplus x_4$ ,  $x_9 = x_1 \oplus x_5$ ,  $x_{10} = x_3 \oplus x_5$ ,  $x_{11} = x_2 \oplus x_6$ ,  $x_{12} = x_8 \oplus x_9$ ,  $x_{13} = x_8 \vee x_{10}$ ,  $x_{14} = x_7 \oplus x_{13}$ ,  $x_{15} = \bar{x}_{12} \wedge x_{13}$ ,  $x_{16} = \bar{x}_{11} \wedge x_{14}$ ,  $x_{17} = x_{11} \oplus x_{15}$ ,  $\bar{S}_{0,3,6} = x_{18} = x_{16} \vee x_{17}$ . The case  $n = 6$  and  $a \neq 0$ , which lies tantalizingly close to the limits of today’s solvers, is still unknown. What is  $C(S_{1,4}(x_1, \dots, x_6))$ ?)

**481.** (a) Since  $z \oplus z' = \langle x_1x_2x_3 \rangle$  and  $z' = x_1 \oplus x_2 \oplus x_3$ , this circuit is called a “modified full adder.” It costs one less than a normal full adder, since  $z' = (x_1 \oplus x_2) \oplus x_3$  and  $z = (x_1 \oplus x_2) \vee (x_1 \oplus x_3)$ . (And it’s the special case  $u = 0$  of the more general situation in exercise 7.1.2–28.) Part (b) describes a “modified double full adder.”

(b) The function  $z_2$  has 20 don’t-cares, so there are many eight-step solutions (although 7 are impossible); for example,  $x_6 = x_1 \oplus x_5$ ,  $x_7 = x_2 \oplus x_5$ ,  $z_3 = x_8 = x_3 \oplus x_6$ ,  $x_9 = x_4 \oplus x_6$ ,  $x_{10} = x_1 \vee x_7$ ,  $x_{11} = \bar{x}_3 \wedge x_9$ ,  $z_2 = x_{12} = x_6 \oplus x_{11}$ ,  $z_1 = x_{13} = x_{10} \oplus x_{11}$ .

(c) Letting  $y_{2k-1}y_{2k} = \llbracket x_{2k-1}x_{2k} \rrbracket$ , it suffices to show that the binary representation of  $\Sigma_n = \nu\llbracket y_1y_2 \rrbracket + \dots + \nu\llbracket y_{2n-1}y_{2n} \rrbracket + y_{2n+1}$  can be computed in at most  $8n$  steps. Four steps are enough when  $n = 1$ . Otherwise, letting  $c_0 = y_{2n+1}$ , we can compute  $z$ ’s bits with  $\nu\llbracket y_{4k-3}y_{4k-2} \rrbracket + \nu\llbracket y_{4k-1}y_{4k} \rrbracket + c_{k-1} = 2\nu\llbracket z_{2k-1}z_{2k} \rrbracket + c_k$  for  $1 \leq k \leq \lfloor n/2 \rfloor$ . Then  $\Sigma_n = 2(\nu\llbracket z_1z_2 \rrbracket + \dots + \nu\llbracket z_{n-1}z_n \rrbracket) + c_{n/2}$  if  $n$  is even,  $\Sigma_n = 2(\nu\llbracket z_1z_2 \rrbracket + \dots + \nu\llbracket z_{n-2}z_{n-1} \rrbracket) + z_n + c'$  if  $n$  is odd, where  $\nu\llbracket y_{2n-1}y_{2n} \rrbracket + c_{\lfloor n/2 \rfloor} = 2z_n + c'$ , at a cost of  $4n$  in both cases. The remaining sum costs at most  $8\lfloor n/2 \rfloor$  by induction. [See E. Demenkov, A. Kojevnikov, A. S. Kulikov, and G. Yaroslavtsev, *Information Processing Letters 110* (2010), 264–267.]

**482.** (a)  $\sum_{j=1}^k (2y_j - 1)$  is odd when  $k$  is odd, and it’s  $\pm 1$  when  $k = 1$ .

(b) Adapting Sinz’s cardinality clauses as in exercises 29 and 30, we only need the auxiliary variables  $a_j = s_j^{j-1}$ ,  $b_j = s_j^j$ , and  $c_j = s_j^{j+1}$ , because  $s_j^{j+2} = 0$  and  $s_j^j = 1$ . The clauses are then  $(\bar{b}_j \vee a_{j+1}) \wedge (\bar{c}_j \vee b_{j+1}) \wedge (b_j \vee \bar{c}_j) \wedge (a_{j+1} \vee \bar{b}_{j+1})$ , for  $1 \leq j < t/2 - 1$ ; and  $(\bar{y}_{2j-2} \vee a_j) \wedge (\bar{y}_{2j-1} \vee \bar{a}_j \vee b_j) \wedge (\bar{y}_{2j} \vee \bar{b}_j \vee c_j) \wedge (\bar{y}_{2j+1} \vee \bar{c}_j) \wedge (y_{2j-2} \vee \bar{c}_{j-1}) \wedge (y_{2j-1} \vee c_{j-1} \vee \bar{b}_j) \wedge (y_{2j} \vee b_j \vee \bar{a}_{j+1}) \wedge (y_{2j+1} \vee a_{j+1})$  for  $1 \leq j < t/2$ , omitting  $\bar{a}_1$ ,  $c_0$ , and the two clauses that contain  $y_0$ .

don’t-cares  
Kojevnikov  
Kulikov  
Yaroslavtsev  
Biere  
modified full adder  
full adder, modified  
don’t-cares  
Demenkov  
Kojevnikov  
Kulikov  
Yaroslavtsev  
Sinz

(c) Use the construction in (b) with  $y_j = x_{jd}$  for  $1 \leq d \leq n/3$  and independent auxiliary variables  $a_{j,d}$ ,  $b_{j,d}$ ,  $c_{j,d}$ . Also, assuming that  $n \geq 720$ , break symmetry by asserting the unit clause ( $x_{720}$ ). (That's much better than simply asserting ( $x_1$ )).

This problem was shown to be satisfiable if and only if  $n < 1161$  by B. Konev and A. Lisitsa [*Artificial Intelligence* **224** (2015), 103–118], thereby establishing the case  $C = 2$  of a well-known conjecture by Paul Erdős [*Michigan Math. J.* **4** (1957), 291–300, Problem 9]. Algorithm C can prove unsatisfiability for  $n = 1161$  in less than 600 gigamems, using the parameters of exercise 512.

**483.** Using a direct encoding as in (15), with  $v_{jk}$  meaning that  $v_j$  has color  $k$ , we can generate the clauses  $(\bar{v}_{jk})$  for  $1 \leq j < k \leq d$  and  $(\bar{v}_{j(k+1)} \vee \bigvee_{i=k}^{j-1} v_{ik})$  for  $2 \leq k < j \leq n$ . A similar but slightly simpler scheme works with the order encoding, when  $v_{jk}$  means that  $v_j$  has color  $> k$ . [See Ramani, Markov, Sakallah, and Aloul, *Journal of Artificial Intelligence Research* **26** (2006), 289–322. The vertices might be ordered in such a way that  $\text{degree}(v_1) \geq \dots \geq \text{degree}(v_n)$ , for example.]

Those book graphs can be colored optimally with (11, 11, 13, 11, 10) colors, respectively. Such colorings are found with less than a megamem of work by Algorithm W or Algorithm C, *without* any symmetry breaking; Algorithm L also finds them, but after more than an order of magnitude more effort. The symmetry breaking clauses actually will *retard* this search, especially in the case of *homer*. On the other hand when we ask for only (10, 10, 12, 10, 9) colors those clauses are extremely helpful: The runtime for *anna* and *david* decreases from about 350 G $\mu$  to only about 200 K $\mu$  with Algorithm C! For *huck* and *jean* the reduction is roughly 333 G $\mu \rightarrow$  833 M $\mu$  and 14 G $\mu \rightarrow$  4.3 M $\mu$ ; for *homer*, dozens or more of T $\mu$  go down to about 11 G $\mu$ . (Algorithm L is hopelessly slow on these unsatisfiable coloring problems, even with symmetry broken.)

**484.** (a) A type (iii) move will work if and only if  $v_1 \text{ --- } v_4$ ,  $v_2 \text{ --- } v_4$ ,  $v_2 \text{ --- } v_3$ .

(b) For  $0 \leq t < n - 1$  we have the clause  $(\bigvee_{k=1}^{n-t-1} q_{t,k} \vee \bigvee_{l=1}^{n-t-3} s_{t,l})$ , as well as the following for  $1 \leq i < j < n - t$ ,  $1 \leq k < n - t$ ,  $1 \leq l < n - t - 2$ :  $(\bar{q}_{t,k} \vee x_{t,k,k+1})$ ;  $(\bar{q}_{t,k} \vee \bar{x}_{t+1,i,j} \vee x_{t,i',j'})$ ;  $(\bar{s}_{t,l} \vee x_{t,l,l+3})$ ;  $(\bar{s}_{t,k} \vee \bar{x}_{t+1,i,j} \vee x_{t,i',j'})$ ; here  $i' = i + [i \geq k]$ ,  $j' = j + [j \geq k]$ , and  $\{i'', j''\}$  are the min and max of  $\{i + [i \geq l + 3] + 3[i = l], j + [j \geq l + 3] + 3[j = l]\}$ . Finally there's a unit clause  $(\bar{x}_{0,i,j})$  for all  $1 \leq i < j \leq n$  with  $v_i \text{ --- } v_j$ .

(These clauses essentially compute [ $G$  is quenchable], which is a monotone Boolean function of the  $\binom{n}{2}$  elements above the diagonal in the adjacency matrix of  $G$ . The prime implicants of this function correspond to certain spanning trees, of which there are respectively 1, 1, 2, 6, 28, 164, 1137, ... when  $n = 1, 2, 3, 4, 5, 6, 7, \dots$ )

**485.** Let  $t' = t + 1$ . Instances of commutativity are:  $(q_{t,k}, q_{t',k'}) \leftrightarrow (q_{t,k'+1}, q_{t',k})$  if  $k < k'$ ;  $(s_{t,l}, s_{t',l'}) \leftrightarrow (s_{t,l'+1}, s_{t',l})$  if  $l + 2 < l'$ ;  $(q_{t,k}, s_{t',l'}) \leftrightarrow (s_{t,l'+1}, q_{t',l})$  if  $k < l'$ ;  $(s_{t,l}, q_{t',k'}) \leftrightarrow (q_{t,k'+1}, s_{t',l})$  if  $l + 2 < k'$ ;  $(s_{t,l}, s_{t',l}) \leftrightarrow (q_{t,l+3}, s_{t',l})$ . These can be broken by appending the clauses  $(\bar{q}_{t,k'+1} \vee \bar{q}_{t',k})$ ,  $(\bar{s}_{t,l'+1} \vee \bar{s}_{t',l})$ , ...,  $(\bar{q}_{t,l+3} \vee \bar{s}_{t',l})$ .

Endomorphisms are also present in the two cases  $(q_{t,k}, q_{t',k}) \leftrightarrow (q_{t,k+1}, q_{t',k})$  and  $(s_{t,k+1}, q_{t',k}) \leftrightarrow (q_{t,k+1}, s_{t',k})$ , provided that both pairs of transitions are legal. These are exploited by the clauses  $(\bar{q}_{t,k+1} \vee \bar{q}_{t',k} \vee \bar{x}_{t,k,k+1})$  and  $(\bar{q}_{t,k+1} \vee \bar{s}_{t',k} \vee \bar{x}_{t,k+1,k+4})$ .

**486.** This game is a special case of graph quenching, so we can use the previous two exercises. Algorithm C finds a solution after about 1.2 gigamems, without the symmetry-breaking clauses; this time goes down to roughly 85 megamems when those clauses are added. Similarly, the corresponding 17-card problem after  $\mathbf{A}\clubsuit \times \mathbf{J}\clubsuit$  is found to be unsatisfiable, after 15 G $\mu$  without and 400 M $\mu$  with. ( $\mathbf{A}\clubsuit \times \mathbf{10}\clubsuit$  fails too.)

Those SAT problems have respectively (1242, 20392, 60905), (1242, 22614, 65590), (1057, 15994, 47740), (1057, 17804, 51571) combinations of (variables, clauses, cells),

auxiliary variables  
break symmetry  
Konev  
Lisitsa  
Erdős  
direct encoding  
order encoding  
Ramani  
Markov  
Sakallah  
Aloul  
WalkSAT  
comparison of running times  
monotone Boolean function  
adjacency matrix  
prime implicants  
Endomorphisms  
quenching

and they are *not* handled easily by Algorithms A, B, D, or L. In one solution *both*  $q_{0,11}$  and  $s_{0,7}$  are true, thus providing two ways to win(!), when followed by  $q_{1,15}$ ,  $s_{2,13}$ ,  $q_{3,12}$ ,  $s_{4,10}$ ,  $s_{5,7}$ ,  $q_{6,7}$ ,  $s_{7,5}$ ,  $q_{8,5}$ ,  $s_{9,4}$ ,  $q_{10,5}$ ,  $s_{11,3}$ ,  $q_{12,3}$ ,  $s_{13,1}$ ,  $s_{14,1}$ ,  $q_{15,1}$ ,  $q_{16,1}$ .

*Notes:* This mildly addictive game is an interesting way to waste time in case you ever get lost with a pack of cards on a desert island. If you succeed in reducing the original 18 piles to a single pile, you can continue by dealing 17 more cards and trying to reduce the new 18 piles. And if you succeed also at that, you have 17 more cards for a third try, since  $52 = 18 + 17 + 17$ . Three consecutive wins is a Grand Slam.

In a study of ten thousand random deals, just 4432 turned out to be winnable. Computer times (with symmetry breaking) varied wildly, from 1014  $K\mu$  to 37  $G\mu$  in the satisfiable cases (median 220  $M\mu$ ) and from 46  $K\mu$  to 36  $G\mu$  in the others (median 848  $M\mu$ ). The most difficult winnable and unwinnable deals in this set were respectively

$$9\spadesuit 7\clubsuit 3\clubsuit K\heartsuit 7\spadesuit 3\heartsuit 2\heartsuit 8\clubsuit 6\heartsuit J\heartsuit 8\spadesuit 2\heartsuit 6\spadesuit 4\heartsuit 5\spadesuit 4\heartsuit 10\heartsuit Q\spadesuit \quad \text{and} \\ A\heartsuit Q\heartsuit 2\heartsuit 9\heartsuit 7\clubsuit 7\heartsuit 8\heartsuit K\clubsuit 3\heartsuit 10\clubsuit 3\clubsuit 3\spadesuit Q\spadesuit 8\clubsuit 2\clubsuit K\spadesuit 6\heartsuit 5\clubsuit.$$

Students in Stanford's graduate problem seminar investigated this game in 1989 [see K. A. Ross and D. E. Knuth, Report STAN-CS-89-1269 (Stanford Univ., 1989), Problem 1]. Ross posed an interesting question, still unsolved: Is there a sequence of (say) nine "poison cards," such that all games starting with those cards are lost?

The classic game Idle Year is also known by many other names, including Tower of Babel, Tower of London, Accordion, Methuselah, and Skip Two. Albert H. Morehead and Geoffrey Mott-Smith, in *The Complete Book of Solitaire and Patience Games* (1949), 61, suggested that moves shouldn't be too greedy.

**487.** Every queen in a set of eight must attack at least 14 vacant cells. Thus  $|\partial S|$  gets its minimum value  $8 \times 14 = 112$  when the queens occupy the top row. Solutions to the 8 queens problem, when queens are independent, all have  $|\partial S| \leq 176$ . The maximum  $|\partial S|$  is 184, achieved symmetrically for example in Fig. A-9(a). (This problem is *not* at all suitable for SAT solvers, because the graph has 728 edges. The best way to proceed is to run through all  $\binom{64}{8}$  possibilities with the revolving door Gray code (Algorithm 7.2.1.3R), because incremental changes to  $|\partial S|$  are easy to compute when a queen is deleted or inserted. The total time by that method is only 601 gigamems.)

The maximum of  $|\partial_{\text{out}} S|$  is obviously  $64 - 8 = 56$ . The minimum, which corresponds to Turton's question, is 45; it can be achieved symmetrically as in Fig. A-9(b), leaving  $64 - 8 - 45 = 11$  cells unattacked (shown as black queens). In this case SAT solvers win: The revolving door method needs 953 gigamems, but SAT methods show the impossibility of 44 after only 2.2  $G\mu$  of work. With symmetry reduction as in the following exercise, this goes down to 900  $M\mu$  although there are 789 variables and 4234 clauses. [Bernd Schwarzkopf, in *Die Schwalbe* **76** (August 1982), 531, computed all solutions of minimum  $|\partial_{\text{out}} S|$ , given  $|S|$ , for  $n \times n$  boards with  $n \leq 8$ . Extensions of Turton's problem to larger  $n$  have been surveyed by B. Lemaire and P. Vitushinskiy in two articles, written in 2011 and accessible from [www.ffjm.org](http://www.ffjm.org). Optimum solutions for  $n > 16$  are conjectured but not yet known.]

All sets  $S$  of eight queens trivially have  $|\partial_{\text{in}} S| = 8$ .

**488.** Let variables  $w_{ij}$  and  $b_{ij}$  represent the presence of white or black queens on cell  $(i, j)$ , with clauses  $(\bar{w}_{ij} \vee \bar{b}_{i'j'})$  when  $(i, j) = (i', j')$  or  $(i, j) \text{ --- } (i', j')$ . Also, if each army is to have at least  $r$  queens, add clauses based on (20) and (21) to ensure that  $\sum w_{ij} \geq r$  and  $\sum b_{ij} \geq r$ . Optionally, add clauses based on Theorem E to ensure that  $k$  of the  $w$  variables for the top row are lexicographically greater than or equal to the

Stanford  
Ross  
Knuth  
poison cards  
Tower of Babel  
Tower of London  
Accordion  
Methuselah  
Skip Two  
Morehead  
Mott-Smith  
8 queens problem  
revolving door  
Gray code  
symmetry reduction  
Schwarzkopf  
Lemaire  
Vitushinskiy  
lexicographically

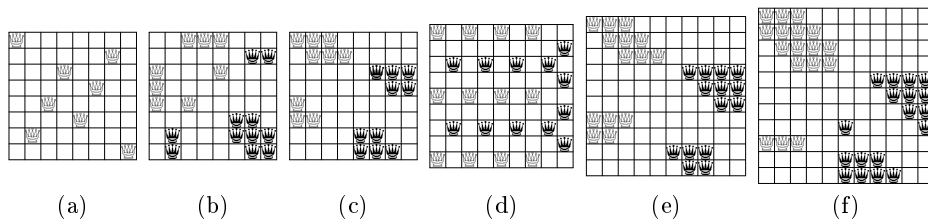


Fig. A-9. Optimum queen placements of various kinds.

breaking the symmetries  
 Smith  
 Petrie  
 Gent  
 CSP: The constraint satisfaction problem  
 asymptotic  
 Gardner  
 Chung  
 Graham  
 lex-leaders  
 Sims tables

corresponding  $k$  variables in fifteen symmetrical variants. (For instance, if  $k = 3$ , we might require  $w_{11}w_{12}w_{13} \geq b_{1n}b_{2n}b_{3n}$ , thus partially breaking the symmetries.)

The maximum army sizes for  $1 \leq n \leq 11$  are found to be 0, 0, 1, 2, 4, 5, 7, 9, 12, 14, and 17, respectively. A construction with 21 armies is known for  $n = 12$ , but 22 has not yet been proved impossible. [B. M. Smith, K. E. Petrie, and I. P. Gent obtained similar results using CSP methods in *LNC3* 3011 (2004), 271–286.] An extra black queen can actually be included in the cases  $n = 2, 3, 4, 6, 8, 10$ , and 11. Solutions appear in Fig. A-9; the construction shown in Fig. A-9(d) generalizes to armies of  $2q(q+1)$  queens whenever  $n = 4q+1$ , while those in parts (c), (e), (f) belong to another family of constructions that achieve the higher asymptotic density  $\frac{9}{64}n^2$ .

When  $n = 8$  and  $r = 9$ , Algorithm C typically finds a solution in about 10 megamems ( $k = 0$ ), or about 30 megamems ( $k = 3$ ); but with  $r = 10$  it typically proves unsatisfiability in about 1800  $M\mu$  ( $k = 0$ ) or 850  $M\mu$  ( $k = 3$ ) or 550  $M\mu$  ( $k = 4$ ) or 600  $M\mu$  ( $k = 5$ ). Thus the symmetry breaking constraints are helpful for unsatisfiability in this case, but not for the easier satisfiability problem. On the other hand, the extra constraints do turn out to be helpful for both the satisfiable and unsatisfiable variants when  $n$  is larger. The “sweet spot” turns out to be  $k = 6$  when  $n = 10$  and  $n = 11$ ; unsatisfiability was proved in those cases, with  $r = 15$  and  $r = 18$ , after about 185  $G\mu$  and 3500  $G\mu$ , respectively. [See Martin Gardner, *Math Horizons* 7, 2 (November 1999), 2–16, for generalizations to coexisting armies of sizes  $r$  and  $s$ . F. R. K. Chung and R. L. Graham conjecture that the maximum value of  $s$ , if  $r = 3q^2 + 3q + 1$ , is asymptotically  $n^2 - (6q+3)n + O(1)$ .]

**489.**  $T_0 = 1, T_1 = 2, T_n = 2T_{n-1} + (2n-2)T_{n-2}$  (see Eq. 5.1.4-(40)). The generating function  $\sum_n T_n z^n / n!$  and the asymptotic value are given in exercise 5.1.4-31.

**490.** Yes. For example, using the signed permutation  $\bar{4}13\bar{2}$ , we’re allowed to assume that some solution satisfies  $\bar{x}_4 x_1 x_3 \bar{x}_2 \leq \bar{x}'_4 x'_1 x'_3 \bar{x}'_2$  for every endomorphism — because the solution with lexicographically smallest  $\bar{x}_4 x_1 x_3 \bar{x}_2$  has this property. Notice that the signed permutation  $\bar{1}\bar{2} \dots \bar{n}$  converts ‘ $\leq$ ’ to ‘ $\geq$ ’.

**491.** Let  $\sigma$  be the permutation  $(1234\bar{1}\bar{2}\bar{3}\bar{4})$ . Then  $\sigma^4 = (1\bar{1})(2\bar{2})(3\bar{3})(4\bar{4})$ ; and by Theorem E we need only search for solutions that satisfy  $x_1 x_2 x_3 x_4 \leq \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4$ . We’re therefore allowed to append the clause  $(\bar{x}_1)$  without affecting satisfiability.

(We actually are allowed to assert that  $x_1 = x_2 = x_4 = 0$ , because 0000 and 0010 are the lex-leaders of the two 8-cycles when  $\sigma$  is written as a permutation of states.)

In general if an automorphism  $\sigma$  is a permutation of literals having a cycle that contains both  $v$  and  $\bar{v}$ , for some variable  $v$ , we can simplify the problem by assigning a fixed value to  $v$  and then by restricting consideration to automorphisms that don’t change  $v$ . (See the discussion of Sims tables in Section 7.2.1.2.)

**492.** Suppose  $x_1 \dots x_n$  satisfies all clauses of  $F$ ; we want to prove that  $(x_1 \dots x_n)\tau = x'_1 \dots x'_n$  also satisfies them all. And that’s easy: If  $(l_1 \vee \dots \vee l_k)$  is a clause, we have



$l'_1 = l_1\tau, \dots, l'_n = l_n\tau$ ; and we know that  $(l_1\tau \vee \dots \vee l_n\tau)$  is true because it's subsumed by a clause of  $F$ . [See S. Szeider, *Discrete Applied Math.* **130** (2003), 351–365.]

**493.** Using the global ordering  $p_1 \dots p_9 = 543219876$  and Corollary E, we can add clauses to assert that  $x_5 = 0$  and  $x_4x_3x_2x_1 \leq x_6x_7x_8x_9$ . A contradiction quickly follows, even if we stipulate only the weaker relation  $x_4 \leq x_6$ , because that forces  $x_6 = 1$ .

**494.** Exercise 504(b) shows that  $(uv)(\bar{u}\bar{v})$  is a symmetry of the underlying Boolean function, although not necessarily of the clauses  $F$ . [This observation is due to Aloul, Ramani, Markov, and Sakallah in the cited paper.] The other symmetries allow us to assert (i)  $(\bar{x}_i \vee x_j) \wedge (\bar{x}_j \vee \bar{x}_k)$ , (ii)  $(\bar{x}_i \vee \bar{x}_j) \wedge (\bar{x}_j \vee \bar{x}_k)$ , (iii)  $(\bar{x}_i \vee \bar{x}_j) \wedge (\bar{x}_j \vee x_k)$ .

**495.** Suppose, for example, that  $m = 3$  and  $n = 4$ . The variables can then be called 11, 12, 13, 14, 21,  $\dots$ , 34; and we can give them the global ordering 11, 12, 21, 13, 22, 31, 14, 23, 32, 24, 33, 34. To assert that  $21\ 22\ 23\ 24 \leq 31\ 32\ 33\ 34$ , we use the involution that swaps rows 2 and 3; this involution is (21 31)(22 32)(23 33)(24 34) when expressed in form (192) with signs suppressed. Similarly we can assert that  $12\ 22\ 13 \leq 13\ 23\ 33$  because of the involution (12 13)(22 23)(32 33) that swaps columns 2 and 3. The same argument works for any adjacent rows or columns. And we can replace ' $\leq$ ' by ' $\geq$ ', by complementing all variables.

For general  $m$  and  $n$ , consider any global ordering for which  $x_{ij}$  precedes or equals  $x_{i'j'}$  when  $1 \leq i \leq i' \leq m$  and  $1 \leq j \leq j' \leq n$ . The operation of swapping adjacent rows makes the global lexicographic order increase if and only if it makes the upper row increase lexicographically; and the same holds for columns.

[See Ilya Shlyakhter, *Discrete Applied Mathematics* **155** (2007), 1539–1548.]

**496.** No; that reasoning would “prove” that  $m$  pigeons cannot fit into  $m$  holes. The fallacy is that his orderings on rows and columns aren't simultaneously consistent with a *single* global ordering, as in the previous exercise.

**497.** A BDD with 71,719 nodes makes it easy to calculate the total, 818,230,288,201, as well as the generating function  $1 + z + 3z^2 + 8z^3 + 25z^4 + \dots + 21472125415z^{24} + 31108610146z^{25} + \dots + 10268721131z^{39} + 6152836518z^{40} + \dots + 24z^{60} + 8z^{61} + 3z^{62} + z^{63} + z^{64}$ . (The relatively small coefficients of  $z^{39}$  and  $z^{40}$  help account for the fact that  $\geq$  was chosen in (185)–(186); problems with sparse solutions tend to favor  $\geq$ .)

[Pólya's theorem in Section 7.2.3 shows that exactly 14,685,630,688 inequivalent matrices exist; compare this to  $2^{64} \approx 1.8447 \times 10^{19}$  without any symmetry reduction.]

**498.** Consider the global ordering  $x_{01}, x_{11}, \dots, x_{m1}; x_{12}, x_{22}, \dots, x_{m2}, x_{02}; x_{23}, x_{33}, \dots, x_{m3}, x_{03}, x_{13}; \dots; x_{(m-1)m}, x_{mm}, x_{0m}, \dots, x_{(m-2)m}$ . There's a column symmetry that fixes all elements preceding  $x_{(j-1)j}$  and takes  $x_{(j-1)j} \mapsto x_{(j-1)k}$ .

**499.** No. The unusual global ordering in answer 498 is not consistent with ordinary lexicographic row or column ordering. [Nor can the analogous clauses  $(x_{ii} \vee \bar{x}_{ij})$  for  $1 \leq i \leq m$  and  $i < j \leq n$  be appended to (185) and (186). No quad-free matrix for  $m = n = 4$  and  $r = 9$  satisfies all those constraints simultaneously.]

**500.** If  $F_0$  has a solution, then it has a solution for which  $l$  is true. But  $(F_0 \cup F_1) \mid l$  might be unsolvable. (For example, let  $F_0 = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_1)$ , which has the symmetry  $\bar{1}2$ ; so we can take  $S = (\bar{x}_1), l = \bar{x}_1$ . Combine that with  $F_1 = (x_1)$ .)

**501.** Let  $x_{ij}$  denote a queen in cell  $(i, j)$ , for  $1 \leq i \leq m$  and  $1 \leq j \leq n$ . Also let  $r_{ij} = [x_{i1} + \dots + x_{ij} \geq 1]$  and  $r'_{ij} = [x_{i1} + \dots + x_{i(j+1)} \geq 2]$ , for  $1 \leq i \leq m$  and  $1 \leq j < n$ . Using (18) and (19) we can easily construct about  $8mn$  clauses that define the  $r$ 's in terms of the  $x$ 's and also ensure that  $x_{i1} + \dots + x_{in} \leq 2$ . Thus  $r'_{i(n-1)} = [x_{i1} + \dots + x_{in} = 2]$ ; call this condition  $r_i$ .

Szeider  
Aloul  
Ramani  
Markov  
Sakallah  
Shlyakhter  
fallacy  
generating function  
Pólya's theorem  
global ordering  
quad-free

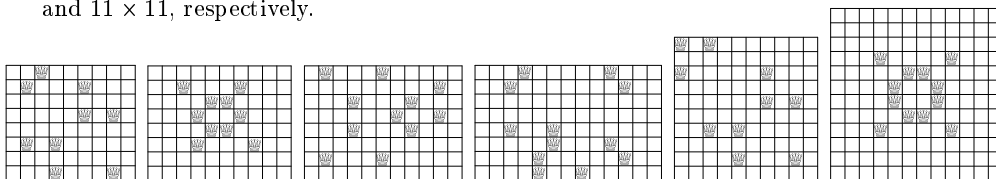
Similar conditions  $c_j$ ,  $a_d$ , and  $b_d$  are readily established for column  $j$ , and for the diagonals with  $i+j = d+1$  or  $i-j = d-n$ , for  $1 \leq i \leq m$ ,  $1 \leq j \leq n$ , and  $1 \leq d < m+n$ . Then condition (ii) corresponds to the  $mn$  clauses  $(x_{ij} \vee r_i \vee c_j \vee a_{i+j-1} \vee b_{i-j+n})$ .

Finally we have clauses from (20) and (21) to ensure that  $\sum x_{ij} \leq r$ .

When  $m = n$ , the lower bound  $r \geq n - [n \bmod 4 = 3]$  has been established by A. S. Cooper, O. Pikhurko, J. R. Schmitt, and G. S. Warrington [AMM **121** (2014), 213–221], who also used backtracking to show that  $r \geq 12$  on an  $11 \times 11$  board. SAT methods, with symmetry breaking, yield that result much more quickly (after about 9 teramems of computation); but this problem, like the tomography problem of Fig. 36, is best solved by integer programming techniques when  $m$  and  $n$  are large.

If we call the upper left corner white, solutions with  $m = n = r - 1$  and all queens on white squares appear to exist for all  $n > 2$ , and they are found almost instantly. However, no general pattern is apparent. In fact, when  $n$  is odd it appears possible to insist that the queens all appear in odd-numbered rows *and* in odd-numbered columns.

Here are examples of optimum placements on smallish boards. The solutions for  $8 \times 9$ ,  $8 \times 10$ ,  $8 \times 13$ ,  $10 \times 10$ , and  $12 \times 12$  also work for sizes  $8 \times 8$ ,  $9 \times 10$ ,  $8 \times 12$ ,  $9 \times 9$ , and  $11 \times 11$ , respectively.



This placement of ten queens on a  $10 \times 10$  board can be described by the “magic sequence”  $(a_1, \dots, a_5) = (1, 3, 7, 5, 9)$ , because the queens appear in positions  $(a_i, a_{i+1})$  and  $(a_{i+1}, a_i)$  for  $1 \leq i < n/2$  as well as in  $(a_1, a_1)$  and  $(a_{n/2}, a_{n/2})$ . The magic sequences  $(1, 3, 9, 13, 15, 5, 11, 7, 17)$  and  $(9, 3, 1, 19, 5, 11, 15, 25, 7, 21, 23, 13, 17)$  likewise describe optimum placements for  $n = 18$  and  $26$ . No other magic sequences are known; none exist when  $n = 34$ .

**502.** For each  $j$ , construct independent cardinality constraints for the relation  $x_1^{(j)} + \dots + x_n^{(j)} \leq r_j$ , using say (20) and (21), where  $x_k^{(j)} = (s_{jk} ? \bar{x}_k : x_k)$ .

**503.** The Hamming distance  $d(x, y) = \nu(x \oplus y)$  between binary vectors of length  $n$  satisfies  $d(x, y) + d(\bar{x}, y) = n$ . Thus there is no  $x$  with  $d(x, s_j) \geq r_j + 1$  for all  $j$  if and only if there is no  $x$  with  $d(\bar{x}, s_j) \leq n - 1 - r_j$  for all  $j$ . [See M. Karpovsky, *IEEE Transactions IT-27* (1981), 462–472.]

**504.** (a) Assume that  $n \geq 4$ . For strings of length  $2n$  we have  $d(z, w) + d(z, \bar{w}) = 2n$ ; hence  $d(z, w) \leq n$  and  $d(z, \bar{w}) \leq n$  if and only if  $d(z, w) = d(z, \bar{w}) = n$ . Every string  $z$  with  $z_{2k-1} \neq z_{2k}$  for  $1 \leq k \leq n$  satisfies  $d(z, w_j) = n$  for  $1 \leq j \leq n$ . Conversely, if  $d(z, w_j) = d(z, w_k) = n$  and  $1 \leq j < k \leq n$ , then  $z_{2j-1} + z_{2j} = z_{2k-1} + z_{2k}$ . Thus if  $z_{2j-1} = z_{2j}$  for some  $j$  we have  $z = 00 \dots 0$  or  $11 \dots 1$ , contradicting  $d(z, w_1) = n$ .

(b) For each string  $\hat{x} = \bar{x}_1 x_1 \bar{x}_2 x_2 \dots \bar{x}_n x_n$  that satisfies part (a) we have  $d(\hat{x}, y) = 2\bar{l}_1 + 2\bar{l}_2 + 2\bar{l}_3 + n - 3$ , which is  $\leq n + 1$  if and only if  $(l_1 \vee l_2 \vee l_3)$  is satisfied.

(c) Let  $s_j = w_j$  and  $r_j = n$  for  $1 \leq j \leq 2n$ ; let  $s_{2n+k} = y_k$  and  $r_{2n+k} = n + 1$  for  $1 \leq k \leq m$ , where  $y_k$  is the string in (b) for the  $k$ th clause of  $F$ . This system has a closest string  $\hat{x} = \bar{x}_1 x_1 \bar{x}_2 x_2 \dots \bar{x}_n x_n$  if and only if  $x_1 \dots x_n$  satisfies every clause. [A similar construction in which all strings have length  $2n + 1$  and all  $r_j$  are equal to  $n + 1$  is obtained if we append the bit  $[n < j \leq 2n]$  to each  $s_j$ . See M. Frances and A. Litman, *Theory of Computing Systems* **30** (1997), 113–119.]

Cooper  
Pikhurko  
Schmitt  
Warrington  
symmetry breaking  
tomography problem  
integer programming  
magic sequences  
cardinality constraints  
Hamming distance  
Karpovsky  
Frances  
Litman

(d) Boilerplate 11000000, 00110000, 00001100, 00000011, 00111111, 11001111, 11110011, 00000011, at distance  $\leq 4$ ; for the clauses, 01011000, 00010110, 01000101, 10010001, 10100100, 00101001, 10001010, and possibly 01100010, at distance  $\leq 5$ .

**505.** (For  $k = 0, 1, \dots, n - 1$  one can set  $j$  to a uniform integer in  $[0..k]$  and  $\text{INX}[k + 1] \leftarrow j$ ; also if  $j = k$  set  $\text{VAR}[k] \leftarrow k + 1$ , otherwise  $i \leftarrow \text{VAR}[j]$ ,  $\text{VAR}[k] \leftarrow i$ ,  $\text{INX}[i] \leftarrow k$ ,  $\text{VAR}[j] \leftarrow k + 1$ .) With nine random seeds, typical runtimes for D3 are (1241, 873, 206, 15, 748, 1641, 1079, 485, 3321)  $M\mu$ . They're much less variable for the unsatisfiable K0, namely (1327, 1349, 1334, 1330, 1349, 1322, 1336, 1330, 1317)  $M\mu$ ; and even for the satisfiable W2: (172, 192, 171, 174, 194, 172, 172, 170, 171)  $M\mu$ .

mutilated  
Looking ahead  
ParamILS  
pi  
double lookahead  
author

**506.** (a) *Almost* true: That sum is the total number of clauses of length  $\geq 2$ , because every such clause of length  $k$  contributes  $1/\binom{k}{2}$  to the weights of  $\binom{k}{2}$  edges.

(b) Each of the  $12^2 - 2 = 142$  cells of the mutilated  $12 \times 12$  board contributes one positive clause  $(v_1 \vee \dots \vee v_k)$  and  $\binom{k}{2}$  negative clauses  $(\bar{v}_i \vee \bar{v}_j)$ , when that cell can be covered by  $k$  potential dominoes  $\{v_1, \dots, v_k\}$ . So the weight between  $u$  and  $v$  is 2,  $4/3$ , or  $7/6$  when dominoes  $u$  and  $v$  overlap in a cell that can be covered in 2, 3, or 4 ways. Exactly 6 cells can be covered in just 2 ways (and exactly  $10^2$  in 4 ways).

(The largest edge weights in all of Fig. 52 are  $37/6$ , between 20 pairs of vertices in K6. At the other extreme, 95106 of the 213064 edges in X3 have the tiny weight  $1/8646$ , and 200904 of them have weight at most twice that much.)

**507.** Consider, for example, the clauses  $(u \vee \bar{t})$ ,  $(v \vee \bar{t})$ ,  $(\bar{u} \vee \bar{v} \vee t)$ ,  $(u \vee \bar{t}')$ ,  $(v \vee \bar{t}')$ ,  $(\bar{u} \vee \bar{v} \vee t')$  from (24). Looking ahead from  $t = 1$  yields the windfall  $(\bar{t} \vee t')$ , and looking ahead from  $t' = 1$  yields  $(\bar{t}' \vee t)$ . Henceforth Algorithm L knows that  $t$  equals  $t'$ .

**508.** According to (194), the purging parameters were  $\Delta_p = 1000$  and  $\delta_p = 500$ ; thus we have learned approximately  $1000k + 500\binom{k}{2}$  clauses when doing the  $k$ th purging phase. After  $1000L$  clauses this works out to be  $\approx (\sqrt{16L + 9} - 3)/2$  phases, which is  $\approx 34.5$  when  $L = 323$ . (And the actual number was indeed 34.)

**509.** One remedy for overfitting is to select training examples at random. In this case such randomness is already inherent, because of the different seeds used while training.

**510.** (a) From Fig. 53 or Fig. 54 or Table 7 we know that  $T1 < T2 < L6$  in the median rankings; thus T2 obscures L6 and T1.

(b) Similarly,  $L8 < M3 < Q2 < X6 < F2 < X4 < X5$ ; X6 obscures L8 and X4.

(c) X7 obscures K0, K2, and (indirectly) A2, because K2 obscures K0 and A2.

**511.** (a) Nine random runs finished in only (4.9, 5.0, 5.1, 5.1, 5.2, 5.2, 5.3, 5.4, 5.5)  $M\mu$ (!).

(b) Nine random runs now each were aborted after a teramem of trials. (No theoretical explanation for this discrepancy, or for the wildness of P4 in Fig. 54, is known.)

(c) (0.2, ..., 0.5, ..., 3.2)  $M\mu$  without; (0.3, ..., 0.5, ..., 0.7)  $M\mu$  with.

**512.** A training run with ParamILS in 2015 suggested the parameters

$$\alpha = 0.7, \quad \rho = 0.998, \quad \varrho = 0.99995, \quad \Delta_p = 100000, \quad \delta_p = 2000, \\ \tau = 10, \quad w = 1, \quad p = 0, \quad P = 0.05, \quad \psi = 0.166667, \quad (*)$$

which produce the excellent results in Fig. A-10.

**513.** After training on  $\text{rand}(3, 1062, 250, 314159)$ , ParamILS choose the values  $\alpha = 3.5$  and  $\Theta = 20.0$  in (195), together with distinctly different values that favor double lookahead, namely  $\beta = .9995$ ,  $Y = 32$ . [The untuned values  $\alpha = 3.3$ ,  $\beta = .9985$ ,  $\Theta = 25.0$ , and  $Y = 8$  had been used by the author when preparing exercise 173.]

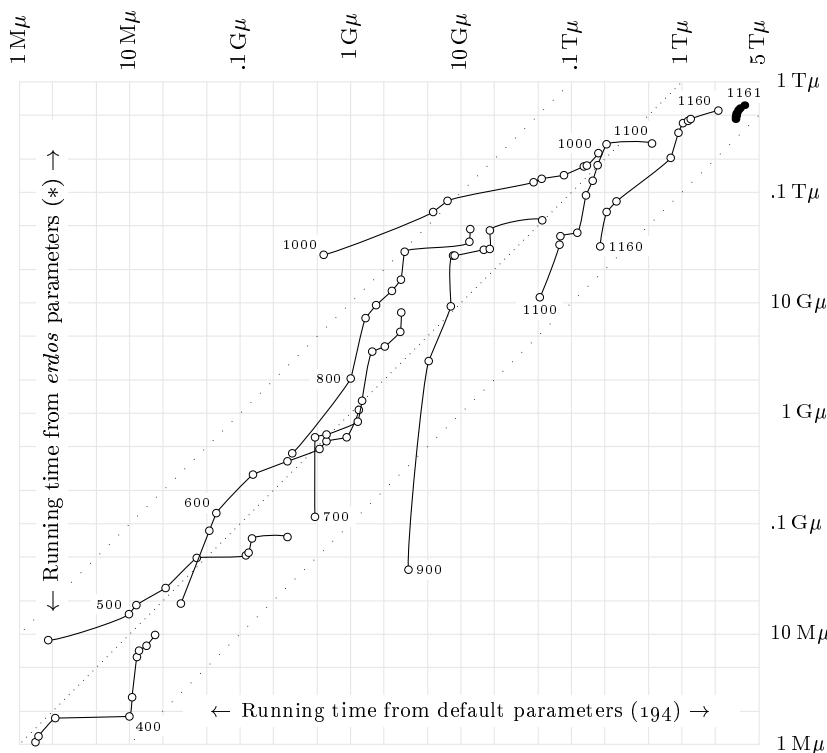


Fig. A-10. Running times for Algorithm C, with and without special parameter tuning.

514. ParamILS suggests  $p = .85$  and  $N = 5000n$ ; that gives a median time  $\approx 690 M\mu$ . (But those parameters give horribly *bad* results on most other problems.)

515. Use variables  $S_{ijk}$  meaning that cell  $(i, j)$  in the solution holds  $k$ , and  $Z_{ij}$  meaning that cell  $(i, j)$  is blank in the puzzle. The 729  $S$  variables are constrained by  $4 \times 81 \times (1 + \binom{9}{2}) = 11,988$  clauses like (13). From condition (i), we need only 41 variables  $Z_{ij}$ . Condition (ii) calls for 15 clauses such as  $(Z_{11} \vee \dots \vee Z_{19})$ ,  $(Z_{11} \vee \dots \vee Z_{51} \vee Z_{49} \vee \dots \vee Z_{19})$ ,  $(Z_{15} \vee \dots \vee Z_{55})$ ,  $(Z_{44} \vee Z_{45} \vee Z_{46} \vee Z_{54} \vee Z_{55})$ , when equal  $Z$ 's are identified via (i). Condition (iii), similarly, calls for 28 clauses such as  $(\bar{Z}_{11} \vee \bar{Z}_{12} \vee \bar{Z}_{13})$ ,  $(\bar{Z}_{11} \vee \bar{Z}_{21} \vee \bar{Z}_{31})$ ,  $(\bar{Z}_{45} \vee \bar{Z}_{55})$ . Condition (vi) is enforced by 34,992 clauses epitomized by  $(\bar{S}_{111} \vee \bar{Z}_{11} \vee \bar{S}_{122} \vee \bar{Z}_{12} \vee \bar{S}_{412} \vee \bar{Z}_{41} \vee \bar{S}_{421} \vee \bar{Z}_{42})$ .

For conditions (iv) and (v), we introduce auxiliary variables  $V_{ijk} = S_{ijk} \wedge \bar{Z}_{ij}$ , meaning that  $k$  is visible in  $(i, j)$ ;  $R_{ik} = V_{i1k} \vee \dots \vee V_{i9k}$ , meaning that  $k$  is visible in row  $i$ ;  $C_{jk} = V_{1jk} \vee \dots \vee V_{9jk}$ , meaning that  $k$  is visible in column  $j$ . Also  $B_{bk} = \bigvee_{\langle i,j \rangle = b} V_{ijk}$ , meaning that  $k$  is visible in box  $b$ ; here  $\langle i, j \rangle = 1 + 3[(i-1)/3] + [(j-1)/3]$ . Then  $P_{ijk} = Z_{ij} \wedge \bar{R}_{ik} \wedge \bar{C}_{jk} \wedge \bar{B}_{\langle i,j \rangle k}$  means that  $k$  is a possible way to fill cell  $(i, j)$  without conflict. These 1701 auxiliary variables are defined with 8262 clauses.

Condition (iv) is enforced by nine 9-ary clauses for each  $i$  and  $j$ , stating that we mustn't have exactly one of  $\{P_{ij1}, \dots, P_{ij9}\}$  true. Condition (v) is similar, enforced by three sets of  $81 \times 9$  clauses of length 9; for example, one of those clauses is

$$(P_{417} \vee P_{427} \vee P_{437} \vee \bar{P}_{517} \vee \bar{P}_{527} \vee P_{537} \vee P_{617} \vee P_{627} \vee P_{637}).$$

(“We aren’t obviously forced to put 7 into box 4 by using cell (5, 2).”)

Finally, some of the symmetry is usefully broken by asserting the unary clauses  $S_{1kk} \wedge \bar{Z}_{11} \wedge Z_{12}$ . The grand total is 58,212 clauses with 351,432 cells, on 2,471 variables.

(This problem was suggested by Daniel Kroening. There are zillions of solutions, and about one in every five or six appears to be completable uniquely to the setting of the  $S$  variables. Thus we can obtain as many “hard sudoku” puzzles as we like, by adding additional unary clauses such as  $S_{553} \wedge \bar{Z}_{17}$  more or less at random, then weeding out ambiguous cases via dancing links. The clauses are readily handled by Algorithms L or C, but they’re often too difficult for Algorithm D. That algorithm did, however, find the uniquely completable solution (a) below after only 9.3 gigamems of work.)

If we beef up condition (iii), insisting now that no box contains a row or column with more than *one* blank, condition (vi) becomes superfluous. We get solutions such as (b) below, remarkable for having no forced moves in spite of 58 visible clues, yet uniquely completable. That puzzle is, however, quite easy; only 2, 4, 7 are unplaced.

|                   |                   |                   |                   |
|-------------------|-------------------|-------------------|-------------------|
| 1 . . . 6 . 8 .   | 1 . 3 . 5 6 . 8 9 | 1 . 3 . 5 . 7 . . | 1 . 3 . 5 6 . 8 9 |
| 5 . 8 7 2 1 4 . 6 | 5 9 7 3 8 . 6 1 . | . 5 . 7 9 . . . 1 | 6 8 . 3 . 9 1 . 5 |
| . 6 . 3 8 . 2 . 1 | 6 8 . 1 . 9 3 . 5 | 7 . . . . 1 2 5 . | . 9 5 1 8 . 6 3 . |
| 8 4 . . . 3 . . 5 | 9 5 6 . 3 1 8 . 7 | . . 1 . . 5 . 7 6 | 3 . 8 9 6 . . 5 1 |
| . . 5 . 6 . 8 . . | . 3 1 5 . 8 9 6 . | . . 5 . 7 . 1 . . | . 1 9 5 . 8 3 6 . |
| 6 . . 8 . . . 4 2 | 2 . 8 9 6 . 1 5 3 | 4 7 . 1 . . 5 . . | 5 6 . . 3 1 9 . 8 |
| 3 . 6 . 4 8 . 2 . | 8 . 9 6 . 5 . 3 1 | . 1 8 5 . . . . 7 | . 5 6 . 9 3 8 1 . |
| 4 . 7 6 3 2 1 . 8 | . 6 5 . 1 3 2 9 8 | 5 . . . 8 7 . 1 . | 8 . 1 6 . 5 . 9 3 |
| . 8 . 5 . . . . 4 | 3 1 . 8 9 . 5 . 6 | . . 7 . 1 . 8 . 5 | 9 3 . 8 1 . 5 . 6 |
| (a)               | (b)               | (c)               | (d)               |

We might also try to strengthen conditions (iv) and (v) by requiring at least *three* ways to make each choice, not just two. Then we get solutions like (c) above. Unfortunately, however, that one is completable in 1237 ways! Even if we also strengthen condition (iii) as in (b), we get solutions like (d), which can be completed in 12 ways. No uniquely completable sudoku puzzles are known to have such ubiquitous threefold ambiguity.

**516.** This conjecture can be expressed in several equivalent forms. R. Impagliazzo and R. Paturi [*JCSS* **62** (2001), 367–375] defined  $s_k = \inf\{\lg \tau \mid \text{there exists an algorithm to solve } k\text{SAT in } \tau^n \text{ steps}\}$ , and stated the *exponential time hypothesis*:  $s_3 > 0$ . They also defined  $s_\infty = \lim_{k \rightarrow \infty} s_k$ , and proved that  $s_k \leq (1 - d/k)s_\infty$  for some positive constant  $d$ . They conjectured that  $s_\infty = 1$ ; this is the strong exponential time hypothesis. An alternative formulation [C. Calabro, R. Impagliazzo, and R. Paturi, *IEEE Conf. on Computational Complexity* **21** (2006), 252–260] was found later: “If  $\tau < 2$ , there is a constant  $\alpha$  such that no randomized algorithm can solve every SAT problem with  $\leq \alpha n$  clauses in fewer than  $\tau^n$  steps, where  $n$  is the number of variables.”

**517.** (a) If there are  $n$  variables, introduce  $\binom{2n}{2}$  new variables  $ll' = l'l$ , one for each pair of literals  $\{l, l'\}$ , with the equations  $ll' + \bar{l}\bar{l}' + \bar{l} = 1$ . Similarly, introduce  $\binom{2n}{3}$  variables  $ll'l''$ , via  $ll'l'' + ll'\bar{l}'' + \bar{l}\bar{l}'\bar{l}'' + \bar{l} = 1$ . Then the ordinary ternary clause  $l \vee l' \vee l''$  is true if and only if we have  $ll'l'' + ll'\bar{l}'' + \bar{l}\bar{l}'\bar{l}'' + \bar{l}l'l'' + \bar{l}l'\bar{l}'' + \bar{l}\bar{l}'l'' + \bar{l}\bar{l}'\bar{l}'' = 1$ .

(b) Remove clauses of length  $> 3$  by using the fact that  $l_1 + \dots + l_k = 1$  if and only if  $l_1 + \dots + l_j + t = 1$  and  $l_{j+1} + \dots + l_k + \bar{t} = 1$ , where  $t$  is a new variable. Also, if  $a, b, c$ , and  $d$  are new variables with  $a + b + d = a + c + \bar{d} = 1$ , beef up short clauses using  $l + l' = 1 \iff l + l' + a = 1$  and  $l = 1 \iff \bar{l} + b + c = 1$ .

symmetry breaking  
 Kroening  
 dancing links  
 Impagliazzo  
 Paturi  
 exponential time hypothesis  
 Calabro  
 density of clauses

[Thomas J. Schaefer proved the NP-completeness of 1-in-3 SAT as a special case of considerably more general results, in *STOC* 10 (1978), 216–226.]

**518.** (a)  $A = \begin{pmatrix} x & y & y \\ y & x & y \\ y & y & x \end{pmatrix}$ , where  $x = \begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix}$ ,  $y = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$ .

(b) Twice in the  $n$  variable rows and  $n$  variable columns; once in the  $3m$  output rows and  $3m$  input columns; never in the  $3m$  input rows and  $3m$  output columns.

(c) By (a), each way to choose 2s in different rows and columns contributes zero to the permanent unless, in every clause, the subset of chosen inputs is nonempty and matches the chosen outputs. In the latter case it contributes  $16^m 2^n$ . [See A. Ben-Dor and S. Halevi, *Israel Symp. Theory of Computing Systems* 2 (IEEE, 1993), 108–117.]

**519.** The unsatisfiable problem corresponding to D1 and D2 has median running time 2099 M $\mu$  (losing to both *factor\_fifo* and *factor\_lifo*). The satisfiable one corresponding to D3 and D4 is unstable (as in Fig. 54), with median 903 M $\mu$  (winning over both).

**520.** (Solution by Sven Mallach, 2015, using solvers X and Y, where X was CPLEX 12.6 and Y was GUROBI 6, both used with emphasis on mixed-integer-program feasibility, constant objective function, and solution limit 1.) With a time cutoff of 30 minutes on a single-threaded Xeon computer, neither X nor Y could solve any of the 46 problems A1, A2, C1, C2, C3, C4, C5, C6, C8, D1, D2, E1, E2, F1, F2, G1, G2, G5, G6, G7, G8, K7, K8, M5, M7, M8, O1, O2, P0, P1, P2, Q7, S3, S4, T5, T6, T7, T8, W2, W4, X1, X3, X5, X6, X7, X8. (In particular, this list includes P0, S4, and X1, which are extremely easy for Algorithm C.) On the other hand both X and Y solved the *langford* problems L3 and L4—which were the toughest for Algorithm C—in less than a second.

Algorithm C performs about 20 G $\mu$  per minute on a comparable Xeon. In these experiments it significantly outperformed the geometric methods except on problems K0, K1, K2, L3, L4, and P4 (and some easy cases such as B2).

Of course we must keep in mind that the particular clauses in Table 6 aren't necessarily the best ways to solve the corresponding combinatorial problems with an IP solver, just as they aren't necessarily the best encodings for a SAT solver. We are comparing here only black-box clause-solving speeds.

**521.** A variety of simple schemes has been surveyed by S. Jabbour, J. Lonlac, L. Saïs, and Y. Salhi, [arXiv:1402.1956](https://arxiv.org/abs/1402.1956) [cs.AI] (2014), 13 pages.

**522.** For cycles of length  $T$  we can introduce  $27T$  variables  $xyz_t$  for  $1 \leq x, y, z \leq 3$  and  $0 \leq t < T$ , signifying that vertex  $(x, y, z)$  occupies slot  $t$  in the path. Binary exclusion clauses  $\neg xyz_t \vee \neg x'y'z'_t$ , when  $xyz = x'y'z'$  and  $t \neq t'$  or when  $xyz \neq x'y'z'$  and  $t = t'$ , ensure that no vertex appears twice in the path, and that no two vertices occupy the same slot. A valid path is specified via the adjacency clauses

$$\neg xyz_t \vee \bigvee \{x'y'z'_{(t+1) \bmod T} \mid 1 \leq x', y', z' \leq 3 \text{ and } |x' - x| + |y' - y| + |z' - z| = 1\}.$$

We represent the shadows by introducing 36 variables  $a!b*$ ,  $ba!*$ ,  $a!*b$ ,  $b*a!$ ,  $*a!b$ ,  $*ba!$  for  $1 \leq a \leq 2$  and  $1 \leq b \leq 3$ ; here  $a!*b$  (for example) means that the shadow of  $(x, z)$  coordinates has a transition between  $(a, b)$  and  $(a+1, b)$ . These variables appear in ternary clauses such as  $(\neg xyz_t \vee \neg(x+1)yz'_t \vee x!*z) \wedge (\neg xyz_t \vee \neg(x+1)yz'_t \vee x!y*)$  whenever  $x < 3$  and  $t' \equiv t \pm 1 \pmod{T}$ . To exclude loops we append clauses like

$$\neg 1!1* \vee \neg 2!1* \vee \neg 3!1* \vee \neg 3!2* \vee \neg 2!3* \vee \neg 2!2* \vee \neg 1!2* \vee \neg 1!1*;$$

this one excludes the loop in the example illustration. There are 39 such loop-defeating clauses, one for each of the 13 simple cycles in each shadow.

Schaefer  
Ben-Dor  
Halevi  
Mallach  
CPLEX  
GUROBI  
Xeon computer  
*langford*  
per minute  
Jabbour  
Lonlac  
Saïs  
Salhi  
exclusion clauses  
at-most-one

Finally we can break symmetry by asserting the unary clauses  $121_{T-1}$ ,  $111_0$ ,  $112_1$  without loss of generality, after verifying that no solution can avoid all eight corners.

Clearly  $T$  must be an even number, because the graph is bipartite; also  $T < 27$ . If the method of exercise 12 is used for the exclusions, we obtain a total of 6264 clauses, 822 variables, and 17439 cells when  $T = 16$ ; there are 9456 clauses, 1242 variables, and 26199 cells when  $T = 24$ . These clauses are too difficult for Algorithm D. But Algorithm L resolves them almost instantaneously for any given  $T$ ; they turn out to be satisfiable if and only if  $T = 24$ , and in that case there are two essentially different solutions. One of these cycles, due to John Rickard (who introduced this problem at Cambridge University, circa 1990), is beautifully symmetric, and it is illustrated on the cover of Peter Winkler's book *Mathematical Mind-Benders* (2007). It can be represented by the delta sequence  $(322\bar{3}133\bar{1}\bar{2}112\bar{3}\bar{2}\bar{2}3\bar{1}\bar{3}\bar{3}12\bar{1}\bar{1}\bar{2})$ , where ' $k$ ' and ' $\bar{k}$ ' change coordinate  $k$  by  $+1$  or  $-1$ . The other is unsymmetric and represented by  $(3321\bar{2}1\bar{3}\bar{3}\bar{1}\bar{2}21\bar{2}323\bar{1}\bar{1}\bar{3}1\bar{2}\bar{1}\bar{3}\bar{2})$ .

**523.** (Solution by Peter Winkler.) With coordinates  $(x, y, z)$  for  $1 \leq x \leq m$ ,  $1 \leq y \leq n$ ,  $1 \leq z \leq 2$ , any cycle with loopless shadows must contain at least two steps  $(x, y, 1) \text{---} (x, y, 2)$  and  $(x', y', 1) \text{---} (x', y', 2)$ . We can assume that  $x < x'$  and that  $x' - x$  is minimum. The  $m \times 2$  shadow contains  $(x, 1) \text{---} (x, 2)$  and  $(x', 1) \text{---} (x', 2)$ , together with (say) the path  $(x, 1) \text{---} \dots \text{---} (x', 1)$ , but without the edge  $(x'', 2) \text{---} (x''+1, 2)$  for some  $x''$  with  $x \leq x'' < x'$ . The unique shortest path from  $(x, y)$  to  $(x', y')$  in the  $m \times n$  shadow contains some edge  $(x'', y'') \text{---} (x''+1, y'')$ ; hence  $(x'', y'', 1) \text{---} (x''+1, y'', 1)$  must occur twice in the cycle.

**524.** This problem involves clauses very much like those for a cyclic path, but simpler; we have  $T = 27$  and no “wrap-around” conditions. With typically 1413 variables, 10410 clauses, and 28701 cells, Algorithm L shines again, needing only a gigamem or two to handle each of several cases that break symmetry based on starting and ending points. There are four essentially different solutions, each of which can be assumed to start at 111; one ends at 333, another at 133, another at 113, and the other at 223. Using the delta sequence notation above, they are:  $332\bar{3}\bar{3}2331\bar{3}\bar{3}\bar{2}33\bar{2}\bar{3}\bar{3}1332\bar{3}\bar{3}233$  (which is reflected ternary code);  $31\bar{3}133\bar{1}\bar{1}211\bar{3}\bar{3}\bar{1}3\bar{1}\bar{3}231\bar{3}133\bar{1}\bar{1}$ ;  $32\bar{3}231\bar{3}\bar{2}2\bar{3}\bar{3}132\bar{3}233\bar{2}\bar{1}22\bar{1}\bar{2}$ ;  $1122\bar{1}\bar{1}213\bar{1}211\bar{2}\bar{2}\bar{1}\bar{3}1122\bar{1}\bar{1}$ .

[Such paths, and more generally spanning trees that have loopless shadows, were invented in 1983 by Oskar van Deventer, who called them “hollow mazes”; see *The Mathematician and Pied Puzzler* (1999), 213–218. His *Mysterians* puzzle is based on an amazing Hamiltonian path on  $P_5 \square P_5 \square P_5$  that has loopless shadows.]

**525.** The author's best solution, as of July 2015, had 100 variables, 400 clauses, and 1200 literals (cells); it was derived from Tseytin's examples of exercise 245, applied to a more-or-less random 4-regular graph of girth 6 on 50 vertices. Tseytin's construction, with one odd vertex and 49 even ones, yields 400 clauses of 4SAT, which are quite challenging indeed. It can be simplified to a 3SAT problem by insisting further that every even vertex must have degree exactly 2 in the subgraph specified by true edges. (See K. Markström, *J. Satisfiability, Boolean Modeling and Comp.* **2** (2006), 221–227).

That simplified problem still turned out to be fairly challenging: It was proved unsatisfiable by Algorithm L in 3.3 T $\mu$  and by Algorithm C in 1.9 T $\mu$ . (But by applying the endomorphisms of exercise 473, which broke symmetry by adding 142 clauses of length 6, the running time went down to just 263 M $\mu$  and 949 M $\mu$ , respectively.)

Another class of small-yet-difficult problems is worth mentioning, although it doesn't fit the specifications of this exercise [see I. Spence, *ACM J. Experimental Algorithmics* **20** (2015), 1.4:1–1.4:14]: Every instance of 3D matching whose representation

break symmetry  
unary clauses  
bipartite  
exclusions  
at-most-one  
Rickard  
Winkler  
delta sequence  
Winkler  
break symmetry  
reflected ternary code  
spanning trees  
van Deventer  
hollow mazes  
Mysterians  
author  
Tseytin  
graph-based axioms  
parity-related clauses  
4-regular  
girth  
4SAT  
Markström  
comparison of run times  
endomorphisms  
broke symmetry  
Spence  
3D matching

as an exact cover problem has  $5n$  rows and  $3n$  columns, with five 1s in each column and three 1s in each row, can be represented as a SAT problem in  $3n$  variables,  $10n$  binary clauses, and  $2n$  quinary clauses, hence only  $30n$  total literals. This 5SAT problem has the same number of literals as the 3SAT problem discussed above, when  $n = 40$ ; yet it is considerably more difficult if the matching problem is unsatisfiable. (On the other hand, the problem of this kind that defeated all the SAT solvers in the 2014 competition corresponds to a matching problem that is solved almost instantaneously by the dancing links method: Algorithm 7.2.2.1D needs less than  $60 M\mu$  to prove it unsatisfiable.)

**526.** We prove by induction on  $|F|$  that it's possible to leave at most  $w(F)$  clauses unsatisfied, where  $w(F) = \sum_{C \in F} 2^{-|C|}$ . If all clauses of the multiset  $F$  are empty we have  $w(F) = |F|$ , and the result holds. Otherwise suppose the variable  $x$  appears in  $F$ . Let  $l = x$  if  $w(\{C \mid x \in C \in F\}) \geq w(\{C \mid \bar{x} \in C \in F\})$ ; otherwise  $l = \bar{x}$ . A simple calculation shows that  $w(F|l) \leq w(F)$ . [*JCSS* **9** (1974), 256–278, Theorem 3.]

**999.** ...

exact cover problem  
5SAT  
competition  
dancing links



## INDEX TO ALGORITHMS AND THEOREMS

- Algorithm 7.2.2.2A, 28–29, 208.
- Algorithm 7.2.2.2A\*, 208–209.
- Algorithm 7.2.2.2B, 31.
- Lemma 7.2.2.2B, 58.
- Theorem 7.2.2.2B, 59.
- Algorithm 7.2.2.2C, 68.
- Theorem 7.2.2.2C, 52–54.
- Algorithm 7.2.2.2D, 33–34.
- Algorithm 7.2.2.2E, 176.
- Corollary 7.2.2.2E, 113.
- Theorem 7.2.2.2E, 111.
- Algorithm 7.2.2.2F, 205.
- Theorem 7.2.2.2F, 86.
- Theorem 7.2.2.2G, 70.
- Algorithm 7.2.2.2I, 61.
- Theorem 7.2.2.2J, 82.
- Algorithm 7.2.2.2K, 216–217.
- Theorem 7.2.2.2K, 90.
- Algorithm 7.2.2.2L, 38–39.
- Algorithm 7.2.2.2L', 212.
- Lemma 7.2.2.2L, 82.
- Theorem 7.2.2.2L, 82.
- Algorithm 7.2.2.2M, 82–83.
- Theorem 7.2.2.2M, 83.
- Algorithm 7.2.2.2P, 77.
- Algorithm 7.2.2.2P', 241.
- Program 7.2.2.2P', 242.
- Algorithm 7.2.2.2R, 146.
- Theorem 7.2.2.2R, 56.
- Algorithm 7.2.2.2S, 93.
- Theorem 7.2.2.2S, 87.
- Algorithm 7.2.2.2T, 249–250.
- Theorem 7.2.2.2U, 78–79.
- Algorithm 7.2.2.2W, 79–80, 242.
- Corollary 7.2.2.2W, 79.
- Algorithm 7.2.2.2X, 44–45.

## INDEX AND GLOSSARY

GOLDSMITH

*The republic of letters is at present divided into three classes.  
One writer, for instance, excels at a plan or a title-page,  
another works away the body of the book,  
and a third is a dab at an index.*

— OLIVER GOLDSMITH, in *The Bee* (1759)

When an index entry refers to a page containing a relevant exercise, see also the *answer* to that exercise for further information. An answer page is not indexed here unless it refers to a topic not included in the statement of the exercise.

- $\partial S$  (boundary set), 58, 154, 180, 188.
- 0–1 matrices, 106–109, 151, 176–177, 181,  
  *see also* Grid patterns.
- 1SAT, 49, 148.
- 2-colorability of hypergraphs, 185.
- 2SAT, 49, 51–54, 77–78, 80, 101, 144,  
  147, 149, 157, 159, 266.
- 3-regular graphs, 147, 154, 231.
- 3CNF, 3, 148.
- 3D MATCHING problem, 134, 225, 290–291.
- 3D visualizations, 116–118.
- 3SAT, 3–4, 47–51, 59, 60, 78–80, 93–94, 131,  
  135, 146, 148–151, 153, 182–184, 231.
- 4-cycles, 109–110, 178, 225, 274.
- 4-regular graphs, 290.
- 4SAT, 49, 51, 150, 290.
- 5SAT, 51, 58, 224, 291.
- 6SAT, 51.
- 7SAT, 51, 151.
- 8 queens problem, 25, 282.
- 90°-rotational symmetry, 138, 202, 275.
- 100 test cases, 113–124, 127, 182, 184.
- $\emptyset$  (the empty set), 185.
- $\epsilon$  (the empty clause), 3, 27, 185, 291.
- $\epsilon$  (the empty string), 3, 85.
- $\epsilon$  (the tolerance for convergence), 93–94.
- $\epsilon$  (offset in heuristic scores), 126, 213.
- $\nu x$  (1s count), *see* Sideways sum.
- $\pi$  (circle ratio), *see* Pi.
- $\rho$  (damping factor for variable activity),  
  67, 125–127, 155, 286.
- $\rho$  (damping factor for reinforcement), 93–94.
- $\varrho$  (damping factor for clause activity),  
  74, 125–127, 286.
- $\tau$  parameter, 125–127, 235, 286.
- $\tau(a, b)$  function, 147.
- $\phi$  (golden ratio), 146, 147, 160, 251.
- $\psi$  (agility threshold), 76–77, 124–127,  
  240, 286.
- $\psi$  (confidence level), 93, 255.
- a.s.: almost surely, 149, 153.
- AAAI: American Association for Artificial  
  Intelligence (founded in 1979);
- Association for the Advancement of  
  Artificial Intelligence (since 2007), 67.
- Absorbed clauses, 168.
- Accordion solitaire, 282.
- Achlioptas, Dimitris (Αχλιόπτας,  
  Δημήτρης), 221.
- ACT( $c$ ), 74, 125.
- ACT( $k$ ), 66–68, 75, 125, 132.
- Active path, 13.
- Active ring, 32.
- Activity scores, 67, 74–76, 125, 132,  
  155, 239.
- Acyclic orientation, 161.
- Adams, Douglas Noel (42), 126.
- Adaptive control, 46, 126.
- Addition, encoding of, 100–101, 114; *see*  
  *also* Full adders, Half adders.
- Adjacency matrix, 281.
- Adjacent pairs of letters, avoiding, 248.
- AGILITY, 76, 158, 240.
- Agility level, 76, 124, 158.
- Agility threshold ( $\psi$ ), 76–77, 124–127,  
  240, 286.
- Ahmed, Tanbir (তানবীর আহমেদ), 5, 147.
- Alava, Mikko Juhani, 80.
- Aldous, David John, 219.
- Algorithm  $L^0$ , 39, 147.
- Alice, 20–24, 139–141.
- All-different constraint, 171.
- All solutions, 143, 266.
- Alon, Noga (נוגה אלון), 174, 254, 260.
- Aloul, Fadi Ahmed (فادي أحمد العالول),  
  112, 281, 284.
- Analysis of algorithms, 146–152,  
  158–160, 164.
- Ancestors, 43.
- AND operation, 9, 10, 13.  
  bitwise ( $x \& y$ ), 28, 29, 31, 37, 38, 66, 68,  
  76, 81, 196, 209–211, 220, 241.
- André, Pascal, 131.
- Anisimov, Anatoly Vasilievich (Анисимов,  
  Анатолий Васильевич), 249.
- Annexstein, Fred Saul, 274.
- Anti-maximal-element clauses, 56, 62, 97,  
  115, 153, 155, 157, 167.

- Antisymmetry, 178.  
 Appier dit Hanzelet, Jean, 57.  
 April Fool, 7.  
 Ardila Mantilla, Federico, 256.  
 Arithmetic progressions, 4, 114.  
   avoiding, 135.  
 Armies of queens, 180.  
 Asín Achá, Roberto Javier, 267.  
 Asserting clause, *see* Forcing clause.  
 Associative block design, 4.  
 Associative law, 227.  
 Asymmetric Boolean functions, 178.  
 Asymmetric elimination, 260.  
 Asymmetric tautology, *see* Certifiable clauses.  
 Asymptotic methods, 53–54, 147–151, 164, 210, 226, 230, 283.  
 At-least-one constraint, 171, 265.  
 At-most-one constraint, 6, 97–99, 103, 104, 120, 134, 149, 170, 171, 238, 265, 266, 289.  
 ATPG: Automatic test pattern generation, *see* Fault testing.  
 Atserias, Albert Perí, 262.  
 Audemard, Gilles, 72.  
 Aurifeuille, Léon François Antoine, factors, 14.  
 Autarkies, 44, 71, 146, 152, 177, 214, 215, 217.  
   testing for, 146, 214.  
 Autarky principle, 44.  
 Automatic test pattern generation, *see* Fault testing.  
 Automaton, 272.  
 Automorphisms, 108, 111, 180, 197, 236, 277.  
 Autosifting, 220.  
 Auxilliary variables, 6, 8, 15, 17, 60, 97, 101, 104, 105, 109, 135, 136, 148, 170–174, 186, 262, 268, 276–279, 280–281, 287.  
 AVAIL stack, 257.  
 Averages, 120.  
 Avoiding submatrices, 106–107.  
 Awkward trees, 227.  
 Axiom clauses, 54, 59, 100, 264, 266.
- Bacchus, Fahiem, 73, 271.  
 Backjumping, 64, 68, 74, 132, 233, 236, 239.  
 Backtrack trees, *see* Search trees.  
 Backtracking, 4, 27–34, 38–39, 64, 105, 128, 129, 132, 151, 176, 190, 204, 219, 231, 236.  
 Bailleux, Olivier, 8, 26, 135, 137, 143, 174, 272.  
 Baker, Andrew Baer, 98.  
 Balas, Egon, 206.  
 Baldassi, Carlo, 93.  
 Ball, Walter William Rouse, 180.  
 Ballot numbers, 78.
- Balls and urns, 221.  
 Banbara, Mutsunori (番原睦則), 264, 267, 268.  
 Bartley, William Warren, III, 129.  
 Basket weavers, 141.  
 Batcher, Kenneth Edward, 266.  
 Baumert, Leonard Daniel, 265.  
 Bayardo, Roberto Xavier, Jr., 132.  
 Bayes, Thomas, networks, 95.  
 BCP: Boolean constraint propagation, *see* Unit propagation.  
 BDD: A reduced, ordered binary decision diagram, 17–18, 102, 103, 132, 137, 148, 174, 181, 188, 193, 194, 197, 202, 220.  
 BDD base, 219.  
 Belief propagation, 95.  
 Ben-Dor, Amir (אמיר בן-דור), 289.  
 Ben-Sasson, Eli (אלי בן-ששין), 57–58, 153, 231.  
 Benchmark tests, 35, 131–133, 139, 147, 190, 206.  
   100 test cases, 113–124, 127, 182, 184.  
 Bender, Edward Anton, 250.  
 Beresin, May, 275.  
 Berghammer, Rudolf, 204.  
 BerkMin solver, 132.  
 Berlekamp, Elwyn Ralph, 17.  
 Berman, Piotr, 224.  
 Bernhart, Frank Reiff, 188.  
 Bernoulli, Jacques (= Jakob = James), distribution, multivariate, 89.  
 Bethe, Hans Albrecht, 95.  
 Better reasons, 157.  
 Bias messages, 92.  
 Biased random bits, 12, 241.  
 Biere, Armin, v, 66, 76, 96, 129, 132, 166, 188, 258, 260, 261, 269, 280.  
 Big clauses, 145.  
 BIMP tables, 36–41, 43, 45, 124, 144, 235.  
 Binary addition, 114.  
 Binary clauses, 3, 6, 36, 124, 133, 155–156.  
 Binary constraints, 171.  
 Binary decoder, 179.  
 Binary implication graph, *see* Dependency digraph, 41.  
 Binary matrices, 106–109, 151, 176–177, 181, *see also* Grid patterns.  
 Binary multiplication, 8.  
 Binary number system, 9, 98.  
 Binary recurrence relations, 189.  
 Binary relations, 56.  
 Binary search, 187.  
 Binary strings, 181.  
 Binary tensor contingency problem, 142, 151.  
 Binomial coefficients, 149.  
 Binomial convolutions, 250.  
 Bipartite graphs, 58, 177, 290.  
 Bipartite matching, 150.

- Bipartite structure, 90.
- Birthday paradox, 49.
- Bishops, 141.
- Bitmaps, 17, 139.
- Bitwise operations, 11, 12, 81, 158, 161, 241, 246, 258–259.
- Black and blue principle, 146, 216.
- Black and white principle, 146.
- Blake, Archie, 130.
- blit*, 234, 236.
- Block decomposition, 275.
- Block designs, 106.
- Block diagonal matrices, 177.
- Blocked clauses, 102, 215, 260, 261, 269.
  - binary, 146.
  - elimination of, 167.
- Blocked self-subsumption, 167.
- Blocking digraph, 215.
- Blocks in Life, 197, 200.
- Bloom, Burton Howard, coding, 258.
- Bloom, Thomas Frederick, 185.
- Bob, 20–24, 115, 139–141.
- Böhm, Max Joachim, 131.
- Bollobás, Béla, 54, 220.
- Bonacina, Maria Paola, 129.
- book* graphs, 126, 179.
- Boole, George, 129.
- Boolean chains, 9, 11, 12, 102, 114, 173.
  - optimum, 178–179.
- Boolean formulas, 1.
- Boolean functions, 14–16.
  - expressible in  $k$ CNF, 220.
  - synthesis of, 178–179.
- Boppana, Ravi Babu, 174.
- Borgs, Christian, 54.
- Bottom-up algorithms, 252.
- Boufkhad, Yacine (ياسين بوفخاد), 8, 26, 131, 135, 137, 143, 174, 272.
- Boundary sets, 58, 154, 180, 188.
- Boundary variables, 230.
- Bounded model checking, 16–24, 132, 137–141, 157, 179–180.
- Branching heuristics, 105, 144, *see also* Decision literals.
- Branching programs, 102, 173, 174.
- Branchless computation, 242.
- Braunstein, Alfredo, 90, 91, 256.
- Breadth-first search, 37, 43, 68, 130, 235.
- Break count, 79.
- Breaking symmetries, vii, 5, 19, 105–114, 138, 176–181, 187, 188, 190–192, 238, 267, 281–283, 285, 288–290.
  - in graph coloring, 99–100, 114, 171, 179, 187.
- Broadcasting, 170.
- Broadword computations, 11, 12, 158, 161, 246, 258.
- Brown, Cynthia Ann Blocher, 30, 32, 131, 151, 226.
- Brown, Thomas Craig, 185.
- Brummayer, Robert Daniel, 269.
- Brunetti, Sara, 206.
- Bryant, Randal Everitt, v, 7, 187.
- BST( $l$ ), 211.
- BSTAMP counter, 211.
- Buckingham, David John, 197, 200.
- Buddy system, 36, 144, 235.
- Bugrara, Khaled Mohamed (خالد محمد ابوغراره), 226.
- Bugs, 16, 69, 77, 133, 240.
- Bulnes-Rozas, Juan Bautista, 215.
- Bumped process, 21.
- Bundala, Daniel, 196.
- Burney, Charles, viii.
- Burns, James Edward, 204.
- Buro, Michael, 131.
- Buss, Samuel Rudolph, v, 153, 270.
- Bystanders, *see* Easy clauses.
- C-SAT solver, 131.
- Cache memories, 24.
- Calabro, Chris, 288.
- Candidate variables, 40–44, 131, 214.
- Canonical forms, 138, 248.
- Cardinality constraints, 7–8, 26, 104, 106, 113, 114, 121, 135, 143, 187, 188, 193, 194, 196, 204, 285.
  - for intervals, 100, 190, 280.
- Carlier, Jacques, 131.
- Carlitz, Leonard, 162.
- Carriers in Life, 197, 200.
- Carroll, Lewis (= Dodgson, Charles Lutwidge), 129–130.
- Carry bits, 9, 12, 101, 192, 193.
- Cartier, Pierre Emile, 83, 86, 163.
- Case analysis, 27, 130.
- CDCL (conflict driven clause learning) solvers, 62–71, 103, 121, 132–133, 155.
  - combined with lookahead solvers, 129.
  - compared to lookahead solvers, 98–100, 118–121, 182, 290.
- Cells of memory, 28, 122–124.
- Cellular automata, 17, 202.
- Certifiable clauses, 168, 260.
- Certificates of unsatisfiability, 69–71, 157, 169, 176, 178.
- Chaff solver, 67, 132.
- Chain rule for conditional probability, 254.
- Chains, *see* Boolean chains, Resolution chains,  $s$ -chains.
- Channel assignment, 136.
- Channeling clauses, 264.
- Characteristic polynomial of a matrix, 163, 218.
- Chavas, Joël, 91.
- Chayes, Jennifer Tour, 54.
- Chebyshev (= Tschebyscheff), Pafnutii Lvovich (Чебышевъ, Пафнутий

- Львовичъ = Чебышев, Пафнутий Львович), inequality, 221.  
 polynomials, 247.
- Cheshire Tom, 24–26, 115, 142–143.
- Chess, 7, 170.
- Chessboards, 18, 25, 99, 106, 115, 138, 180.
- Chiral symmetry (rotation but not reflection), 138, 202, 275.
- Chordal graphs, 163–164.
- Chromatic number  $\chi(G)$ , 99, 135–136, 147, 174, 281.
- Chung Graham, Fan Rong King (鍾金芳蓉), 283.
- Chvátal, Václav (= Vašek), 5, 52, 59, 185.
- Cimatti, Alessandro, 132.
- Circuits, Boolean, 10, 101–103, 114, *see also* Boolean chains.
- Circular lists, 32.
- Clarke, Edmund Melson, Jr., 132.
- Clashing pairs of letters, 84.
- Clausal proofs, *see* Certificates of unsatisfiability.
- Clause activity scores, 74, 239.
- Clause-learning algorithms, 61–62, 103, 118, 121, 132–133, 154–155.
- Clauses per literal, 150, 231; *see also* Density of clauses.
- Claw graph, 249.
- Clichés, 76.
- Clique Local Lemma, 165.
- Cliques, 81, 100, 162, 167, 169, 171, 179, 190.  
 covering by, 165.
- Closest strings, 114, 181, 182.
- Clusters, 166.
- CNF: Conjunctive normal form, 9, 101, 154, 173, 193, 196.
- Cocomparability graphs, 249, 250.
- Coe, Timothy Vance, 201.
- Coexisting armies of queens, 180.
- Cographs, 163, 250.
- Cohen, Bram, 79, 246.
- Coja-Oghlan, Amin, 221.
- Colexicographic order, 206, 278.
- Coloring a graph, 6–7, 99–100, 153, 179, 260.  
 fractional, 135.  
 multiple, 135.  
 of queens, 99–100, 114–115, 171.  
 radio, 136.
- Column sums, 151.
- Commutative law, 27, 180, 227.  
 partial, 83, 250–251.
- Comparator modules, 115, 137.
- Comparison, lexicographic, 101, 111–113.
- Comparison of running times, 34–35, 39, 69, 97–100, 105–107, 110, 112, 118–128, 182, 184, 218, 237, 264, 281, 290.
- Compensation resolvents, 39, 144, 147.
- Competitions, 131–133, 291.
- Complement of a graph, 134.
- Complementation of unary representations, 100.
- Complemented literals, 2–4, 37, 62–64, 78, 111, 210, 266.
- Complete binary trees, 8, 135, 230.
- Complete bipartite graphs  $K_{m,n}$ , 250, 254.
- Complete graphs  $K_n$ , 153, 178, 186, 262.
- Complete  $k$ -partite graphs, 250, 262.
- Complete  $t$ -ary trees, 160.
- Compressing, *see* Purging unhelpful clauses.
- Conditional autarkies, 215.
- Conditional expectation inequality, 150.
- Conditional symmetries, 107, *see* Endomorphisms.
- Conditioning operations ( $F|I$  and  $F|L$ ), 27, 96, 143, 157, *see* Unit conditioning.
- Cones in trace theory, 87.
- Confidence level ( $\psi$ ), 93, 255.
- Conflict clauses, 63, 70, 171; *see also* Preclusion clauses.
- Conflict driven clause learning, 62–69, 103, 121, 132–133, 155.
- Conflicts, 62, 124, 132.
- Conjunctive normal form, 1, 9, 101, 154, 173, 193, 196.  
 irredundant, 257.
- Conjunctive prime form, 104.
- Connected graphs, 177.
- Connectedness testing, 169–170.
- Connection puzzles, 170.
- CoNP-complete problems, 3, 207.
- Consecutive 1s, 88, 175, 254.
- Consensus of implicants, 130.
- Consistent Boolean formulas, *see* Satisfiable formulas.
- Consistent partial assignments, 30, 165.
- Constrained variables in partial assignments, 165–166.
- Contests, 131–132.
- Context free languages, 175.
- Contiguous United States, 136.
- Contingency tables, binary, 142.  
 3D, 151.
- Convex functions, 216.
- Convex hulls, 247.
- Convolution principle, 250.
- Conway, John Horton, 17, 139, 201.
- Cook, Stephen Arthur, 61, 62, 130–131, 154, 229, 237.  
*cook* clauses, 157.
- Cooper, Alec Steven, 285.
- Core assignments, 166.
- Core of Horn clauses, 174, 216.
- Coupon collector's test, 220.
- Covering assignments, 166, 221, 255.
- Covering problems, 2, 193, 194, *see also* Domino coverings.
- Covering strings, 181.

- CPLEX system, 26, 289.  
 CPU: Central Processing Unit (one computer thread), 121.  
 Crawford, James Melton, Jr., 98, 113.  
 Cray 2 computer, 137.  
 Critical sections, 21–23, 140–141.  
 Crossover point, *see* Threshold of satisfiability.  
 Crusoe (= Kreutznaer), Robinson, vii.  
 CSP: The constraint satisfaction problem, 283.  
 Cube and conquer method, 129.  
 Cubic graphs (3-regular, trivalent), 147, 154, 231.  
 Cufflink pattern, 255.  
 Culver, Clayton Lee, 185.  
 Cut rule, 59.  
 Cutoff parameters, 41, 145.  
 Cutting planes, 184, 206.  
 Cycle detection problem, 260.  
 Cycle graphs  $C_n$ , 135, 160, 262.  
 Cycle structure of a permutation, 108, 112–113, 163, 178, 277.  
 Cyclic DPLL algorithm, 33.  
 Cyclic patterns, 19.  
 Cyclic permutations, 163.
- da Vinci, Leonardo di ser Piero, 7.  
 Dadda, Luigi, 9, 114, 136, 173.  
 Dags: Directed acyclic graphs, 54.  
   of resolutions, 54–56, 70.  
 Damping factors, 46, 67, 74, 76, 93–94, 125, 126, 155.  
 Dancing links, 5, 121, 134, 208, 288, 291.  
 Dantchev, Stefan Stoyanov (Данчев, Стефан Стоянов), 110.  
 Darwiche, Adnan Youssef (عدنان يوسف درويش), 67, 262.  
 Data structures, 28–34, 36–38, 43, 66–67, 80, 95–96, 143–145, 155–156, 159, 167, 238, 273.  
 Davis, Martin David, 9, 31–32, 130, 298.  
 Dawson, Thomas Rayner, 170.  
 De Morgan, Augustus, laws, 3.  
 de Vries, Sven, 206.  
 de Wilde, Boris, 213.  
 Deadlock, 22–23.  
 Debugging, 69, 77.  
 Dechter, Rina Kahana (רינה כהנא דכטר), 67.  
 Decision literals, 62, 69, 124, 132.  
 Decision trees, *see* Search trees.  
 Decomposable matrices, 177.  
 Default parameters, 93, 125–126.  
 Default values of gates, 11.  
 Definite Horn clauses, 174.  
 Defoe, Daniel (= Daniel Foe), vii.  
 Degree of a vertex, 191.  
 Degrees of truth, 37–39, 42–43, 45–46, 216.  
 Dekker, Theodorus Jozef, 140.  
 Del Lungo, Alberto, 206.  
 Delayer, 55–56, 152–153.  
 Deletion from a heap, 234.  
 Delta sequence, 290.  
 Demenkov, Evgeny Alexandrovich (Деменков, Евгений Александрович), 280.  
 Density of clauses: The number of clauses per variable, 50–51, 150, 231, 288.  
 Dependence graph in trace theory, 248.  
 Dependence of literals, 63.  
 Dependency digraph (of literals), 41, 131, 168, 215, 237, 260.  
 Dependency-directed backtracking, *see* Backjumping.  
 Dependency graph (of events), 82, 164, 165.  
 Dependency on a variable, 137.  
 Depth-first search, 130.  
 Dequen, Gilles Maurice Marceau, 131.  
 Determinants, 162, 163, 251.  
 Deterministic algorithm, 17, 120.  
 Deventer, Mattijs Oskar van, 290.  
 DF<sub>FAIL</sub>, 46, 147.  
 Dfalse literals, 45.  
 Diagonals of a matrix, 24–25, 141–142.  
 Diagram of a trace, 84.  
 Díaz Cort, José Maria (= Josep), 51.  
 Dick, William Brisbane, 180.  
 Difficult instances of SAT, 5, 14, 26, 51, 55–59, 118–121, 153–154, 184, 190, 192, 197, 206, 280.  
 Digital tomography, 24–26, 115, 141–143, 167, 285.  
 Digraphs, 54, 108, 161, 162, 263, *see also* Blocking digraph, Dependency digraph, Implication digraph.  
 Dijkstra, Edsger Wybe, 22, 202, 204.  
 DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, 131.  
 DIMACS: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, inaugurated in 1990.  
 Ding, Jian (丁剑), 51.  
 Direct encoding, 98, 114, 171, 186, 264, 265, 281.  
 Direct sum of graphs or matrices, 162, 177.  
 Directed acyclic graphs of resolutions, 54–56, 70.  
 Directed graphs, *see* Digraphs.  
 Discarding the previous learned clause, 72, 156.  
 Discrepancy patterns, 114, 182.  
 Disjoint shortest paths, 276.  
 Disjunctive normal forms, 14, 115, 130, 195, 257.  
 Distance  $d(u, v)$  in a graph, 262.  
 Distinct literals, 2.  
 Division of traces, 85, 161, 250.

- DNF: Disjunctive normal form, 14, 115, 130, 195, 257.
- Dodgson, Charles Lutwidge, 129–130.
- Domino coverings, 110, 114, 115, 143, 177, 178.
- Don't-cares, 194, 280.
- Double clique hints, 100, 114, 171.
- Double coloring, 115, 135.
- Double lookahead, 45–46, 126, 131, 286.
- Double order, 214.
- Double truth, 45.
- Doubly linked lists, 28, 257, 259.
- Downhill resolution, 96, 166.
- Downhill transformations, 95.
- Doyle, Arthur Ignatius Conan, 72.
- DPLL (Davis, Putnam, Logemann, Loveland) algorithm, 32–33, 62.
- with lookahead, 38, 131.
- DT (double truth), 45.
- Dtrue literals, 45.
- Dual of a Boolean function, 130, 174.
- Dubois, Olivier, 131.
- Dudeney, Henry Ernest, 114, 263.
- Dufour, Mark, 37.
- Dull, Brutus Cyclops, 181.
- Durfee, William Pitt, square, 276.
- Dynamic storage allocation, 144.
- Dynamical system, discrete, 16.
- e*, as source of “random” data, 12, 193.
- Eager data structures, 30, 36, 156.
- Easy clauses, 149.
- Eaters in Life, 20, 139.
- Eén, Niklas Göran, v, 67, 96, 166, 203, 260, 268.
- Ehlers, Thorsten, 196.
- Eightfold symmetry, 138, 198.
- Elegance, 35–36, 196.
- Elimination of clauses, 167–168; *see also* Purgung unhelpful clauses.
- Elimination of variables, 60–61, 95–97, 101, 102, 129, 130, 154–155, 166–168, 173, 174, 256–257, 259–260, 270, 272.
- Embedded graphs, 169, 262.
- Empilements, 84, 161, 248.
- Empirical performance measurements, 122–124.
- Empty clause ( $\epsilon$ ), 3, 27, 185.
- Empty list, representation of, 33, 210.
- Empty partial assignment, 166.
- Empty set ( $\emptyset$ ), 185.
- Empty string ( $\epsilon$ ), 3, 85.
- Encoding into clauses, 6, 18, 97–105, 120, 134, 170, 179, 198, 202.
- ternary data, 100, 141, 179.
- Endomorphisms, 107–111, 177–178, 181, 281, 290.
- Equal sums, encoding of, 174.
- Equally spaced 1s, 4, 114, 135; *see also waerden*.
- Equivalence classes in trace theory, 84.
- Equivalence of Boolean functions, 178.
- Erdős, Pál (= Paul), 81, 107, 190, 281.
- discrepancy patterns, 114, 179, 182.
- Erp rules, 95–96, 166–168, 259.
- Evaluation of Boolean functions, 137, 178–179, 194.
- Even-length cycles, 277.
- Even-odd endomorphisms, 110, 177–178.
- Exact cover problems, vii, 2, 5–6, 28, 134, 183, 186, 219, 225, 257, 291.
- by pairs (perfect matchings), 109–110, *see also* Domino coverings.
- by triples (3D MATCHING), 134, 225, 290–291.
- fractional, 135–136.
- Exclusion clauses, 6, 21, 99, 114, 134, 149, 153, 238, 260, 289.
- Exclusive or, ternary, 136.
- Existential quantifiers, 60.
- Expander graphs, 58, 231.
- Exploitation stack, 259.
- Exploration phase of lookahead, 40, 43–44.
- Exponential time, 144.
- hypothesis, 288.
- Extended resolution, 60, 71, 133, 154, 168, 215.
- Extreme distribution, 87, 89, 163.
- factor\_fifo*( $m, n, z$ ), 10, 12, 114, 184, 192.
- factor\_lifo*( $m, n, z$ ), 10, 114, 184, 192.
- factor\_rand*( $m, n, z, s$ ), 10, 184.
- Factorization, 8–10, 136, 184, 192.
- of traces, 86, 162, 250.
- Failed literals, 97, 167, 175, 269.
- Fallacious reasoning, 16, 284.
- False hits, 258.
- False literals preferred, 31, 33, 67, 125–127, 286.
- Fanout gates, 10–14, 136.
- Fat clauses, 58.
- Fault testing, 10–14, 114, 126, 136–137, 167, 260.
- Feedback mechanism, 46, 104.
- Fermat, Pierre de, 10.
- Fernandez de la Vega, Wenceslas, 52.
- Fibonacci, Leonardo, of Pisa (= Leonardo filio Bonacii Pisano), numbers, 160, 215, 254.
- ruler function, 246.
- Fichte, Johannes Klaus, 262.
- Field of a variable, 91, 165.
- FIFO: first in, first out, 10.
- Finite-state automata, 175.
- First in, first out, 10.
- First moment principle, 53, 148, 150.
- First order logic, 59, 130.
- Fischetti, Matteo, 206.
- Fixed point of endomorphisms, 177.

- Fixed values of literals, 37, 42–46.  
 FKG inequality, 89.  
 Flag bits, 235.  
 Flammenkamp, Achim, 198.  
 Flexibility coefficients, 91.  
 Flickering state variables, 141.  
 Flipflops in Life, 138, 143.  
 Floating point arithmetic, 91–92, 217, 239.  
   overflow, 67.  
 Floor tiling, 115, 143, 199.  
 Flower snarks, 69, 147, 153, 157.  
 Flushing literals and restarting, 68, 75–77,  
   124, 132, 157, 158, 169, 234, 246.  
 Foata, Dominique Cyprien, 83, 86, 163.  
 Focus of attention, 41, 67, 91, 132.  
 Foe, Daniel (= Daniel Defoe), vii.  
 Footprint heuristic, 279.  
 Forced literals, 45.  
 Forcing clause, 62, *see* Unit propagation.  
 Forcing representations, 104–105, 174,  
   175, 274.  
 Forests, 43, 87, 163.  
 Forgetting clauses, 168, 169, *see* Purging  
   unhelpful clauses.  
 Four bit protocol, 115.  
 Four Color Theorem, 7.  
 Fourfold symmetry, 138.  
 Fractional coloring number, 135.  
 Fractional exact cover, 135–136.  
 Frances, Moti (מורי פרנסס), 285.  
 Franco, John Vincent, 131, 148, 225, 274.  
 Free literals and free variables, 38,  
   66, 165–166.  
 Freeman, Jon William, 131.  
 Friedgut, Ehud (אהוד פרידגוט), 51.  
 Frontier ( $\partial_{in}$ ), 180, 188.  
 Frost, Daniel Hunter, 67.  
*fsnark* clauses, 69, 71, 114, 147–148,  
   153, 157.  
 Full adders, 9, 136, 179, 192, 268, 278.  
   modified, 114, 280.  
 Full runs, 73, 158, 235.  
 Furtlehner, Cyril, 91.
- $G\mu$ : Gigamems = billions of memory  
   accesses, 97, 118.  
   per minute, 289.  
 Gadgets, 134, 183.  
 Gallager, Robert Gray, 95.  
 Game of Life, 17–20, 97, 114, 137–139,  
   143, 167.  
 Garden of Eden, 139.  
 Gardner, Martin, 7, 19, 181, 188, 283.  
 Gates, 10–13, 101–103, 121, 136.  
 GB\_GATES, 13–14.  
 Gebauer, Heidi Maria, 224.  
 Geek art, 116–117.  
 Generalization of resolution, 226.
- Generating functions, 85, 89–90, 151,  
   158–159, 164, 188, 194, 219, 222,  
   230, 284.  
   exponential, 162.  
 Generic graph, 174.  
 Gent, Ian Philip, v, 265, 283.  
 Gentzen, Gerhard Karl Erich, 59.  
 Geometric distribution, 244, 262.  
 Georges, John Pericles, 192.  
 Gerdes, Paulus Pierre Joseph, 205.  
 Gessel, Ira Martin, 248.  
 Giant strong component, 52.  
 Gigamem ( $G\mu$ ): One billion memory  
   accesses, 35, 39, 97, 188.  
   per minute, 289.  
 Ginsberg, Matthew Leigh, 113.  
 Gipatsi patterns, 141.  
 Girth of a graph, 176, 290.  
 Given literals ( $F|l$  or  $F|L$ ), 27, 96, 143,  
   157; *see also* Unit conditioning.  
 Gliders in Life, 19, 138–139, 197, 201.  
   symmetry of, 200.  
 Global ordering, 284.  
 Glucose measure, *see* Literal block distance.  
 Goerdts, Andreas, 52.  
 Goldberg, Allen Terry, 131, 225.  
 Goldberg, Eugene Isaacovich (Гольдберг,  
   Евгений Исаакович), 70, 132.  
 Goldman, Jay Robert, 250.  
 Goldsmith, Oliver, 293.  
 Golomb, Solomon Wolf, 265.  
 González-Arce, Teófilo Francisco, 268.  
 Gosper, Ralph William, Jr., 20, 198.  
 Goultiaeva, Alexandra Borisovna  
   (Гультяева, Александра Борисовна),  
   73.  
 Grabarchuk, Peter Serhiyevich (Грабарчук,  
   Петро Сергійович), 263.  
 Grabarchuk, Serhiy Alexeevich (Грабарчук,  
   Сергій Олексійович), 263.  
 Grabarchuk, Serhiy Serhiyevich (Грабарчук,  
   Сергій Сергійович), 263.  
 Graham, Ronald Lewis (葛立恒), 185, 283.  
 Graph-based axioms, 59, 154, 178, 290.  
 Graph coloring problems, *see* Coloring  
   a graph.  
 Graph embedding, 169, 262.  
 Graph layout, 116–118.  
 Graph quenching, 114, 179–180, 281.  
 Gray, Frank, codes, 201, 282.  
 Greedy algorithms, 80, 136, 172.  
 Greenbaum, Steven Fine, 102.  
 Grid graphs, 110, 136, 151, 162–163.  
   list coloring of, 151.  
 Grid patterns, 17–20, 24–26, 137–139, 142.  
   rotated 45°, 141–142.  
 Griggs, Jerrold Robinson, 192.  
 Grinbergs, Emanuels Donats Frīdrihs  
   Jānis, 218.



- Gritzman, Peter, 206.  
 Grötschel, Martin, 264.  
 Gu, Jun (顧鈞), 77.  
 Guéret-Jussien, Christelle, 267.  
 Guilherme De Carvalho Resende, Mauricio, 16.  
 GUROBI system, 289.  
 Guy, Richard Kenneth, 17, 19, 107.  
 Gwynne, Matthew Simon, 105, 270, 273.
- Hackers, 199.  
 Haken, Armin, 57, 58.  
 Halevi, Shai (שי הלוי), 289.  
 Half adders, 9, 192, 268.  
 Halting problem, 130.  
 Hamadi, Youssef (حمادي يوسف), 236.  
 Hamilton, William Rowan, cycles, 169.  
 paths, 184.  
 Hamming, Richard Wesley, distance, 285.  
 Han, Hyojung (한효정), 236.  
 Handwaving, 89.  
 Hanzelet, *see* Appier dit Hanzelet.  
 Hard clauses, 168.  
 Hard sudoku, 183.  
 Hartman, Christiaan, 202.  
 Haven, G Neil, 131.  
 Head of the list, 28.  
 Header elements, 225.  
 HEAP array, 67, 158, 233–235, 240.  
 Heap data structure, 67, 214.  
 insertion and deletion from, 234.  
 Heaps of pieces, 83.  
 Height of a literal, 214.  
 Height of a trace, 85.  
 Heilmann, Ole Jan, 251.  
 Helpful rounds, 169.  
 Heule, Marienus (= Marijn) Johannes Hendrikus, iv, v, 37, 40, 46, 71, 75, 97–98, 104, 129, 134, 147, 182, 186, 202, 213, 239, 260, 261, 263.  
 Heuristic scores, 90–95.  
 for clauses, 72–74, 125–127, 158, 239, 286.  
 for variables, 40–44, 61, 67, 80, 126, 145–147, 214.  
 Hidden weighted bit function, 173.  
 Hierarchy of hardness, 176, 178.  
 Hilton, Anthony John William, 191.  
 Hint clauses, 100, 114, 171.  
 Hirsch, Edward Alekseevich (Гирш, Эдуард Алексеевич), 215.  
 Historical notes, 32, 59–60, 105, 129–133, 231.  
 Hollow mazes, 290.  
 Holmes, Thomas Sherlock Scott, 72.  
 Homomorphic embedding, 169, 262.  
 Honest representations, 105, 270.  
 Hoory, Shlomo (שלמה חורי), 224.  
 Hoos, Holger Hendrik, v, 125–127, 133, 160.  
 Horn, Alfred, clauses, 132, 166, 176, 216, 263.  
 core of, 174, 216.  
 renamed, 176, 263.  
 Horsley, Daniel James, 274.  
 Horton, Robert Elmer, numbers, 152.  
 Hsiang, Jieh (項傑), 129.  
 Hume, David, 1.  
 Hunt, Warren Alva, Jr., 71, 239.  
 Hutter, Frank Roman, 125, 133.  
 Hypergraph 2-colorability, 185.  
 Hyperresolution, 56, 228, 257.
- Idle Year solitaire, 180.  
 If-then-else operation ( $u? v: w$ ), 81, 102, 152, 173, 219.  
 ILS: Iterated Local search, 125.  
 Impagliazzo, Russell Graham, 55, 288.  
 Implication digraph, 52, 144.  
*in situ* deletion, 156.  
 Inclusion and exclusion, 221–222, 256.  
 Inconsistent clauses, *see* Unsatisfiable formulas.  
 Indecomposable matrices, 177.  
 Independent vertices, 7, 147.  
 Induced graph, 262.  
 Induction proofs by machine, 24, 203.  
 Infinite loop, 244.  
 Initial guess for literals, 31, 33, 66, 125–127, 286.  
 Initial state  $X_0$ , 16–17, 21, 24, 140, 202.  
 Inprocessing, 95, 168.  
 Input and output, 120.  
 Input states, 175.  
 Insertion into a heap, 234.  
 Integer programming, 26, 184, 206, 285.  
 Interactive SAT solving, 142.  
 Interlaced roots of polynomials, 163.  
 Internet, ii, iii, v, 118.  
 Intersection graphs, 84, 161.  
 Interval graphs, 87, 163.  
 Intervals, cardinality constrained to, 100, 190, 280.  
 Invariant assertions, 23–24, 43, 115, 140, 203, 216, 217, 255, 261.  
 Inverse permutations, 112, 265.  
 Inversions of a permutation, 213.  
 Involution polynomial of a set, 163.  
 Involutions, signed, 112–113, 180, 277–278.  
 INX array, 38, 211, 286.  
 IP: Integer programming, 26, 184, 206, 285.  
 Irredundant CNF, 257.  
 Irreflexive relation, 56.  
 Irving, Robert Wylie, 151.  
 Isaacs, Rufus Philip, 218.  
 Isolated vertices, 262.  
 IST array, 38.  
 ISTACK array, 38, 145.  
 ISTAMP counter, 37–38, 46, 145.



- Levels of values, 62–66, 156, 233.
- Levesque, Hector Joseph, 50.
- Levine, Eugene, 275.
- Lewis, Jerome Luther, 275.
- Lex-leader: The lexicographically smallest element, 111, 283.
- Lexicographic order, 4, 25, 26, 30, 101, 105, 107, 109, 111–113, 115, 197, 282–283. encoded in clauses, 101, 173, 174.
- Lexicographic row/column symmetry, 106–107, 177, 181, 274.
- Lexicographically smallest (or largest) solution, 25–26, 111–113, 142, 157, 282, 283.
- Lexicographically smallest traces, 84, 161, 162, 250.
- Leyton-Brown, Kevin Eric, 125, 133.
- Li, Chu Min (李初民), 131.
- Li, Wei (李未), 149.
- Lieb, Elliott Hershel, 251.
- Life, Game of, 17–20, 97, 114, 137–139, 143, 167.
- Light speed in Life, 139.
- Line graph of a graph, 147, 249.
- Linear equations, 26, 231.
- Linear extensions, *see* Topological sorting.
- Linear inequalities, 184. encoding of, 100–101, 172, 173.
- Linear programming relaxation, 26.
- Lines, abstracted, 106.
- Links, dancing, 5, 121, 134, 208, 288.
- Lisitsa, Alexei Petrovich (Лісіца, Аляксей Пятровіч), 281.
- List coloring of graphs, 135, 151.
- List merging, 231, 258.
- Literal block distance, 72, 74, 158.
- Literals, 2, 111. flushing, 76. internal representation, 28, 37, 66, 208, 209, 242, 257.
- Litman, Ami (עמי ליטמן), 285.
- Livelock, 22–23.
- Llunell, Albert Oliveras i, 267.
- LNCS: Lecture Notes in Computer Science*, inaugurated in 1973.
- Local Lemma, 81–90, 133, 151, 160–165.
- Log encodings, 98, 114–115, 173.
- Logemann, George Wahl, 31–32, 130, 298.
- Longest simple path, 23, 203.
- Lonlac, Jerry, 289.
- Look-back, *see* Backjumping.
- Lookahead autarky clauses, *see* Black and blue principle.
- Lookahead forest, 42–44, 145–147, 168.
- Lookahead solvers, 38–46, 55, 97, 129, 131, 176. combined with CDCL solvers, 129. compared to CDCL solvers, 98–100, 118–121, 182, 290.
- Loopless shadows, 184.
- Lopsidedness graphs, 82, 83, 160, 164, 165, 185, 224.
- Lovász, László, 81, 82, 185, 191.
- Loveland, Donald William, 32, 130, 298.
- Lower semimodular lattices, 255–256.
- Loyd, Samuel, 263.
- Luby, Michael George, 80, 159.
- Luks, Eugene Michael, 113.
- $M\mu$ : Megamems = millions of memory accesses, 69, 98.
- Maaren, Hans van, 37, 46.
- MacColl (= McColl), Hugh, 227.
- MacMahon, Percy Alexander, Master Theorem, 250, 251.
- Mader, Adolf, 275.
- Madigan, Conor Francis, 132.
- Magic, 193.
- Magic sequences, 285.
- Magnetic tape, 32.
- Makespan, 172–173.
- Mallach, Sven, 289.
- Malik, Sharad (शरद मलिक), 132.
- Maneva, Elitza Nikolaeva (Манева, Елица Николаева), 166, 256.
- Mapping three items into two-bit codes, 179.
- march** solver, 40, 216.
- Marek, Victor Wiktor, 216.
- Markov (= Markoff), Andrei Andreevich (Марков, Андрей Андреевич), the elder, inequality, 158, 241.
- Markov, Igor Leonidovich (Марков, Игор Леонидович), 112, 281, 284.
- Markström, Klas Jonas, 290.
- Marques da Silva (= Marques-Silva), João Paulo, 132.
- Marriage theorem, 224.
- Martin, Alexander, 264.
- Matching polynomial of a graph, 249.
- Matchings in a graph: Sets of disjoint edges, 150, 230, 249. perfect, 109–110, 177.
- Mathews, Edwin Lee (41), 67.
- Matrix multiplication, 260.
- Mauro, David Whittlesey, 192.
- Maximal elements, 56, 62, 97, 115, 153, 157, 167.
- Maximal planar graphs, 186.
- Maximum independent sets, 87, 136, 187, 188.
- Maximum number of 1s, 106–109, 135, 136, 177.
- MAXSAT lower bound, 184.
- “Maybe” state, 20.
- Mazurkiewicz, Antoni Wiesław, 83.
- McColl (= MacColl), Hugh, 227.
- McGregor, William Charles, 7, 188. graphs, 7–8, 114–115, 134, 135, 188.
- Mean running time, 120.

- Median operation, 9, 136, 179.  
 Median running times, 99, 120–124, 127.  
 Megamem ( $M\mu$ ): One million memory accesses, 201.  
 Méjean, Henri-Michel, 226.  
 Mellin, Robert Hjalmar, transforms, 151.  
 Mem ( $\mu$ ): One 64-bit memory access, 34.  
 MEM array, 66, 68, 124, 156.  
 Memo cache, 60, 233.  
 Memoization technique, 233.  
 Memoryless property, 244.  
 Menagerie, 116–117.  
 Merge networks, 266.  
 Merging lists, 231, 258.  
 Mertens, Stephan, 51.  
 Message passing, 90–95, 165–166.  
 Method I, 61.  
 Method IA, 61, 154.  
 Method of Trees, 129.  
 Methuselah solitaire, 282.  
 Methuselahs in Life, 19.  
 Mézard, Marc Jean Marcel, 51, 90, 91, 95.  
 Midpoint inequalities, 266.  
 Mijnders, Sid, 213.  
 Mills, Burton Everett, 130.  
 Minesweeper, 142–143.  
 Minimally unsatisfiable clauses, 150, 153.  
 Minimum covers, 193.  
 MiniSAT, 67.  
 Minterms, 179.  
 Mitchell, David Geoffrey, 50.  
 Mitters, 121, 182.  
 Mitsche, Dieter Wilhelm, 51.  
 Mixed metaphors, 76.  
 Mixed-radix number systems, 268.  
 MMIX computer, ii, 158.  
 Mobile Life paths, 18–19, 138–139.  
   flipflops, 138.  
 Möbius, Augustus Ferdinand, functions, 86.  
   series, 86, 160, 162–163, 165, 247, 249.  
 Mod 3 addition, 114, 179.  
 Mod 3 parity, 179.  
 Mod 4 parity, 179.  
 Model checking, 16–17, 137–141, 179–180.  
 Model RB, 149.  
 Modified full adders, 114, 280.  
 Modular lattices, 255.  
 none ( $-1$ ), 242.  
 Monien, Burkhard, 215.  
 Monkey wrench principle, 113, 181.  
 Monotone functions, 163.  
   Boolean, 137, 281.  
 Monotonic clauses, 5, 133, 157.  
 Monotonic paths, 108, 276.  
 Montanari, Andrea, 95.  
 Monus operation ( $x \dot{-} y = \max\{0, x-y\}$ ),  
   92, 247, 268.  
 Moore, Edward Forrest, 202.  
 Morehead, Albert Hodges, 282.  
 Morel, Henri, 226.  
 Moser, Robin Alexander, 82, 254.  
 Moskewicz, Matthew Walter, 132.  
 Mossel, Elchanan (אלחנן מוסל), 166.  
 Mott-Smith, Geoffrey Arthur, 282.  
 Move codes, 29–31, 34, 144, 145, 155, 210.  
 MPR: Mathematical Preliminaries Redux, v.  
 Mueller, Rolf Karl, 60, 130.  
 Müller, Mike, 196.  
 Multicommodity flows, 170.  
 Multigraphs, 231.  
 Multilinear function, 86.  
 Multiplication of binary numbers, 8–9,  
   12–14, 114, 136, 173.  
 Multiplication of traces, 85, 161.  
 Multisets, 3, 214, 224, 250.  
 Multivalued graph colorings, 99.  
 Mutilated chessboard, 110, 114,  
   177–178, 286.  
 Mutual exclusion protocols, 20–24,  
   115, 139–141.  
 Mutzbauer, Otto, 275.  
 Mux operation ( $u? v: w$ ), 81, 102,  
   152, 173, 219.  
 Mysterians, 290.  
 $n$ -cube, 79, 136, 148, 184.  
 n.f.: not falsified, 271.  
 $n$  queens problem, 25, 115, 171, 282.  
 NAND operation, 60.  
 Napier, John, Laird of Merchiston, 9, 173.  
 Near truth, 37–39.  
 Necessary assignments, 45, 146.  
 Negated auxiliary variables, 105.  
 Negative  $k$ -clauses, 157.  
 Negative literals, 2, 132, 153.  
 Nesting phase of lookahead, 40, 42–43,  
   145–147.  
 Newbie variables, 41.  
 Newton, Isaac, method for rootfinding,  
   216–217.  
 Niborski, Rodolfo, 190.  
 Niemelä, Ilkka Niilo Fredrik, 105.  
 Nieuwenhuis, Robert Lukas Mario, 267.  
 Nightingale, Peter William, 265.  
 No-player game, 17.  
 Nodes of a search tree, 34–35, 69, 124.  
 Noels, Alain, 202.  
 Noisy data, 181.  
 Nonattacking queens, 25, 115, 171, 282.  
 Nonaveraging sets, 114, 135.  
 Nonchromatic rectangles, 176–177.  
 Nonchronological backtracking, *see*  
   Backjumping.  
 Noncommutative variables, 162.  
 Nonconstructive proofs, 57, 58, 81, 202.  
 Nondeterministic finite-state automata, 175.  
 Nondeterministic polynomial time, 131.  
 Nondeterministic processes, 20, 141, 182.  
 Nonintersecting paths, 170.

- Nonnegative coefficients, 164.  
 Nonprimary columns, 186.  
 Nonterminal symbols, 175.  
 Normal chains, 278.  
 Normal functions, 279.  
 Not-all-equal SAT, 185.  
 Notational conventions, vi.  
 $\partial S$  (boundary set), 58, 154, 180, 188.  
 $C' \diamond C''$  (resolvent), 54, 152.  
 $C \subseteq C'$  (subsumption), 61, 152.  
 $F \mid l$  ( $F$  given  $l$ ), 27, 96, 291.  
 $F \mid L$  ( $F$  given  $L$ ), 27, 103, 157.  
 $F \vdash C$  ( $F$  implies  $C$ ), 59, 152, 153.  
 $F \vdash_1 \epsilon$ , 70, 157, 175.  
 $F \vdash_1 l$ , 103–104, 176.  
 $F \vdash_k \epsilon$ ,  
 $F \vdash_k l$ , 175–176.  
 $G \oplus H$  (direct sum), 162, 177.  
 $|l|$  (a literal's variable), 2.  
 $\pm v$  ( $v$  or  $\bar{v}$ ), 2.  
 $\langle xyz \rangle$  (median), 9.  
 $x \& y$  (bitwise AND), 28, 29, 31, 37, 38, 66,  
 68, 76, 81, 196, 209–211, 220, 241.  
 $x \mid y$  (bitwise OR), 43, 196, 241, 258–259.  
 $x \oplus y$  (bitwise XOR), 28, 137, 196,  
 208, 220, 241.  
 $x \dot{-} y$  (monus), 92, 247, 268.  
 $x? y: z$  (if-then-else), 81, 102, 152,  
 173, 219.  
 $w(\alpha)$ , 57.  
 $w(\alpha \vdash \epsilon)$ , 57.  
 $\|\alpha \vdash C\|$ , 57.  
 $\mu(C)$ , 59, 153.  
 Novikov, Yakov Andreevich (Новиков,  
 Яков Андреевич), 70, 132.  
 Nowakowski, Richard Joseph, 107, 275.  
 NP-complete problems, 1, 3, 27, 87,  
 130–131, 134, 142, 151, 181–183, 207,  
 268, *see also* CoNP-complete problems.  
 NT (near truth), 37–39.  
 Null clause ( $\epsilon$ ), 3, 27, 185, 291.  
 Null list, representation of, 33, 210.  
 Null partial assignment, 166.  
 Null set ( $\emptyset$ ), 185.  
 Null string ( $\epsilon$ ), 85.  
 Nullary clause ( $\epsilon$ ), 3, 27, 185, 291.  
 Number theory, 14, 137, 192.  
  
 Occurrence threshold of a graph, 160.  
 Odd permutations, 218.  
 Odd-even merge network, 266.  
 Odd-even transposition sort, 263.  
 Oliveras i Lluell, Albert, 267.  
 On-the-fly subsumptions, 124, 156.  
 One-in-three satisfiability, 183.  
 One-per-clause satisfiability, 183.  
 Open shop scheduling problems, 115,  
 172–173.  
 OR operation, 9, 10, 13, 258.  
 bitwise ( $x \mid y$ ), 43, 196, 241, 258–259.  
  
 Orbits of a permutation group, 108, 277.  
 Order encoding, 98–101, 114, 120, 170–173,  
 190, 268, 281.  
 Order of a permutation, 111.  
 Organ-pipe permutations, 171.  
 Oriented cycle detection, 260.  
 Oriented trees, 108.  
 Orphan patterns in Life, 139.  
 Orponen, Olli Pekka, 80.  
 Oscillators in Life, 19, 138–139.  
 Output states, 175.  
 OVAL array, 74, 125, 237, 240.  
 Overfitting, 182.  
 Overflow in arithmetic, 67, 240.  
 Oxusoff, Laurent, 215.  
  
 $\varnothing$  (tautology, the always-true clause), 3, 58,  
 60, 152, 180, 215, 226–228, 258.  
 $P = NP$ , 1.  
 Palindromes, 136.  
 Panagiotou, Konstantinos (Παναγιώτου,  
 Κωνσταντίνος), 221.  
 Papadimitriou, Christos Harilaos  
 (Παπαδημητρίου, Χρίστος Χαριλάου),  
 77, 240, 241.  
 Parallel multiplication circuits, 12–14, 137.  
 Parallel processes, 20, 24, 121, 128–129.  
 Parameters, tuning of, 80–81, 93–94,  
 124–128.  
 ParamILS, 125, 286–287.  
 Parity-related clauses, 153–154, 172,  
 178, 231–232, 290.  
 Partial assignments, 30, 61, 62, 165, 176.  
 Partial backtracking, 208.  
 Partial latin square construction, 151.  
 Partial orderings, 56, 85, 115, 248.  
 of dimension  $\leq 2$ , 213.  
 Participants, 41, 44, 145.  
 Path detection, 169.  
 Path graphs  $P_n$ , 84, 160, 253.  
 Patience, *see* Solitaire games.  
 Paturi, Ramamohan (ராமமோஹ் பாதூர்), 288.  
 Paul, Jerome Larson, 5.  
 Paull, Marvin Cohen, 148.  
 $PC_k$ , 176, 178.  
 Pearl, Judea (יהודה פּרל), 95.  
 Pegden, Wesley Alden, v, 164, 253.  
 Peierls, Rudolf Ernst, 95.  
 Peres, Yuval (יובל פרס), 221.  
 Pérez Giménez, Xavier, 51.  
 Perfect matchings in a graph, 109–110, 177.  
 Permanent of a matrix, 183, 251.  
 Permutation polynomial of a set, 163.  
 Permutation posets, 213.  
 Permutations, 105, 265.  
 signed, *see* Signed permutations.  
 weighted, 163.  
 Permuting variables and/or complementing  
 them, *see* Signed permutations.

- Peterson, Gary Lynn, 23, 115, 140, 204.  
 Petrie, Karen Elizabeth Jefferson, 283.  
 Phase saving, 67, 75.  
 Phase transitions, 50–52, 149–150.  
 Phi ( $\phi$ ), 146, 147, 160, 251.  
 Phoenix in Life, 198, 207.  
 Pi ( $\pi$ ), as source of “random” data, 12, 46, 108, 115, 147, 184, 193, 286;  
     *see also* Pi function.  
 Pi function, 102, 174.  
 Pieces, in trace theory, 84–87.  
 Pigeonhole principle, 57.  
     clauses for, 57–59, 105–106, 113, 153, 176, 181, 186, 265.  
 Pikhurko, Oleg Bohdan (Піхурко, Олег Богданович), 285.  
 Pile sums, 151.  
 Pincusians, 133.  
 Pipatsrisawat, Thammanit (= Knot)  
     (พิพัฒน์ศรีสวัสดิ์, ทมมนิต์ (= นอต)), 67, 262.  
 Pixel images, 200; *see also* Grid patterns.  
 Plaisted, David Alan, 102.  
 Planning, 132.  
 Playing cards, 114, 180, 282.  
 Points, abstracted, 106.  
 Poison cards, 282.  
 Poisson, Siméon Denis, probability, 225.  
 Polarities, 3, 67, 76, 207, 237.  
 Pólya, György (= George), theorem, 284.  
 Polynomials in trace theory, 85.  
 Population in Life, 19.  
 Portfolio solvers, 133.  
 Posets, *see* Partial orderings.  
 Positive autarkies, 146.  
 Positive  $j$ -clauses, 157.  
 Positive literals, 2, 132, 146.  
 Posthoff, Christian, 275.  
 Postorder, 42–43, 214.  
 Postprocessor, 96.  
 Preclusion clauses, 99, 171, 186.  
 Preorder, 42–43, 214.  
 Preprocessing of clauses, 95–97, 103, 166–168, 182, 268, 272, 278.  
 Preselection phase of lookahead, 40–42, 147.  
 Prestwich, Steven David, 264.  
 Primary variables, 104, 105.  
 Prime clauses, 174, 270, 273.  
 Prime implicants, 281.  
 Pringsheim, Alfred Israel, 88, 164.  
 Prins, Jan Fokko, 267.  
 Probabilistic method, 81.  
 Probability of satisfiability, 47–54.  
 $prod(m, n)$ , 12–14, 114, 137.  
 Production rules, 175.  
 Profile of a search tree, 151.  
 Progress, display of, 30, 145, 155.  
 Progress saving, 67, *see* Phase saving.  
 Projection of a path, 184.  
 Projective plane, 274.  
 Propagation,  $k$ th order, 175–176, 273.  
 Propagation completeness ( $UC_1$ ), 176.  
 Proper ancestors, 164.  
 Proto truth, 37, 42.  
 Prover–Delayer game, 55–56, 152–153.  
 PSATO solver, 159.  
 Pseudo-Boolean constraints, *see* Threshold functions.  
 PT (proto truth), 37, 42.  
 Pudlák, Pavel, 55.  
 Puget, Jean-François, 113.  
 Purdom, Paul Walton, Jr., 30, 32, 131, 151, 226.  
 Pure cycles, 140.  
 Pure literals, 29, 31, 32, 34, 44, 60, 130, 135, 146, 152, 208, 215, 227, 256, 259, 268, 269, 275.  
 Purging unhelpful clauses, 68, 71–75, 124, 132, 157, 158, 168, 182, 184, 235.  
     threshold for, 74, 125, 127.  
 Putnam, Hilary, 9, 32, 130, 298.  
 Pyramids in trace theory, 87, 162.  
 q.s.: quite surely, 149, 153, 169.  
 QDD: A quasi-BDD, 188.  
 Quad-free matrices, 106–107, 113, 176–177, 274, 284.  
 Quantified formulas, 60, 154.  
 Queen graphs, 25, 99–100, 114–115, 120, 171, 180, 181.  
 Quenchable graphs, 179–180, 281.  
 Quick, Jonathan Horatio, 181.  
 Quilt patterns, 198.  
 Quimper, Claude-Guy, 272.  
 Quine, Willard Van Orman, 129, 130.  
 $\mathcal{R}(G)$  (Local Lemma bounds), 82, 87–90, 160, 163–165.  
 Radio colorings, 136.  
 Radix- $d$  representation, 173.  
 Rado, Richard, 191.  
 Ramakrishnan, Kajamalai Gopaldaswamy, 16.  
*raman* graphs, 231.  
 Ramani, Arathi (ஆர்த்தி ரமணி), 112, 281, 284.  
 Ramanujan Iyengar, Srinivasa (ஸ்ரீனிவாஸ ராமானுஜன் ஐயங்கார்), graphs, 154;  
     *see also* *raman* graphs.  
 Ramos, Antonio, 75.  
 Ramsey, Frank Plumpton, theorem, 81.  
*rand*, 39–40, 46, 50, 115, 147, 182.  
 Random bits, biased, 12, 241.  
 Random choices, 12.  
 Random decision variables, 125–127, 155, 286.  
 Random graphs, 81.  
 Random permutations, 233.

- Random satisfiability problems, 47–54, 91, 151.  
 2SAT, 51–54, 149.  
 3SAT, 39–40, 46–51, 59–60, 80, 93–94, 147–149, 153, 242.  
 $k$ SAT, 49–51, 146, 148.
- Random walks, 77–81, 125, 243.
- Random words, 149.
- Randomized methods, 77, 129, 182, 210.
- RANGE scores, 74, 125–127, 158, 239.
- RAT, *see* Resolution certifiable clauses.
- Rauzy, Antoine Bertrand, 131, 215.
- Reachability in a graph, 169.
- Ready list, 32.
- Real roots of polynomials, 163, 249.
- Real truth, 37–39.
- Reasons, 62, 72, 157, 165, 233.
- Rebooting, 22.
- Reckhow, Robert Allen, 61.
- Recurrence relations, 151, 177, 189, 215, 243.
- Recursive procedures, 27, 130, 172, 186, 233.
- Recycling of clauses, 66, 124.
- Reduction of clauses, 27, 143; *see also* Simplification of clauses.
- Redundant clauses, 257.
- Redundant literals, 65, 155–156, 232, 234.
- Redundant representations, 171.
- Reed, Bruce Alan, 52.
- Reflected ternary code, 290.
- Reflection symmetries, 112, 138, 156.
- Refutation chains, 57, 227.
- Refutation trees, 152.
- Refutations, 54–60, 70, 110, 152; *see also* Certificates of unsatisfiability.
- Regular expressions, 174–175.
- Regular resolution, 55, 152, 231.
- Reinforcement messages, 91–93.
- Reliability polynomials, 83.
- Reluctant doubling, 77, 80–81, 159–160.
- Reluctant Fibonacci sequence, 160.
- Renamed Horn clauses, 176, 263.
- Repeated clauses, 49.
- Replacement principle, 96.
- Representation of Boolean functions, 104, *see* Encoding into clauses.
- Representing three states with two bits, 179.
- Rescaled activity scores, 67.
- Resende, *see* Guilherme De Carvalho Resende.
- Resizing of data structures, 210.
- Resolution certifiable clauses, 261.
- Resolution chains, 57–59, 152, 153, 227.
- Resolution of clauses, 54–65, 70, 101, 129, 130, 144, 167, 185, 215, 224, 256.  
 implementation of, 167.
- Resolution refutations, 54–60, 70, 110, 152; *see also* Certificates of unsatisfiability.  
 extended, 60, 71, 133, 154, 168, 215.  
 regular, 55, 152, 231.  
 treelike, 55–56, 152–153.
- Resolvable clauses, 164.
- Resolvent ( $C' \diamond C''$ ), 54, 130, 152.
- Restarting, 80–81, 95, 125, 132.  
 and flushing literals, 68, 75–77, 124, 132, 157, 158, 169, 234, 246.
- Restricted growth strings, 179.
- Restricted pigeonhole principle, 58.
- Reusing the trail, 75.
- Reverse unit propagation, 71.
- Revolving door Gray code, 282.
- Reynaud, Gérard, 226.
- Richards, Keith, 1.
- Rickard, John, 290.
- Right division of traces, 85, 161.
- Right factor of a trace, 161.
- Riis, Søren Møller, 110.
- Ripoff, Robert Iosifovich (Рипов, Роберт Иосифович), 7.
- Rivest, Ronald Linn, clauses, 4, 55, 70, 134, 144, 182.
- Roberts, Fred Stephen, 136.
- Robertson, Aaron Jon, 185.
- Robinson, Gilbert de Beauregard, 275.
- Robinson, John Alan, 59, 96, 227.
- Rodríguez Carbonell, Enric, 267.
- Rokicki, Tomas Gerhard, 200.
- Rooij, Iris van, 207.
- Rook paths, 206.
- Rookwise connected cells, 170.
- Ross, Kenneth Andrew, 282.
- Rotational symmetry, 138, 202, 275.
- Rotors in Life, 138.
- Roussel, Olivier Michel Joseph, 132, 272.
- Routing, disjoint, 170.
- Row sums, 151.
- Roy, Amitabha (অমিতাভ রায়), 113.
- RT (real truth), 37–39, 43.
- Ruler doubling, 160.
- Ruler of Fibonacci, 246.
- Running times, 89–90.  
 comparison of, 34–35, 39, 69, 97–100, 105–107, 110, 112, 118–128, 182, 184, 218, 237, 264, 281, 290.  
 mean versus median, 120.  
 worst case, 144, 146, 154.
- Runs of 1s, 26, 143, 175.
- $s$ -chains, 52–53, 149.
- $s$ -snares, 53, 149.
- $S_1(y_1, \dots, y_p)$ , 6.
- $S_k(m, n)$ , 50–54.
- $S_{k,n}$ , 49–51, 148, 149.
- $S_{\leq r}(x_1, \dots, x_n)$  and  $S_{\geq r}(x_1, \dots, x_n)$ , 8,  
*see* Cardinality constraints.
- Saddle point method, 226.
- Sahni, Sartaj Kumar (सरताज कुमार साहनी), 268.
- Saïs, Lakhdar (⊙•⊙ ••×E•O, سايس للخضر), 236, 289.

- Sakallah, Karem Ahmad (أحمد ساقالله كارم), 112, 132, 281, 284.
- Salhi, Yakoub (يعقوب صالح), 289.
- Sampling with and without replacement, 49–50, 132, 226.
- Samson, Edward Walter, 60, 130.
- SAT: The satisfiability problem, 3.
- SAT solvers, 1, 131–133.
- SATexamples.tgz, 118.
- Satisfiable formulas, 1.  
variability in performance, 35, 120–121, 128, 287.
- Satisfiability, 1–184.  
history, 32, 59–60, 105, 129–133.  
thresholds for, 50–54, 91, 148–149, 221.
- Satisfiability-preserving transformations, 107–113.
- Satisfying assignments, 1, 30, 143–144, 166, 214, 219.
- SATzilla solver, 132–133.
- Schaefer, Thomas Jerome, 289.
- Schensted, Craig Eugene (= Ea Ea), 275.
- Schlipf, John Stewart, 274.
- Schmitt, John Roger, 285.
- Schoenfeld, Jon Ellis, 192.
- Schöning, Uwe, 78.
- Schrag, Robert Carl, 132.
- Schroepel, Richard Crabtree, 197.
- Schwarzkopf, Bernd, 282.
- Scott, Alexander David, 224, 251, 252.
- Scott, Allan Edward Jolicoeur, 207.
- Scott, Sidney Harbron, 191.
- Scoville, Richard Arthur, 162.
- Search trees, 28–29, 32–34, 124, 152.  
expected size, 151–152.  
optimum, 144.
- Second moment principle, 54, 221, 222.
- Seitz, Simo Sakari, 80.
- Self-subsumption, 96, 167, 168, 257.
- Selman, Bart, 50, 79, 132.
- Semimodular lattices, 255–256.
- Sentinel values, 259.
- Sequential consistency, 24.
- Sequential lists, 36–37, 144.
- Sequents, 59.
- Serial correlation coefficients, 143.
- Set partitions, 220.
- SGB, *see* Stanford GraphBase.
- Shadows of paths, 184.
- Shandy, Tristram, iii.
- Sharp thresholds, 51–52, 149.
- Shearer, James Bergheim, 82, 87, 160.
- Sheeran, Mary, 203.
- Shlyakhter, Пуга Alexander (Шляхтер, Илья Александрович), 284.
- Shmoys, David Bernard, 267.
- Shortest paths, 262.
- Shortz, William Frederic, v.
- SIAM: The Society for Industrial and Applied Mathematics, 204.
- Sideways sum ( $\nu x$ ): Sum of binary digits, 114, 143, 179, 195, 279.  
second order ( $\nu^{(2)}x$ ), 143.
- Sifting, 219, 220.
- Sift up in a heap, 234.
- Signature of a clause, 72, 158.
- Signature of a literal, 258.
- Signed mappings, 180–181.
- Signed permutations, 4, 111, 178.  
involutions, 112–113, 180, 277–278.
- Silva, *see* Marques da Silva.
- Silver, Stephen Andrew, 138, 200.
- Simmons, Gustavus James, 192.
- Simon, Laurent Dominique, 72, 132.
- Simple cycles and paths, 23–24, 140.  
*simplex* graphs, 136.
- Simplification of clauses, 65, 155, 232; *see also* Preprocessing of clauses.
- Sims, Charles Coffin, tables, 283.
- Simultaneous read/write, 141.
- Simultaneous write/write, 141.
- Sinclair, Alistair, 80, 159, 256.
- Singh, Satnam, 203.
- Single lookahead unit resolution, 105, 176.
- Single-stuck-at faults, 10–14, 114, 136–137.
- Sink: A vertex with no successor, 87, 214.  
components, 108–110.
- Sinz, Carsten Michael, v, 8, 117, 118, 135, 174, 189, 280.
- Skip Two solitaire, 282.
- Slack, in trace theory, 88, 251.
- Slisenko (= Slissenko), Anatol Olesievitch (Слисенко, Анатолий Олесьевич), 59.
- SLS: Stochastic local search, 77.
- SLUR algorithm, 105, 176.
- Sly, Allan Murray, 51.
- Smile, 207.
- Smith, Barbara Mary, 283.
- Snake dance, 138.
- Snakes, 52–54, 149.
- Snares, 52–54, 149.
- Snark graphs, 69, 147, 153, 157.
- Snevily, Hunter Saint Clair, 5.
- Socrates, son of Sophroniscus of Alopece (Σωκράτης Σωφρωνίσκου Ἀλωπεκῆθεν), 129.
- Soft clauses, 168.
- Sokal, Alan David, 251, 252.
- Solitaire games, 180, 282.
- Solutions, number of, 48, 219.
- Somenzi, Fabio, 236.
- Sörensson, Niklas Kristofer, v, 67, 155, 203, 268.
- Sorting networks, 115, 137, 203, 263, 266.
- Source: A vertex with no predecessor, 87, 252.
- Spaceships in Life, 139, 201.



- Spanning trees, 281, 290.  
 Sparse encoding, *see* Direct encoding.  
 Speckenmeyer, Ewald, 131, 215.  
 Spence, Ivor Thomas Arthur, 290.  
 Spencer, Joel Harold, 81, 82, 254.  
 Spiral order, 206.  
 Stable Life configurations, 19, 197.  
 Stable partial assignments, 165–166.  
 Stacks, 37–39, 43.  
 Stacking the pieces, 84–85.  
 Stålmærck, Gunnar Martin Natanael, 56, 132, 153, 203, 232, 238.  
 Stamm-Wilbrandt, Hermann, 131.  
 STAMP( $l$ ), 258.  
 Stamping of data, 37–38, 64, 66, 145, 155, 211, 236, 258–260.  
 Standard deviation, 48, 240.  
 Stanford GraphBase, ii, 12, 13, 126, 179, 214, 231.  
 Stanford University, 282.  
 Stanley, Richard Peter, 275.  
 Starfish graphs, 249.  
 Starvation, 22–24, 115, 140, 141.  
 Statistical mechanics, 90.  
 Stators in Life, 138.  
 Stege, Ulrike, 207.  
 Stein, Clifford Seth, 267.  
 Steinbach, Heinz Bernd, 275.  
 Steiner, Jacob, tree packing, 264.  
 triple systems, 106, 274.  
 Sterne, Laurence, iii.  
 Stickel, Mark Edward, 132.  
 Sticking values, 67, *see* Phase saving.  
 Still Life, 19, 138, 200.  
 Stirling, James, approximation, 221, 240.  
 subset numbers, 149, 220, 226.  
 Stochastic local search, 77.  
 Stopping time, 48–50, 148.  
 Strahler, Arthur Newell, numbers, 152.  
 Strengthening a clause, 96, 156, 259–260.  
 Stríbrná, Jitka, 224.  
 Strichman, Ofer (עופר שטריכמן), 203.  
 Strictly distinct literals, 2–3, 52, 165.  
 Strings generalized to traces, 83.  
 Strong components: Strongly connected components, 41–42, 52–53, 108, 131, 215, 221, 263.  
 Strong exponential time hypothesis, 183.  
 Strong product of graphs, 134.  
 Strongly balanced sequences, 179.  
 Stuck-at faults, single, 10–14, 114, 136–137.  
 Stützel, Thomas Günter, 125.  
 Subadditive law, 59.  
 Subcubes, 148.  
 Subforests, 42.  
 Subinterval constraints, 190.  
 Submatrices, 106–109, 177.  
 Subset sum problem, 268.  
 Substitution, 257.  
 Subsumption of clauses, 61, 96, 124, 152, 155, 156, 166–168, 181, 269.  
 implementation, 167, 259.  
 on-the-fly, 124, 156.  
 Subtraction, encoding of, 100.  
 Sudoku, 183, 225.  
 Summation by parts, 48.  
 Summers, Jason Edward, 200.  
 Sun, Nike (孙妮克), 51.  
 Support clauses, 99, 114, 171.  
 Survey propagation, 51, 90–95, 165–166, 213.  
 Swaminathan, Ramasubramanian (= Ram) Pattu (ராமசுப்ரமணியன் பட்டி சுவாமிநாதன்), 274.  
 Swapping to the front, 211, 242.  
 Sweep of a matrix, 108–109, 177.  
 Swoop of a matrix problem, 109.  
 Syllogisms, 129.  
 Symeater in Life, 200.  
 Symmetric Boolean functions, 179, 207, 219, 270; *see also* Cardinality constraints.  
 $S_{\leq 1}$ , *see* At-most-one constraint.  
 $S_1$ , 6, 220.  
 $S_{\geq 1}$ , *see* At-least-one constraint.  
 $S_r$ , 135, 179, 256.  
 Symmetric threshold functions, *see* Cardinality constraints.  
 Symmetrical clauses, 105–106, 156.  
 Symmetrical solutions, 138, 183, 274.  
 Symmetries of Boolean functions, 178.  
 Symmetry breaking, vii, 5, 105–114, 138, 176–181, 187, 188, 190–192, 238, 267, 281–283, 285, 288–290.  
 in graph coloring, 99–100, 114, 171, 179, 187.  
 Symmetry from asymmetry, 19, 201.  
 Synthesis of Boolean functions, 137, 178–179, 194.  
 Szabó, Tibor András, 224.  
 Szegedy, Máriaó, 90, 161, 255.  
 Szeider, Stefan Hans, 224, 284.  
 Szemerédi, Endre, 59.  
 Szpankowski, Wojciech, 225.  
 $t$ -snakes, 53, 54, 149.  
 T $\mu$ : teramems = trillions of memory accesses, 110, 121, 126, 265, 281.  
 Tableaux, 275.  
 Taga, Akiko (多賀明子), 264, 267.  
 Tajima, Hiroshi (田島宏史), 100.  
 Tak, Peter van der, 75.  
 Takaki, Kazuya (高木和哉), 224.  
 Tamura, Naoyuki (田村直之), 100, 171, 264, 267, 268.  
 “Take account,” 37, 43, 45–46, 217, 235.  
 Tanjo, Tomoya (丹生智也), 268.  
 TAOCP: *The Art of Computer Programming*, problem, 115, 169.  
 Tape records, 32.  
 Tardos, Gábor, 82, 224, 254.

- Tarjan, Robert Endre, 41, 42, 214, 217.  
Tarnished wires, 13, 193.  
Tatami tilings, 115, 143.  
TAUT: The tautology problem, 3, 129, 130.  
Tautological clause ( $\varphi$ ), 3, 58, 60, 152, 180, 215, 226–228, 258.  
Tensors, 151.  
Teramem ( $T\mu$ ): One trillion memory accesses, 40, 106, 107, 110, 217, 218, 286.  
Ternary clauses, 3–6, 36, 118, 131, 183; *see also* 3SAT.  
Ternary numbers, 100, 141, 179.  
Ternary operations, 9, 136.  
Territory sets, 84, 161, 163.  
Test cases, 113–124.  
    capsule summaries, 114–115.  
Test patterns, *see* Fault testing.  
Tetris, 84.  
Theobald, Gavin Alexander, 190.  
Theory and practice, 109.  
Three-coloring problems, *see* Flower snarks.  
Threshold functions, 100–101, 175.  
Threshold of satisfiability, 50–54, 91, 148–149, 221.  
Threshold parameter  $\Theta$ , 126, 213, 286.  
Thurley, Marc, 262.  
Tie-breakers, 74, 239.  
Tiling a floor, 115, 138, 143, 199.  
Time stamps, *see* Stamping of data.  
Timeouts, 120.  
TIMP tables, 36–40, 43, 45, 144–145.  
To-do stack, 259.  
Tomographically balanced matrices, 141.  
Tomography, 24–26, 115, 141–143, 167, 285.  
Top-down algorithms, 252.  
Topological sorting, 85, 248.  
Toruses, 134, 138, 200.  
Touched clauses, 44.  
Touched variables, 259.  
Tovey, Craig Aaron, 150, 223.  
Tower of Babel solitaire, 282.  
Tower of London solitaire, 282.  
Trace of a matrix: The sum of its diagonal elements, 108, 218.  
Traces (generalized strings), 83–90, 161–162, 252, 254.  
Tradeoffs, 125–126.  
Trail (a basic data structure for Algorithm C), 62–65, 68, 72, 124, 166, 236, 238.  
    reusing, 75.  
Training sets, 15–16, 115, 125–127, 133, 137, 182, 286.  
Transitions between states, 16–24, 175, 202, 218.  
Transitive law, 56, 228.  
Tree-based lookahead, *see* Lookahead forest.  
Tree function, 230.  
Tree-ordered graphs, 163–164.  
Treelike resolution, 55–56, 152–153.  
Treengeling solver, 121.  
Triangle-free graphs, 167.  
Triangles (3-cliques), 167, 238, 264.  
Triangular grids, 136.  
Tribonacci numbers, 216.  
Triggers, 46, 126.  
Trivalent graphs, 147, 154, 231.  
Trivial clauses, 124–127, 156, 236, 239.  
Trivially satisfiable clauses, 3.  
Truemper, Klaus, 273.  
Truszczyński, Mirosław (= Mirek) Janusz, 216.  
Truth, degrees of, 37–39, 42–43, 45–46, 216.  
Truth tables, 129–130, 179, 194, 220, 277.  
Tseytin, Gregory Samuelovich (Цейтин, Григорий Самуилович), 9, 59–60, 71, 133, 152, 154, 168, 178, 215, 231, 290.  
    encodings, 9, 17, 101–102, 136, 173, 195.  
    encodings, half of, 192, 268.  
Tsimelzon, Mark Boris, 134.  
Tuning of parameters, 124–128, 133, 182.  
Turán, Pál (= Paul), 190.  
Turton, William Harry, 180.  
Two-level circuit minimization, 257.  
UC<sub>k</sub>, 176, 273.  
UIP: Unique implication point, 132, 233.  
Unary clauses, *see* Unit clauses.  
Unary representation (= order encoding), 98–101, 114, 120, 170–173, 190, 268, 281.  
Undoing, 28–31, 37–39, 95–96, 143–145, 208, 212, 217–218.  
Uniform distribution, 159.  
Unique implication points, 132, 233.  
Uniquely satisfiable clauses, 48, 219.  
Unit clauses (= unary clauses), 3, 6, 9, 13, 21, 23, 30, 31, 33, 35, 36, 66, 70, 130, 144, 151, 157, 192, 205, 210, 238, 290.  
Unit conditioning, 27, 96, 166, 259, 261.  
Unit propagation ( $\vdash_1$ ), 31–34, 36, 62, 65, 68, 70–71, 93, 97–99, 103–104, 132, 155, 157, 165, 171, 174, 236, 270, 272, 276.  
    generalized to  $\vdash_k$ , 175.  
Universality of Life, 17.  
Unnecessary branches, 55, 227.  
Unsatisfiable core, 185.  
Unsatisfiable formulas, 1.  
    implications of, 104, 175–176.  
Unsolvable problems, 130.  
Urns and balls, 221.  
Urquhart, Alisdair Ian Fenton, 231.  
VAL array, in Algorithm C, 66–68, 73–76, 233–236, 238, 240.  
    in Algorithm L, 37–39, 43, 216.  
Valid partial assignments, 165–166.  
Van de Graaff, Robert Jemison, 198.  
van der Tak, Peter, 75.

- van der Waerden, Bartel Leendert, 4.  
 numbers, 5, *see*  $W(k_0, \dots, k_{b-1})$ .
- van Deventer, Mattijs Oskar, 290.
- Van Gelder, Allen, 71, 233, 237, 263.
- van Maaren, Hans, 37, 46.
- van Rooij, Iris, 207.
- van Zwieten, Joris Edward, 37.
- VAR array, in Algorithm L, 38, 182, 211.
- Variability in performance on satisfiable problems, 35, 120–121, 128, 287.  
 on unsatisfiable problems, 69, 121, 128, 287.
- Variable elimination, 96–97, 101, 102, 129, 154–155, 166–168, 173, 174, 256–257, 259–260, 270, 272.
- Variable interaction graphs, 116–118, 182.
- Variables, 2.  
 introducing new, 3, 6, 8, 9, 13, 60;  
*see* Auxiliary variables, Extended resolution.
- Variance, 49, 158, 164, 240, 243.
- Vassilevska Williams, Virginia Panayotova (Василевска, Виргиния Панайотова), 167.
- Vaughan, Theresa Phillips, 162.
- Verification, 16, 157; *see also* Certificates of unsatisfiability.
- Viennot, Gérard Michel François Xavier, 83, 84, 87, 162, 249.
- Vinci, Leonardo di ser Piero da, 7.
- Virtual unswapping, 211.
- Visualizations, 116–118.
- Vitushinskiy, Pavel Viktorovich (Витушинский, Павел Викторович), 282.
- Vries, Sven de, 206.
- VSIDS, 132.
- $W(k_0, \dots, k_{b-1})$  (van der Waerden numbers), 4–5, 127, 133.
- waarden* ( $j, k; n$ ), 4–5, 32, 35, 37, 39–42, 45, 63–66, 69, 71–75, 97, 112, 115, 121, 127–129, 133, 142–145, 156, 157, 166, 167, 181, 210, 236, 256.
- Waerden, Bartel Leendert van der, 4.  
 numbers, 5, *see*  $W(k_0, \dots, k_{b-1})$ .
- Wagstaff, Samuel Standfield, Jr., 190.
- Wainwright, Robert Thomas, 138, 166, 197, 198.
- Walks in a graph, 260.
- WalkSAT algorithm, 79–81, 93–94, 118, 125, 159–160, 182, 191, 265, 281.
- Walsh, Toby, 272.
- Warmup runs, 125, 239.
- Warners, Johannes (= Joost) Pieter, 268.
- Warrington, Gregory Saunders, 285.
- Watched literals, 30–34, 65–66, 68, 132, 144, 155, 233–236.
- Weakly forcing, 174.
- Websites, ii, iii, v, 118.
- Weighted permutations, 163.
- Wein, Joel Martin, 267.
- Weismantel, Robert, 264.
- Welzl, Emmerich Oskar Roman (= Emo), 158.
- Wermuth, Udo Wilhelm Emil, v.
- Wetzler, Nathan David, 71, 239.
- Wheel graphs ( $W_n$ ), 191.
- Whittlesey, Marshall Andrew, 192.
- Width of a resolution chain, 57–59, 153–154.
- Wieringa, Siert, 129.
- Wigderson, Avi (אבי ויגדרון), 57–58, 153, 231.
- Wilde, Boris de, 213.
- Williams, Richard Ryan, v, 270.
- Williams, Virginia Panayotova Vassilevska (Виргиния Панайотова Василевска), 167.
- Wilson, David Bruce, 54, 149, 221.
- Windfalls, 43, 147, 182, 217.
- Winkler, Peter Mann, 290.
- Winn, John Arthur, Jr., 275.
- Wires of a circuit, 10–14, 136.
- Wobble function, 51, 151.
- Worst case, 144, 146, 154, 239, 244.
- Write buffers, 24.
- Xeon computer, 289.
- XOR operation, 9, 10, 13, 136.  
 bitwise ( $x \oplus y$ ), 28, 137, 196, 208, 220, 241.
- Xray-like projections, 24.
- Xu, Ke (许可), 149.
- Xu, Lin (徐林), 133.
- Xu, Yixin (徐一新), 255.
- Yaroslavtsev, Grigory Nikolaevich (Ярославцев, Григорий Николаевич), 280.
- Yeh, Roger Kwan-Ching (葉光清), 192.
- Yuster, Raphael (רפאל יוסטר), 260.
- $Z(m, n)$  (Zarankewicz numbers), 106–107, 176.
- Zanette, Arrigo, 206.
- Zarankewicz, Kazimierz, 106.  
 quad-free problem, 106–107, 113, 176.
- Závodný, Jakub, 196.
- Zecchina, Riccardo, 51, 90, 91, 256.
- Zhang, Hantao (张瀚涛), 129, 132.
- Zhang, Lintao (张霖涛), 132.
- Zhao, Ying (赵颖), 132.
- Zhu, Yunshan (朱允山), 132.
- ZSEV (zero or set if even), 242.
- Zuckerman, David Isaac, 80, 159.
- Zwick, Uri (אורי צויק), 260.
- Zwieten, Joris Edward van, 37.