THE ART OF COMPUTER PROGRAMMING

VOLUME 4 PRE-FASCICLE 1B

A DRAFT OF SECTION 7.1.4: BINARY DECISION DIAGRAMS

DONALD E. KNUTH *Stanford University*

ADDISON–WESLEY

Internet
Stanford GraphBase

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.

Burn after reading.

Zeroth printing (revision 6), 22 December 2008

Knuth
CERUZZI
Internet
Stanford GraphBase
Internet

PREFACE

How can Knuth finish the series,
given all that has happened in computing
since volume 1 appeared in 1968?
— P. E. CERUZZI, *Computing Reviews* 8805-0370 (May 1988)

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, and 3 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 huge; 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 Section 7.1.4 of a long, long chapter on combinatorial algorithms. Chapter 7 will eventually fill at least three volumes (namely Volumes 4A, 4B, and 4C), more likely four, assuming that I'm able to remain healthy. It will begin with a short review of graph theory, with emphasis on some highlights of significant graphs in the Stanford GraphBase, from which I will be drawing many examples. Then comes Section 7.1: Zeros and Ones, beginning with basic material about Boolean operations in Section 7.1.1 and Boolean evaluation in Section 7.1.2. Section 7.1.3 applies those ideas to make computer programs run fast. And Section 7.1.4, which you're about to read here, discusses the representation of Boolean functions.

The next part, 7.2, is about generating all possibilities, and it begins with Section 7.2.1: Generating Basic Combinatorial Patterns. Section 7.2.2 will deal with backtracking in general. And so it will continue, if all goes well; an outline of the entire Chapter 7 as currently envisaged appears on the `taocp` webpage that is cited on page ii. Fascicles for everything that precedes Section 7.2.2 have already been published, except for Sections 7.1.3 and 7.1.4 (which will soon be packaged into Volume 4 Fascicle 1, filling the gap between Volume 4 Fascicle 0 and Volume 4 Fascicle 2). The pre-fascicle for Section 7.1.3 is available on the Internet for beta-testing.

decision tables

This part of *The Art of Computer Programming* gave me many more surprises than anything else so far. It deals with a topic that burst on the scene in 1986, long after old-timers like me thought that we had already seen all of the basic data structures that would ever prove to be of extraspecial importance. I didn't actually learn about binary decision diagrams until 1995 or so, because I was preoccupied with other things. At that time I wrote some experimental programs and realized that I must try to "shoehorn" this topic into Section 7.1 somehow. I kept seeing more and more papers about it in the literature, and I filed them away with the evergrowing pile of things-to-read-before-revising-Section-7.1. (My first draft of Section 7.1, written in 1977, included a dozen or so pages of material about "decision tables," which I've now discarded because the new ideas are much more important.)

I began to write Section 7.1.4 in May of 2007, thinking that it would eventually fill roughly 35 pages, and that I could easily draft it in three months. Now, more than a year later, I'm looking at more than 130 completed pages — even though I've constantly had to cut, cut, cut! Every week I've been coming across fascinating new things that simply cry out to be part of *The Art*.

Binary decision diagrams (BDDs) are wonderful, and the more I play with them the more I love them. For fifteen months I've been like a child with a new toy, being able now to solve problems that I never imagined would be tractable. (Just last week I was finally able to answer research problem 7.1.1–68 for $n \le 15$, resolving a question that had been bugging me for years.) Every time I've tried a new application, I've learned more. I suspect that many readers will have the same experience, and that there will always be more to learn about such a fertile subject. Already I know that I could easily teach a one-semester college course about binary decision diagrams, at either the undergraduate or graduate level, with more than enough important material to cover. Many aspects of this subject are still ripe for further investigation and improvement.

Most of the theory and practice related to BDDs is due to researchers in the areas of hardware design, testing, and verification. I have, however, tried to present it from the standpoint of a programmer who is primarily interested in combinatorial algorithms. The topic of Boolean functions and binary decision diagrams can of course be interpreted so broadly that it encompasses the entire subject of computer programming. The real goal of this fascicle is to focus on concepts that appear at the lowest levels, concepts on which we can erect significant superstructures. And even these apparently lowly notions turn out to be surprisingly rich, with explicit ties to sections 2.2.1, 2.3.2, 2.3.3, 2.3.4.1, 2.3.4.2, 3.2.2, 3.4.1, 4.3.2, 4.6.4, 5.1.4, 5.3.1, 5.3.4, 6.3, and 6.4 of the first three volumes. I strongly believe in building up a firm foundation, so I have discussed Boolean topics much more thoroughly than I will be able to do with material that is newer or less basic. Section 7.1.4 presented me with an extreme embarrassment of riches: After typing the manuscript I was astonished to discover that I had come up with 264 exercises, even though — believe it or not — I had to eliminate quite a lot of the interesting material that appears in my files. In fact, I know that I've only begun to scratch the surface in some areas of this topic.

The published literature about binary decision diagrams is vast, and still growing rapidly. Most of it appears in the proceedings of conferences that I have never attended, or in specialized journals that I rarely have occasion to read. So I fear that in several respects my knowledge is woefully behind the times, although I've tried my best. Please look, for example, at the exercises that I've classed as research problems (rated with difficulty level 46 or higher), namely exercises 127, 169, 179, 206, 251, and 264; I've also implicitly mentioned or posed additional unsolved questions in the answers to exercises 41, 74, 118, 121(c), 129, 136, 142, 145, 158, 182, 184, 212, 215, 237, 241, and 245. 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 dozens of ideas that occurred to me as I was preparing this material. I couldn't help thinking of basic questions whose answers were not given in any of the publications I had seen. 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 Theorem P, or to the ideas found in exercises 2, 15, 17, 23, 29, 30, 32, 33, 34, 36, 38, 40, 55, 59(b), 60, 61, 63, 64, 72, 74, 76, 77, 88, 92, 100, 107, 110, 111, 119, 120, 124, 125, 126, 132, 135, 146, 156, 157, 160, 161, 162, 164, 174(a,b), 175, 181, 183, 184, 190, 191, 192, 193, 196, 207, 221, 222, 226, 232, 233, 244, 247, 252, 254, 258, or 259, and/or the implementation of $f^\sharp$ in the answer to exercise 236. Have any of those results appeared in print, to your knowledge?

The experimental toolkits that I wrote for working with BDDs and ZDDs while writing this section are available (in unpolished form) on the Internet from my "downloadable programs" page.

I owe a great debt of gratitude to Randy Bryant, Rick Rudell, and Fabio Somenzi, who helped me significantly at several crucial stages as I was preparing Section 7.1.4. Andy Kacsmar generously provided guest accounts on some of Stanford InfoLab's ever-changing computers, and held my hand as I ran some of the larger programs described herein. And as usual I thank dozens of people who have patiently read what I've written and corrected dozens of dozens of mistakes.

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 D. E. K.
28 August 2008

Internet
Bryant
Rudell
Somenzi
Kacsmar
Stanford
Knuth

I at last deliver to the world a Work which I have long promised,
and of which, I am afraid, too high expectations have been raised.
The delay of its publication must be imputed, in a considerable degree,
to the extraordinary zeal which has been shown by distinguished persons
in all quarters to supply me with additional information.

— JAMES BOSWELL, *The Life of Samuel Johnson, LL.D.* (1791)

BOSWELL
Johnson
notation $\langle xyz\rangle$
median function
majority function
Notation
IEEE Transactions

A note on notation. Several formulas in Section 7.1.4 use the notation $\langle xyz\rangle$, for the median function (aka majority function) that is discussed extensively in Section 7.1.1. If you run across other notations that may be unfamiliar, please look at the Index to Notations at the end of Volumes 1, 2, or 3, and/or the entries under "Notation" in the index to the present booklet. Of course Volume 4 will some day contain its own Index to Notations.

A note on references. References to *IEEE Transactions* include a letter code for the type of transactions, in boldface preceding the volume number. For example, '*IEEE Trans.* **C-35**' means the *IEEE Transactions on Computers*, volume 35. The IEEE no longer uses these convenient letter codes, but the codes aren't too hard to decipher: '**EC**' once stood for "Electronic Computers," '**IT**' for "Information Theory," '**SE**' for "Software Engineering," and '**SP**' for "Signal Processing," etc.; '**CAD**' meant "Computer-Aided Design of Integrated Circuits and Systems."

An external exercise. This fascicle refers to exercise 6.4–78, which did not appear in the second edition of Volume 3 until the 24th printing. Here is a copy of that exercise and its answer. (Please don't peek at the answer until you've worked on the exercise.)

Woelfel
universal hashing
saturating subtraction
monus

▸ **78.** [*M26*] (P. Woelfel.) If $0 \le x < 2^n$, let $h_{a,b}(x) = \lfloor (ax+b)/2^k \rfloor \bmod 2^{n-k}$. Show that the set $\{h_{a,b} \mid 0 < a < 2^n,\ a \text{ odd, and } 0 \le b < 2^k\}$ is a universal family of hash functions from n-bit keys to $(n-k)$-bit keys. (These functions are particularly easy to implement on a binary computer.)

78. Let $g(x) = \lfloor x/2^k \rfloor \bmod 2^{n-k}$ and $\delta(x,x') = \sum_{b=0}^{2^k-1} [g(x+b) = g(x'+b)]$. Then $\delta(x+1, x'+1) = \delta(x,x') + [g(x+2^k) = g(x'+2^k)] - [g(x) = g(x')] = \delta(x,x')$. Also $\delta(x,0) = (2^k \dot{-} (x \bmod 2^n)) + (2^k \dot{-} ((-x) \bmod 2^n))$ when $0 < x < 2^n$, where $a \dot{-} b = \max(a-b, 0)$. Therefore $\delta(x,x') = \bigl(2^k \dot{-} ((x-x') \bmod 2^n)\bigr) + \bigl(2^k \dot{-} ((x'-x) \bmod 2^n)\bigr)$ when $x \not\equiv x'$ (modulo 2^n).

Now let $A = \{a \mid 0 < a < 2^n,\ a \text{ odd}\}$ and $B = \{b \mid 0 \le b < 2^k\}$. We want to show that $\sum_{a\in A} \sum_{b \in B} [g(ax+b) = g(ax'+b)] \le R/M = 2^{n-1+k}/2^{n-k} = 2^{2k-1}$ when $0 \le x < x' < 2^n$. And indeed, if $x' - x = 2^p q$ with q odd, then we have

$$\sum_{a\in A} \sum_{b\in B} [g(ax+b) = g(ax'+b)] = \sum_{a\in A} \delta(ax, ax') = 2 \sum_{a \in A} \bigl(2^k \dot{-} ((2^p aq) \bmod 2^n)\bigr)$$

$$= 2^{p+1} \sum_{j=0}^{2^{n-p-1}-1} \bigl(2^k \dot{-} 2^p(2j+1)\bigr) = 2^{p+1} \sum_{j=0}^{2^{k-p-1}-1} \bigl(2^k - 2^p(2j+1)\bigr)[p<k] = 2^{2k-1}[p<k].$$

[See *Lecture Notes in Computer Science* **1672** (1999), 262–272.]

BDD
ROBDD
WIKIPEDIA
binary decision diagrams–
tries
median function+++
root
branch node
dashed line
LO
HI
sink node

In popular usage, the term **BDD** *almost always refers to*
Reduced Ordered Binary Decision Diagram *(ROBDD in the literature,*
used when the ordering and reduction aspects need to be emphasized).

— WIKIPEDIA, *The Free Encyclopedia* (7 July 2007)

7.1.4. Binary Decision Diagrams

Let's turn now to an important family of data structures that have rapidly become the method of choice for representing and manipulating Boolean functions inside a computer. The basic idea is a divide-and-conquer scheme somewhat like the binary tries of Section 6.3, but with several new twists.

Figure 21 shows the binary decision diagram for a simple Boolean function of three variables, the median function $\langle x_1x_2x_3\rangle$ of Eq. 7.1.1–(43). We can understand it as follows: The node at the top is called the *root*. Every internal node (j), also called a *branch node*, is labeled with a name or index $j = V((j))$ that designates a variable; for example, the root node (1) in Fig. 21 designates x_1. Branch nodes have two successors, indicated by descending lines. One of the successors is drawn as a dashed line and called LO; the other is drawn as a solid line and called HI. These branch nodes define a path in the diagram for any values of the Boolean variables, if we start at the root and take the LO branch from node (j) when $x_j = 0$, the HI branch when $x_j = 1$. Eventually this path leads to a *sink node*, which is either $\boxed{\bot}$ (denoting FALSE) or $\boxed{\top}$ (denoting TRUE).

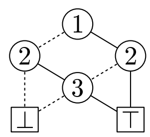
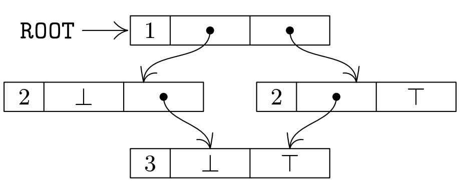


Fig. 21. The binary decision diagram (BDD) for the majority or median function $\langle x_1x_2x_3\rangle$.

In Fig. 21 it's easy to verify that this process yields the function value FALSE when at least two of the variables $\{x_1, x_2, x_3\}$ are 0, otherwise it yields TRUE.

Many authors use $\boxed{0}$ and $\boxed{1}$ to denote the sink nodes. We use $\boxed{\bot}$ and $\boxed{\top}$ instead, hoping to avoid any confusion with the branch nodes (0) and (1).

Inside a computer, Fig. 21 would be represented as a set of four nodes in arbitrary memory locations, where each node has three fields $\boxed{\texttt{V}\,|\,\texttt{LO}\,|\,\texttt{HI}}$. The `V` field holds the index of a variable, while the `LO` and `HI` fields each point to another node or to a sink:

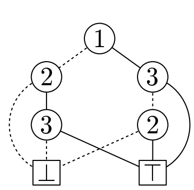


(1)

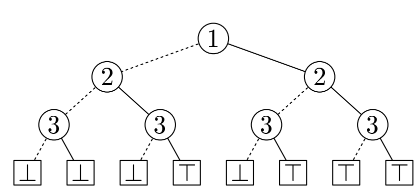
With 64-bit words, we might for example use 8 bits for `V`, then 28 bits for `LO` and the other 28 bits for `HI`.

Such a structure is called a "binary decision diagram," or BDD for short. Small BDDs can readily be drawn as actual diagrams on a piece of paper or a computer screen. But in essence each BDD is really an abstract set of linked nodes, which might more properly be called a "binary decision dag" — a binary tree with shared subtrees, a directed acyclic graph in which exactly two distinguished arcs emanate from every nonsink node.

We shall assume that every BDD obeys two important restrictions. First, it must be *ordered*: Whenever a LO or HI arc goes from branch node (i) to branch node (j), we must have $i < j$. Thus, in particular, no variable x_j will ever be queried twice when the function is evaluated. Second, a BDD must be *reduced*, in the sense that it doesn't waste space. This means that a branch node's LO and HI pointers must never be equal, and that no two nodes are allowed to have the same triple of values (`V`, `LO`, `HI`). Every node should also be accessible from the root. For example, the diagrams



and



(2)

are not BDDs, because the first one isn't ordered and the other one isn't reduced.

Many other flavors of decision diagrams have been invented, and the literature of computer science now contains a rich alphabet soup of acronyms like

FALSE
TRUE
BDD
binary decision dag
binary tree
shared subtrees
directed acyclic graph
dag
Ordered BDD
Reduced BDD

EVBDD, FBDD, IBDD, OBDD, OFDD, OKFDD, PBDD, ..., ZDD. In this book we shall always use the unadorned code name "BDD" to denote a binary decision diagram that is ordered and reduced as described above, just as we generally use the word "tree" to denote an ordered (plane) tree, because such BDDs and such trees are the most common in practice.

Recall from Section 7.1.1 that every Boolean function $f(x_1,\ldots,x_n)$ corresponds to a *truth table*, which is the 2^n-bit binary string that starts with the function value $f(0,\ldots,0)$ and continues with $f(0,\ldots,0,1)$, $f(0,\ldots,0,1,0)$, $f(0,\ldots,0,1,1)$, ..., $f(1,\ldots,1,1,1)$. For example, the truth table of the median function $\langle x_1x_2x_3\rangle$ is 00010111. Notice that this truth table is the same as the sequence of leaves in the unreduced decision tree of (2), with $0 \mapsto \boxed{\bot}$ and $1 \mapsto \boxed{\top}$. In fact, there's an important relationship between truth tables and BDDs, which is best understood in terms of a class of binary strings called "beads."

A truth table of order n is a binary string of length 2^n. A *bead* of order n is a truth table β of order n that is not a square; that is, β doesn't have the form $\alpha\alpha$ for any string α of length 2^{n-1}. (Mathematicians would say that a bead is a "primitive string of length 2^n.") There are two beads of order 0, namely 0 and 1; and there are two of order 1, namely 01 and 10. In general there are $2^{2^n} - 2^{2^{n-1}}$ beads of order n when $n > 0$, because there are 2^{2^n} binary strings of length 2^n and $2^{2^{n-1}}$ of them are squares. The $16 - 4 = 12$ beads of order 2 are

$$0001, 0010, 0011, 0100, 0110, 0111, 1000, 1001, 1011, 1100, 1101, 1110; \qquad (3)$$

these are also the truth tables of all functions $f(x_1, x_2)$ that depend on x_1, in the sense that $f(0, x_2)$ is not the same function as $f(1, x_2)$.

Every truth table τ is a power of a unique bead, called its root. For if τ has length 2^n and isn't already a bead, it's the square of another truth table τ'; and by induction on the length of τ, we must have $\tau' = \beta^k$ for some root β. Hence $\tau = \beta^{2k}$, and β is the root of τ as well as τ'. (Of course k is a power of 2.)

A truth table τ of order $n > 0$ always has the form $\tau_0\tau_1$, where τ_0 and τ_1 are truth tables of order $n-1$. Clearly τ represents the function $f(x_1, x_2, \ldots, x_n)$ if and only if τ_0 represents $f(0, x_2, \ldots, x_n)$ and τ_1 represents $f(1, x_2, \ldots, x_n)$. These functions $f(0, x_2, \ldots, x_n)$ and $f(1, x_2, \ldots, x_n)$ are called *subfunctions* of f; and their truth tables, τ_0 and τ_1, are called *subtables* of τ.

Subtables of a subtable are also considered to be subtables, and a table is considered to be a subtable of itself. Thus, in general, a truth table of order n has 2^k subtables of order $n-k$, for $0 \le k \le n$, corresponding to 2^k possible settings of the first k variables $(x_1, \ldots, x_k)$. Many of these subtables often turn out to be identical; in such cases we're able to represent τ in a compressed form.

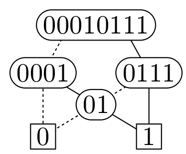
The *beads* of a Boolean function are the subtables of its truth table that happen to be beads. For example, let's consider again the median function $\langle x_1x_2x_3\rangle$, with its truth table 00010111. The distinct subtables of this truth table are $\{00010111, 0001, 0111, 00, 01, 11, 0, 1\}$; and all of them except 00 and 11 are beads. Therefore the beads of $\langle x_1x_2x_3\rangle$ are

$$\{00010111, 0001, 0111, 01, 0, 1\}. \qquad (4)$$

acronyms
EVBDD
FBDD
IBDD
OBDD
OFDD
OKFDD
PBDD
ZDD
truth table+
bead+
square+
primitive
stringology
dependency on a variable
root
subfunctions
subtables
compression of data
beads

$B(f)$
size of its BDD
pi as random ex

And now we get to the point: *The nodes of a Boolean function's BDD are in one-to-one correspondence with its beads.* For example, we can redraw Fig. 21 by placing the relevant bead inside of each node:



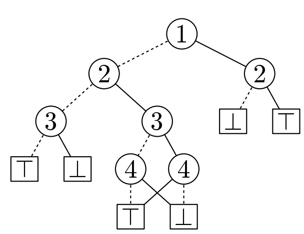
. (5)

In general, a function's truth tables of order $n+1-k$ correspond to its subfunctions $f(c_1,\ldots,c_{k-1},x_k,\ldots,x_n)$ of that order; so its beads of order $n+1-k$ correspond to those subfunctions that depend on their first variable, x_k. Therefore every such bead corresponds to a branch node k in the BDD. And if k is a branch node corresponding to the truth table $\tau' = \tau'_0\tau'_1$, its LO and HI branches point respectively to the nodes that correspond to the roots of τ'_0 and τ'_1.

This correspondence between beads and nodes proves that *every Boolean function has one and only one representation as a BDD.* The individual nodes of that BDD might, of course, be placed in different locations inside a computer.

If f is any Boolean function, let $B(f)$ denote the number of beads that it has. This is the size of its BDD — the total number of nodes, including the sinks. For example, $B(f)=6$ when f is the median-of-three function, because (5) has size 6.

To fix the ideas, let's work out another example, the "more-or-less random" function of 7.1.1–(22) and 7.1.2–(6). Its truth table, 1100100100001111, is a bead, and so are the two subtables 11001001 and 00001111. Thus we know that the root of its BDD will be a 1 branch, and that the LO and HI nodes below the root will both be 2s. The subtables of length 4 are $\{1100, 1001, 0000, 1111\}$; here the first two are beads, but the others are squares. To get to the next level, we break the beads in half and carry over the square roots of the nonbeads, identifying duplicates; this leaves us with $\{11, 00, 10, 01\}$. Again there are two beads, and a final step produces the desired BDD:



. (6)

(In this diagram and others below, it's convenient to repeat the sink nodes $\boxed{\bot}$ and $\boxed{\top}$ in order to avoid excessively long connecting lines. Only one $\boxed{\bot}$ node and one $\boxed{\top}$ node are actually present; so the size of (6) is 9, not 13.)

An alert reader might well be thinking at this point, "Very nice, but what if the BDD is huge?" Indeed, functions can easily be constructed whose BDD is impossibly large; we'll study such cases later. But the wonderful thing is that a great many of the Boolean functions that are of practical importance turn out to have reasonably small values of $B(f)$. So we shall concentrate on the good news first, postponing the bad news until we've seen why BDDs have proved to be so popular.

BDD virtues. If $f(x) = f(x_1, \ldots, x_n)$ is a Boolean function whose BDD is reasonably small, we can do many things quickly and easily. For example:

• We can *evaluate* $f(x)$ in at most n steps, given any input vector $x = x_1 \ldots x_n$, by simply starting at the root and branching until we get to a sink.

• We can *find the lexicographically smallest* x such that $f(x) = 1$, by starting at the root and repeatedly taking the LO branch unless it goes directly to $\boxed{\bot}$. The solution has $x_j = 1$ only when the HI branch was necessary at (j). For example, this procedure gives $x_1x_2x_3 = 011$ in the BDD of Fig. 21, and $x_1x_2x_3x_4 = 0000$ in (6). (It locates the value of x that corresponds to the leftmost 1 in the truth table for f.) Only n steps are needed, because every branch node corresponds to a nonzero bead; we can always find a downward path to $\boxed{\top}$ without backing up. Of course this method fails when the root itself is $\boxed{\bot}$. But that happens only when f is identically zero.

• We can *count the number of solutions* to the equation $f(x) = 1$, using Algorithm C below. That algorithm does $B(f)$ operations on n-bit numbers; so its running time is $O(nB(f))$ in the worst case.

• After Algorithm C has acted, we can speedily *generate random solutions* to the equation $f(x) = 1$, in such a way that every solution is equally likely.

• We can also *list all solutions* x to the equation $f(x) = 1$. The algorithm in exercise 16 does this in $O(nN)$ steps when there are N solutions.

• We can *solve the linear Boolean programming problem*: Find x such that

$$w_1x_1 + \cdots + w_nx_n \text{ is maximum, subject to } f(x_1, \ldots, x_n) = 1, \tag{7}$$

given constants $(w_1, \ldots, w_n)$. Algorithm B (below) does this in $O(n+B(f))$ steps.

• We can *compute the generating function* $a_0 + a_1z + \cdots + a_nz^n$, where there are a_j solutions to $f(x_1, \ldots, x_n) = 1$ with $x_1 + \cdots + x_n = j$. (See exercise 25.)

• We can *calculate the reliability polynomial* $F(p_1, \ldots, p_n)$, which is the probability that $f(x_1, \ldots, x_n) = 1$ when each x_j is independently set to 1 with a given probability p_j. Exercise 26 does this in $O(B(f))$ steps.

Moreover, we will see that BDDs can be combined and modified efficiently. For example, it is not difficult to form the BDDs for $f(x_1, \ldots, x_n) \wedge g(x_1, \ldots, x_n)$ and $f(x_1, \ldots, x_{j-1}, g(x_1, \ldots, x_n), x_{j+1}, \ldots, x_n)$ from the BDDs for f and g.

Algorithms for solving basic problems with BDDs are often described most easily if we assume that the BDD is given as a sequential list of branch instructions $I_{s-1}, I_{s-2}, \ldots, I_1, I_0$, where each I_k has the form $(\bar{v}_k ?\, l_k \colon h_k)$. For example, (6) might be represented as a list of $s = 9$ instructions

$$\begin{array}{lll} I_8 = (\bar{1}?\, 7\colon 6), & I_5 = (\bar{3}?\, 1\colon 0), & I_2 = (\bar{4}?\, 0\colon 1), \\ I_7 = (\bar{2}?\, 5\colon 4), & I_4 = (\bar{3}?\, 3\colon 2), & I_1 = (\bar{5}?\, 1\colon 1), \\ I_6 = (\bar{2}?\, 0\colon 1), & I_3 = (\bar{4}?\, 1\colon 0), & I_0 = (\bar{5}?\, 0\colon 0), \end{array} \tag{8}$$

with $v_8 = 1$, $l_8 = 7$, $h_8 = 6$, $v_7 = 2$, $l_7 = 5$, $h_7 = 4$, …, $v_0 = 5$, $l_0 = h_0 = 0$. In general the instruction '$(\bar{v}?\, l\colon h)$' means, "If $x_v = 0$, go to I_l, otherwise go to I_h,"

evaluation
lexicographically smallest
truth table
count the number of solutions
enumeration of solutions
SAT-counting, see enumeration of solutions
random solutions
list all solutions
Boolean programming problem
linear Boolean programming
generating function
reliability polynomial
sequential representation of BDDs+

except that the last cases I_1 and I_0 are special. We require that the LO and HI branches l_k and h_k satisfy

$$l_k < k, \qquad h_k < k, \qquad v_{l_k} > v_k, \quad \text{and} \quad v_{h_k} > v_k, \qquad \text{for } s > k \geq 2; \qquad (9)$$

in other words, all branches move downward, to variables of greater index. But the sink nodes $\boxed{\top}$ and $\boxed{\bot}$ are represented by dummy instructions I_1 and I_0, in which $l_k = h_k = k$ and the "variable index" v_k has the impossible value $n+1$.

These instructions can be numbered in any way that respects the topological ordering of the BDD, as required by (9). The root node must correspond to I_{s-1}, and the sink nodes must correspond to I_1 and I_0, but the other index numbers aren't so rigidly prescribed. For example, (6) might also be expressed as

$$\begin{array}{lll} I'_8 = (\bar{1}?\,7\!:\!2), & I'_5 = (\bar{4}?\,0\!:\!1), & I'_2 = (\bar{2}?\,0\!:\!1), \\ I'_7 = (\bar{2}?\,4\!:\!6), & I'_4 = (\bar{3}?\,1\!:\!0), & I'_1 = (\bar{5}?\,1\!:\!1), \\ I'_6 = (\bar{3}?\,3\!:\!5), & I'_3 = (\bar{4}?\,1\!:\!0), & I'_0 = (\bar{5}?\,0\!:\!0), \end{array} \qquad (10)$$

and in 46 other isomorphic ways. Inside a computer, the BDD need not actually appear in consecutive locations; we can readily traverse the nodes of any acyclic digraph in topological order, when the nodes are linked as in (1). But we will imagine that they've been arranged sequentially as in (8), so that various algorithms are easier to understand.

One technicality is worth noting: If $f(x) = 1$ for all x, so that the BDD is simply the sink node $\boxed{\top}$, we let $s = 2$ in this sequential representation. Otherwise s is the size of the BDD. Then the root is always represented by I_{s-1}.

Algorithm C (*Count solutions*). Given the BDD for a Boolean function $f(x) = f(x_1, \ldots, x_n)$, represented as a sequence $I_{s-1}, \ldots, I_0$ as described above, this algorithm determines $|f|$, the number of binary vectors $x = x_1 \ldots x_n$ such that $f(x) = 1$. It also computes the table $c_0, c_1, \ldots, c_{s-1}$, where c_k is the number of 1s in the bead that corresponds to I_k.

C1. [Loop over k.] Set $c_0 \leftarrow 0$, $c_1 \leftarrow 1$, and do step C2 for $k = 2, 3, \ldots, s-1$. Then return the answer $2^{v_{s-1}-1} c_{s-1}$.

C2. [Compute c_k.] Set $l \leftarrow l_k$, $h \leftarrow h_k$, and $c_k \leftarrow 2^{v_l - v_k - 1} c_l + 2^{v_h - v_k - 1} c_h$. ▮

For example, when presented with (8), this algorithm computes

$$c_2 \leftarrow 1,\ c_3 \leftarrow 1,\ c_4 \leftarrow 2,\ c_5 \leftarrow 2,\ c_6 \leftarrow 4,\ c_7 \leftarrow 4,\ c_8 \leftarrow 8;$$

the total number of solutions to $f(x_1, x_2, x_3, x_4) = 1$ is 8.

The integers c_k in Algorithm C satisfy

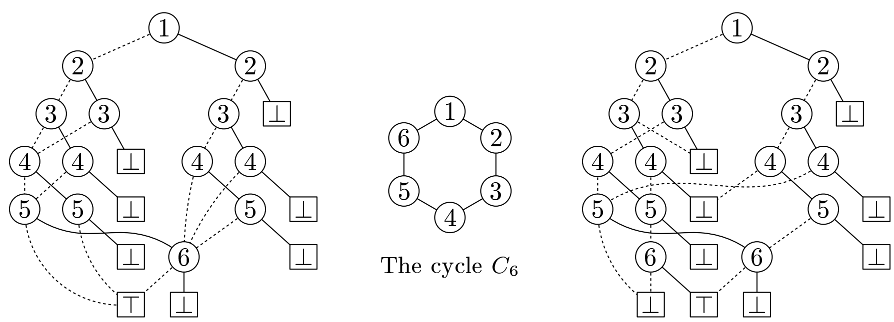
$$0 \leq c_k < 2^{n+1-v_k}, \qquad \text{for } 2 \leq k < s, \qquad (11)$$

and this upper bound is best possible. Therefore multiprecision arithmetic may be needed when n is large. If extra storage space for high precision is problematic, one could use modular arithmetic instead, running the algorithm several times and computing $c_k \bmod p$ for various single-precision primes p; then the final answer would be deducible with the Chinese remainder algorithm, Eq. 4.3.2–(24). On the other hand, floating point arithmetic is usually sufficient in practice.

topological ordering
counting solutions
satisfiability counting
notation $|f|$
multiprecision arithmetic
modular arithmetic
Chinese remainder algorithm
floating point arithmetic

cycle graph
independent
maximal indep subsets
kernels
consecutive 1s forbidden
two-in-a-row function
random solutions to $f(x) = 1+$

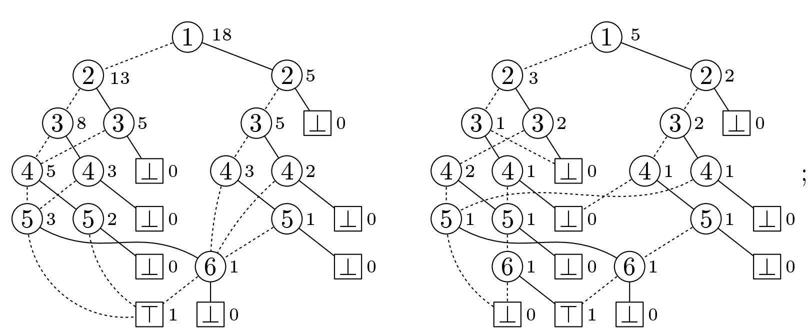
Let's look at some examples that are more interesting than (6). The BDDs



Independent sets — The cycle C_6 — Kernels (12)

represent functions of six variables that correspond to subsets of vertices in the cycle graph C_6. In this setup a vector such as $x_1 \ldots x_6 = 100110$ stands for the subset $\{1, 4, 5\}$; the vector 000000 stands for the empty subset; and so on. On the left is the BDD for which we have $f(x) = 1$ when x is *independent* in C_6; on the right is the BDD for *maximal* independent subsets, also called the *kernels* of C_6 (see exercise 12). In general, the independent subsets of C_n correspond to arrangements of 0s and 1s in a circle of length n, with no two 1s in a row; the kernels correspond to such arrangements in which there also are no three consecutive 0s.

Algorithm C decorates a BDD with counts c_k, working from bottom to top, where c_k is the number of paths from node k to $\boxed{\top}$. When we apply that algorithm to the BDDs in (12) we get



; (13)

hence C_6 has 18 independent sets and 5 kernels.

These counts make it easy to generate uniformly *random* solutions. For example, to get a random independent set vector $x_1 \ldots x_6$, we know that 13 of the solutions in the left-hand BDD have $x_1 = 0$, while the other 5 have $x_1 = 1$. So we set $x_1 \leftarrow 0$ with probability 13/18, and take the LO branch; otherwise we set $x_1 \leftarrow 1$ and take the HI branch. In the latter case, $x_1 = 1$ forces $x_2 \leftarrow 0$, but then x_3 could go either way.

Suppose we've chosen to set $x_1 \leftarrow 1$, $x_2 \leftarrow 0$, $x_3 \leftarrow 0$, and $x_4 \leftarrow 0$; this case occurs with probability $\frac{5}{18} \cdot \frac{5}{5} \cdot \frac{3}{5} \cdot \frac{2}{3} = \frac{2}{18}$. Then there's a branch from 4 to 6, so we flip a coin and set x_5 to a completely random value. In general, a branch from (i) to (j) means that the $j-i-1$ intermediate bits $x_{i+1}\ldots x_{j-1}$ should independently become 0 or 1 with equal probability. Similarly, a branch from (i) to $\boxed{\top}$ should assign random values to $x_{i+1}\ldots x_n$.

Of course there are simpler ways to make a random choice between 18 solutions to a combinatorial problem. Moreover, the right-hand BDD in (13) is an embarrassingly complex way to represent the five kernels of C_6: We could simply have listed them, 001001, 010010, 010101, 100100, 101010! But the point is that this same method will yield the independent sets and kernels of C_n when n is much larger. For example, the 100-cycle C_{100} has 1,630,580,875,002 kernels, yet the BDD describing them has only 855 nodes. One hundred simple steps will therefore generate a fully random kernel from this vast collection.

Boolean programming problem++
binate covering problem, see Boolean programming problem
weighted
Thue sequence+
Morse sequence+

Boolean programming and beyond. A bottom-up algorithm analogous to Algorithm C is also able to find optimum *weighted* solutions (7) to the Boolean equation $f(x)=1$. The basic idea is that it's easy to deduce an optimum solution for any bead of f, once we know optimum solutions for the LO and HI beads that lie directly below it.

Algorithm B (*Solutions of maximum weight*). Let $I_{s-1}, \ldots, I_0$ be a sequence of branch instructions that represents the BDD for a Boolean function f, as in Algorithm C, and let $(w_1,\ldots,w_n)$ be an arbitrary sequence of integer weights. This algorithm finds a binary vector $x=x_1\ldots x_n$ such that $w_1x_1+\cdots+w_nx_n$ is maximum, over all x with $f(x)=1$. We assume that $s>1$; otherwise $f(x)$ is identically 0. Auxiliary integer vectors $m_1\ldots m_{s-1}$ and $W_1\ldots W_{n+1}$ are used in the calculations, as well as an auxiliary bit vector $t_2\ldots t_{s-1}$.

B1. [Initialize.] Set $W_{n+1}\leftarrow 0$ and $W_j\leftarrow W_{j+1}+\max(w_j,0)$ for $n\ge j\ge 1$.

B2. [Loop on k.] Set $m_1\leftarrow 0$ and do step B3 for $2\le k<s$. Then do step B4.

B3. [Process I_k.] Set $v\leftarrow v_k$, $l\leftarrow l_k$, $h\leftarrow h_k$, $t_k\leftarrow 0$. If $l\ne 0$, set $m_k\leftarrow m_l+W_{v+1}-W_{v_l}$. Then if $h\ne 0$, compute $m\leftarrow m_h+W_{v+1}-W_{v_h}+w_v$; and if $l=0$ or $m>m_k$, set $m_k\leftarrow m$ and $t_k\leftarrow 1$.

B4. [Compute the x's.] Set $j\leftarrow 0$, $k\leftarrow s-1$, and do the following operations until $j=n$: While $j<v_k-1$, set $j\leftarrow j+1$ and $x_j\leftarrow[w_j>0]$; if $k>1$, set $j\leftarrow j+1$ and $x_j\leftarrow t_k$ and $k\leftarrow(t_k=0?\ l_k\colon h_k)$. ▮

A simple case of this algorithm is worked out in exercise 18. Step B3 does technical maneuvers that may look a bit scary, but their net effect is just to compute

$$m_k \;\leftarrow\; \max(m_l+W_{v+1}-W_{v_l},\, m_h+W_{v+1}-W_{v_h}+w_v), \tag{14}$$

and to record in t_k whether l or h is better. In fact, v_l and v_h are usually both equal to $v+1$; then the calculation simply sets $m_k\leftarrow\max(m_l, m_h+w_v)$, corresponding to the cases $x_v=0$ and $x_v=1$. Technicalities arise only because we want to avoid fetching m_0, which is $-\infty$, and because v_l or v_h might exceed $v+1$.

With this algorithm we can, for example, quickly find an optimum set of kernel vertices in an n-cycle C_n, using weights based on the "Thue–Morse" sequence,

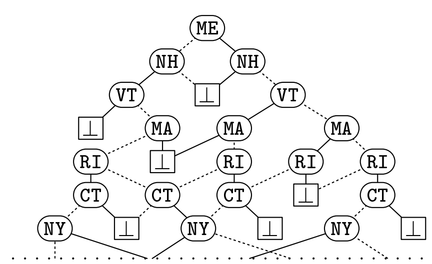
$$w_j \;=\; (-1)^{\nu j}; \tag{15}$$

here νj denotes sideways addition, Eq. 7.1.3–(59). In other words, w_j is -1 or $+1$, depending on whether j has odd parity or even parity when expressed as a binary number. The maximum of $w_1x_1 + \cdots + w_nx_n$ occurs when the even-parity vertices 3, 5, 6, 9, 10, 12, 15, ... most strongly outnumber the odd-parity vertices 1, 2, 4, 7, 8, 11, 13, ... that appear in a kernel. It turns out that

$$\{1,3,6,9,12,15,18,20,23,25,27,30,33,36,39,41,43,46,48,\\ 51,54,57,60,63,66,68,71,73,75,78,80,83,86,89,92,95,97,99\} \qquad (16)$$

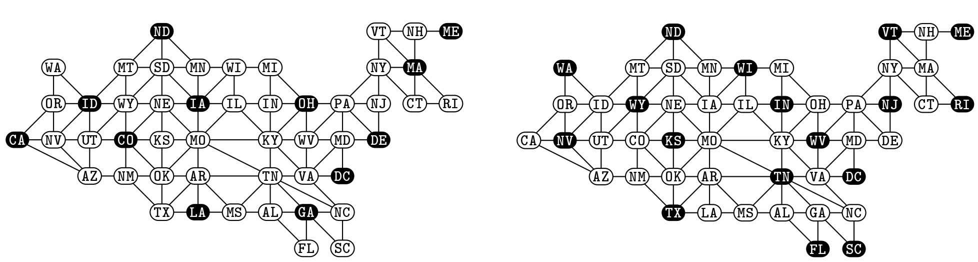
is an optimum kernel in this sense when $n = 100$; only five vertices of odd parity, namely $\{1, 25, 41, 73, 97\}$, need to be included in this set of 38 to satisfy the kernel conditions, hence $\max(w_1x_1+\cdots+w_{100}x_{100}) = 28$. Thanks to Algorithm B, a few thousand computer instructions are sufficient to select (16) from more than a trillion possible kernels, because the BDD for all those kernels happens to be small.

Mathematically pristine problems related to combinatorial objects like cycle kernels could also be resolved efficiently with more traditional techniques, which are based on recurrences and induction. But the beauty of BDD methods is that they apply also to real-world problems that don't have any elegant structure. For example, let's consider the graph of 49 "united states" that appeared in 7–(17) and 7–(61). The Boolean function that represents all the maximal independent sets of that graph (all the kernels) has a BDD of size 780 that begins as follows:



(17)

Algorithm B quickly discovers the following kernels of minimum and maximum weight, when each state vertex is simply weighted according to the sum of letters in its postal code ($w_{\mathtt{CA}} = 3 + 1$, $w_{\mathtt{DC}} = 4 + 3$, ..., $w_{\mathtt{WY}} = 23 + 25$):



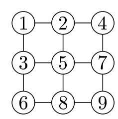
Minimum weight = 155 Maximum weight = 492 (18)

This graph has 266,137 kernels; but with Algorithm B, we needn't generate them all. In fact, the right-hand example in (18) could also be obtained with a smaller BDD of size 428, which characterizes the *independent sets*, because all weights

νj
sideways addition
parity
United States of America, contiguous
contiguous USA
New England
independent sets+

grid
connectedness+
spanning subgraphs
generating function
reliability polynomial+
availability polynomial of a Boolean function, see reliability
characteristic polynomial of a Boolean function, see reliability polynomial
multilinear
integer multilinear representations
Boole

are positive. (A kernel of maximum weight is the same thing as an independent set of maximum weight, in such cases.) There are 211,954,906 independent sets in this graph, many more than the number of kernels; yet we can find an independent set of maximum weight more quickly than a kernel of maximum weight, because the BDD is smaller.



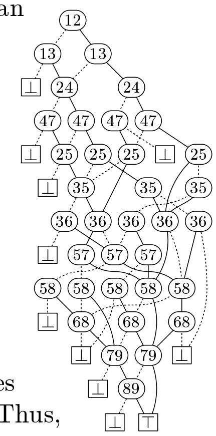


Fig. 22. The grid $P_3 \,\square\, P_3$, and a BDD for its connected subgraphs.

A quite different sort of graph-related BDD is shown in Fig. 22. This one is based on the 3×3 grid $P_3 \,\square\, P_3$; it characterizes the sets of edges that connect all vertices of the grid together. Thus, it's a function $f(x_{12}, x_{13}, \ldots, x_{89})$ of the twelve edges $1 — 2$, $1 — 3$, $\ldots$, $8 — 9$ instead of the nine vertices $\{1, \ldots, 9\}$. Exercise 55 describes one way to construct it. When Algorithm C is applied to this BDD, it tells us that exactly 431 of the $2^{12} = 4096$ spanning subgraphs of $P_3 \,\square\, P_3$ are connected.

A straightforward extension of Algorithm C (see exercise 25) will refine this total and compute the *generating function* of these solutions, namely

$$G(z) \;=\; \sum_x z^{\nu x} f(x) \;=\; 192z^8 + 164z^9 + 62z^{10} + 12z^{11} + z^{12}. \qquad (19)$$

Thus $P_3 \,\square\, P_3$ has 192 spanning trees, plus 164 spanning subgraphs that are connected and have nine edges, and so on. Exercise 7.2.1.6–106(a) gives a formula for the number of spanning trees in $P_m \,\square\, P_n$ for general m and n; but the full generating function $G(z)$ contains considerably more information, and it probably has no simple formula unless $\min(m, n)$ is small.

Suppose each edge $u — v$ is present with probability p_{uv}, independent of all other edges of $P_3 \,\square\, P_3$. What is the probability that the resulting subgraph is connected? This is the *reliability polynomial*, which also goes by a variety of other names because it arises in many different applications. In general, as discussed in exercise 7.1.1–12, every Boolean function $f(x_1, \ldots, x_n)$ has a unique representation as a polynomial $F(x_1, \ldots, x_n)$ with the properties that

i) $F(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$ whenever each x_j is 0 or 1;
ii) $F(x_1, \ldots, x_n)$ is multilinear: Its degree in x_j is ≤ 1 for all j.

This polynomial F has integer coefficients and satisfies the basic recurrence

$$F(x_1, \ldots, x_n) \;=\; (1 - x_1)F_0(x_2, \ldots, x_n) + x_1 F_1(x_2, \ldots, x_n), \qquad (20)$$

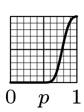
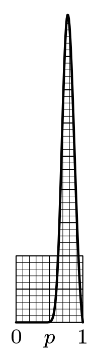
where F_0 and F_1 are the integer multilinear representations of $f(0, x_2, \ldots, x_n)$ and $f(1, x_2, \ldots, x_n)$. Indeed, (20) is George Boole's "law of development."

Two important things follow from recurrence (20). First, F is precisely the reliability polynomial $F(p_1, \ldots, p_n)$ mentioned earlier, because the reliability polynomial obviously satisfies the same recurrence. Second, F is easily calculated from the BDD for f, working upward from the bottom and using (20) to compute the reliability of each bead. (See exercise 26.)

The connectivity function for an 8×8 grid $P_8 \,\square\, P_8$ is, of course, much more complicated than the one for $P_3 \square P_3$; it is a Boolean function of 112 variables and its BDD has 43790 nodes, compared to only 37 in Fig. 22. Still, computations with this BDD are quite feasible, and in a second or two we can compute

$$G(z) = 126231322912498539682594816z^{63} + 1006611140035411062600761344z^{64} + \cdots + 6212z^{110} + 112z^{111} + z^{112},$$

as well as the probability $F(p)$ of connectedness and its derivative $F'(p)$, when each of the edges is present with probability p (see exercise 29):

$F(p)$:  ; $F'(p)$:  . (21)

***A sweeping generalization.** Algorithms B and C and the algorithms we've been discussing for bottom-up BDD scanning are actually special cases of a much more general scheme that can be exploited in many additional ways. Consider an abstract algebra with two associative binary operators $\circ$ and $\bullet$, satisfying the distributive laws

$$\alpha \bullet (\beta \circ \gamma) = (\alpha \bullet \beta) \circ (\alpha \bullet \gamma), \qquad (\beta \circ \gamma) \bullet \alpha = (\beta \bullet \alpha) \circ (\gamma \bullet \alpha). \tag{22}$$

Every Boolean function $f(x_1, \ldots, x_n)$ corresponds to a *fully elaborated truth table* involving the symbols $\circ$, $\bullet$, $\bot$, and $\top$, together with $\bar{x}_j$ and x_j for $1 \le j \le n$, in a way that's best understood by considering a small example: When $n = 2$ and when the ordinary truth table for f is 0010, the fully elaborated truth table is

$$(\bar{x}_1 \bullet \bar{x}_2 \bullet \bot) \circ (\bar{x}_1 \bullet x_2 \bullet \bot) \circ (x_1 \bullet \bar{x}_2 \bullet \top) \circ (x_1 \bullet x_2 \bullet \bot). \tag{23}$$

The meaning of such an expression depends on the meanings that we attach to the symbols $\circ$, $\bullet$, $\bot$, $\top$, and to the literals $\bar{x}_j$ and x_j; but whatever the expression means, we can compute it directly from the BDD for f.

For example, let's return to Fig. 21, the BDD for $\langle x_1x_2x_3\rangle$. The elaborations of nodes $\boxed{\bot}$ and $\boxed{\top}$ are $\alpha_\bot = \bot$ and $\alpha_\top = \top$, respectively. Then the elaboration of 3 is $\alpha_3 = (\bar{x}_3 \bullet \alpha_\bot) \circ (x_3 \bullet \alpha_\top)$; the elaborations of the nodes labeled 2 are $\alpha_2^l = (\bar{x}_2 \bullet (\bar{x}_3 \circ x_3) \bullet \alpha_\bot) \circ (x_2 \bullet \alpha_3)$ on the left and $\alpha_2^r = (\bar{x}_2 \bullet \alpha_3) \circ (x_2 \bullet (\bar{x}_3 \circ x_3) \bullet \alpha_\top)$ on the right; and the elaboration of node 1 is $\alpha_1 = (\bar{x}_1 \bullet \alpha_2^l) \circ (x_1 \bullet \alpha_2^r)$. (Exercise 31 discusses the general procedure.) Expanding these formulas via the distributive laws (22) leads to a full elaboration with $2^n = 8$ "terms":

$$\begin{aligned}\alpha_1 = {} & (\bar{x}_1 \bullet \bar{x}_2 \bullet \bar{x}_3 \bullet \bot) \circ (\bar{x}_1 \bullet \bar{x}_2 \bullet x_3 \bullet \bot) \circ (\bar{x}_1 \bullet x_2 \bullet \bar{x}_3 \bullet \bot) \circ (\bar{x}_1 \bullet x_2 \bullet x_3 \bullet \top) \\ & \circ (x_1 \bullet \bar{x}_2 \bullet \bar{x}_3 \bullet \bot) \circ (x_1 \bullet \bar{x}_2 \bullet x_3 \bullet \top) \circ (x_1 \bullet x_2 \bullet \bar{x}_3 \bullet \top) \circ (x_1 \bullet x_2 \bullet x_3 \bullet \top).\end{aligned} \tag{24}$$

derivative
generalization, sweeping+
abstract algebra+
fully elaborated truth table+
associative
distributive laws
fully elaborated truth table
truth table
literals
median function

maximum operator
symmetric function
sideways addition
condensation
threshold function

Algorithm C is the special case where '$\circ$' is addition, '$\bullet$' is multiplication, '$\bot$' is 0, '$\top$' is 1, '$\bar{x}_j$' is 1, and 'x_j' is also 1. Algorithm B arises when '$\circ$' is the *maximum operator* and '$\bullet$' is addition; the distributive laws

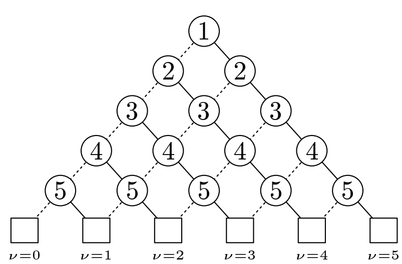
$$\alpha + \max(\beta,\gamma) = \max(\alpha{+}\beta, \alpha{+}\gamma), \quad \max(\beta,\gamma) + \alpha = \max(\beta{+}\alpha, \gamma{+}\alpha) \qquad (25)$$

are easily checked. We interpret '$\bot$' as $-\infty$, '$\top$' as 0, '$\bar{x}_j$' as 0, and 'x_j' as w_j. Then, for example, (24) becomes

$$\max(-\infty, -\infty, -\infty, w_2 + w_3, -\infty, w_1 + w_3, w_1 + w_2, w_1 + w_2 + w_3);$$

and in general the full elaboration under this interpretation is equivalent to the expression $\max\{w_1x_1 + \cdots + w_nx_n \mid f(x_1,\ldots,x_n) = 1\}$.

Friendly functions. Many families of functions are known to have BDDs of modest size. If f is, for example, a symmetric function of n variables, it's easy to see that $B(f) = O(n^2)$. Indeed, when $n = 5$ we can start with the triangular pattern



(26)

and set the leaves to $\boxed{\bot}$ or $\boxed{\top}$ depending on the respective values of f when the value of $\nu x = x_1 {+} \cdots {+} x_5$ equals 0, 1, 2, 3, 4, or 5. Then we can remove redundant or equivalent nodes, always obtaining a BDD whose size is $\binom{n+2}{2}$ or less.

Suppose we take any function $f(x_1,\ldots,x_n)$ and make two adjacent variables equal:

$$g(x_1,\ldots,x_n) \;=\; f(x_1,\ldots,x_{k-1},x_k,x_k,x_{k+2},\ldots,x_n). \qquad (27)$$

Exercise 40 proves that $B(g) \le B(f)$. And by repeating this condensation process, we find that a function such as $f(x_1,x_1,x_3,x_3,x_3,x_6)$ has a small BDD whenever $B(f)$ is small. In particular, the threshold function $[2x_1 + 3x_3 + x_6 \ge t]$ must have a small BDD for any value of t, because it's a condensed version of the symmetric function $f(x_1,\ldots,x_6) = [x_1 + \cdots + x_6 \ge t]$. This argument shows that *any* threshold function with nonnegative integer weights,

$$f(x_1,x_2,\ldots,x_n) \;=\; [w_1x_1 + w_2x_2 + \cdots + w_nx_n \ge t], \qquad (28)$$

can be obtained by condensing a symmetric function of $w_1 + w_2 + \cdots + w_n$ variables, so its BDD size is $O(w_1 + w_2 + \cdots + w_n)^2$.

Threshold functions often turn out to be easy even when the weights grow exponentially. For example, suppose $t = (t_1t_2\ldots t_n)_2$ and consider

$$f_t(x_1,x_2,\ldots,x_n) = [2^{n-1}x_1 + 2^{n-2}x_2 + \cdots + x_n \ge t]. \qquad (29)$$

This function is true if and only if the binary string $x_1x_2\ldots x_n$ is lexicographically greater than or equal to $t_1t_2\ldots t_n$, and its BDD always has exactly $n+2$ nodes when $t_n=1$. (See exercise 170.)

Another kind of function with small BDD is the 2^m-way multiplexer of Eq. 7.1.2–(31), a function of $n=m+2^m$ variables:

$$M_m(x_1,\ldots,x_m;x_{m+1},\ldots,x_n) \;=\; x_{m+1+(x_1\ldots x_m)_2}. \qquad (30)$$

Its BDD begins with 2^{k-1} branch nodes (k) for $1\le k\le m$. But below that complete binary tree, there's just one (k) for each x_k in the main block of variables with $m<k\le n$. Hence $B(M_m)=1+2+\cdots+2^{m-1}+2^m+2=2^{m+1}+1<2n$.

A linear network model of computation, illustrated in Fig. 23, helps to clarify the cases where a BDD is especially efficient. Consider an arrangement of computational modules M_1, M_2, …, M_n, in which the Boolean variable x_k is input to module M_k; there also are wires between neighboring modules, each carrying a Boolean signal, with a_k wires from M_k to M_{k+1} and b_k wires from M_{k+1} to M_k for $1\le k\le n$. A special wire out of M_n contains the output of the function, $f(x_1,\ldots,x_n)$. We define $a_0=b_0=b_n=0$ and $a_n=1$, so that module M_k has exactly $c_k=1+a_{k-1}+b_k$ input ports and exactly $d_k=a_k+b_{k-1}$ output ports for each k. It computes d_k Boolean functions of its c_k inputs.

The individual functions computed by each module can be arbitrarily complicated, but they must be *well defined* in the sense that their joint values are completely determined by the x's: Every choice of $(x_1,\ldots,x_n)$ must lead to exactly one way to set the signals on all the wires, consistent with all of the given functions.

Theorem M. *If f can be computed by such a network, then $B(f)\le\sum_{k=0}^{n}2^{a_k2^{b_k}}$.*

Proof. We will show that the BDD for f has at most $2^{a_{k-1}2^{b_{k-1}}}$ branch nodes (k), for $1\le k\le n$. This is clear if $b_{k-1}=0$, because at most $2^{a_{k-1}}$ subfunctions are possible when x_1 through x_{k-1} have any given values. So we will show that any network that has a_{k-1} forward wires and b_{k-1} backward wires between M_{k-1} and M_k can be replaced by an equivalent network that has $a_{k-1}2^{b_{k-1}}$ forward wires and none that run backward.

For convenience, let's consider the case $k=4$ in Fig. 23, with $a_3=4$ and $b_3=2$; we want to replace those 6 wires by 16 that run only forward. Suppose Alice is in charge of M_3 and Bob is in charge of M_4. Alice sends a 4-bit signal, a, to Bob while he sends a 2-bit signal, b, to her. More precisely, for any fixed value of $(x_1,\ldots,x_n)$, Alice computes a certain function A and Bob computes a function B, where

$$A(b)=a \qquad\text{and}\qquad B(a)=b. \qquad (31)$$

Alice's function A depends on (x_1,x_2,x_3), so Bob doesn't know what it is; Bob's function B is, similarly, unknown to Alice, since it depends on $(x_4,\ldots,x_n)$. But those unknown functions have the key property that, for every choice of $(x_1,\ldots,x_n)$, there's exactly one solution (a,b) to the equations (31).

lexicographically
2^m-way multiplexer
storage access function, see $2^m - way$ multiplexer
Notation M_m
complete binary tree
network model of computation+
modules in a network+
subfunctions

three-in-a-row function
necklace
Shared BDDs, see BDD base
BDD base+
bead
truth tables
root pointers
subfunction
addition, binary

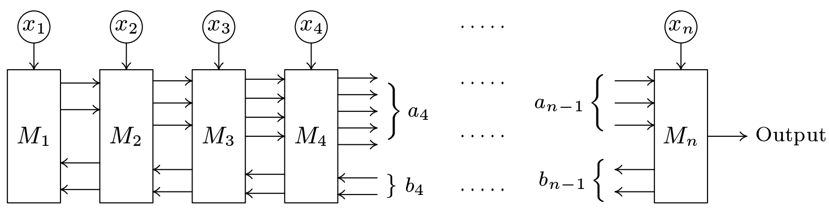


Fig. 23. A generic network of Boolean modules for which Theorem M is valid.

So Alice changes the behavior of module M_3: She sends Bob *four* 4-bit values, $A(00)$, $A(01)$, $A(10)$, and $A(11)$, thereby revealing her A function. And Bob changes the behavior of M_4: Instead of sending any feedback, he looks at those four values, together with his other inputs (namely x_4 and the b_4 bits received from M_5), and discovers the unique a and b that solve (31). His new module uses this value of a to compute the a_4 bits that he outputs to M_5. ▮

Theorem M says that the BDD size will be reasonably small if we can construct such a network with small values of a_k and b_k. Indeed, $B(f)$ will be $O(n)$ if the a's and b's are bounded, although the constant of proportionality might be huge. Let's work an example by considering the *three-in-a-row function*,

$$f(x_1,\ldots,x_n) = x_1x_2x_3 \vee x_2x_3x_4 \vee \cdots \vee x_{n-2}x_{n-1}x_n \vee x_{n-1}x_nx_1 \vee x_nx_1x_2, \quad (32)$$

which is true if and only if a circular necklace labeled with bits $x_1, \ldots, x_n$ has three consecutive 1s. One way to implement it via Boolean modules is to give M_k three inputs (u_k, v_k, w_k) from M_{k-1} and two inputs (y_k, z_k) from M_{k+1}, where

$$\begin{gathered} u_k = x_{k-1}, \quad v_k = x_{k-2}x_{k-1}, \quad w_k = x_{n-1}x_nx_1 \vee \cdots \vee x_{k-3}x_{k-2}x_{k-1}; \\ y_k = x_n, \quad z_k = x_{n-1}x_n. \end{gathered} \quad (33)$$

Here subscripts are treated modulo n, and appropriate changes are made at the left or right when $k = 1$ or $k \ge n-1$. Then M_k computes the functions

$$u_{k+1} = x_k, \quad v_{k+1} = u_kx_k, \quad w_{k+1} = w_k \vee v_kx_k, \quad y_{k-1} = y_k, \quad z_{k-1} = z_k \quad (34)$$

for nearly all values of k; exercise 45 has the details. With this construction we have $a_k \le 3$ and $b_k \le 2$ for all k, hence Theorem M tells us that $B(f) \le 2^{12}n = 4096n$. In fact, the truth is much sweeter: $B(f)$ is actually $< 9n$ (see exercise 46).

Shared BDDs. We often want to deal with several Boolean functions at once, and related functions often have common subfunctions. In such cases we can work with the "BDD base" for $\{f_1(x_1,\ldots,x_n),\ldots,f_m(x_1,\ldots,x_n)\}$, which is a directed acyclic graph that contains one node for every bead that occurs within the truth tables of any of the functions. The BDD base also has m "root pointers," F_j, one for each function f_j; the BDD for f_j is then the set of all nodes reachable from node F_j. Notice that node F_j itself is reachable from node F_i if and only if f_j is a subfunction of f_i.

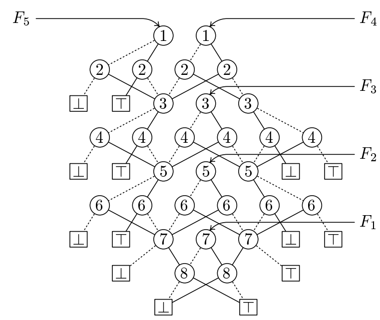
For example, consider the problem of computing the $n+1$ bits of the sum of two n-bit numbers,

$$(f_{n+1}f_nf_{n-1}\ldots f_1)_2 \;=\; (x_1x_3\ldots x_{2n-1})_2 + (x_2x_4\ldots x_{2n})_2. \quad (35)$$

carry
binary decision diagram
Sieling
Wegener
MMIX
ordered
reduction to a BDD++
AVAIL stack+

The BDD base for those $n+1$ bits looks like this when $n=4$:

$$\begin{array}{r} x_1x_3x_5x_7 \\ +\ x_2x_4x_6x_8 \\ \hline f_5\,f_4\,f_3\,f_2\,f_1 \end{array}$$



(36)

The way we've numbered the x's in (35) is important here (see exercise 51). In general there are exactly $B(f_1,\ldots,f_{n+1}) = 9n-5$ nodes, when $n>1$. The node just to the left of F_j, for $1 \le j \le n$, represents the subfunction for a *carry* c_j out of the jth bit position from the right; the node just to the right of F_j represents the complement of that carry, $\bar{c}_j$; and node F_{n+1} represents the final carry c_n.

Operations on BDDs. We've been talking about lots of things to do when a BDD is given. But how do we get a BDD into the computer in the first place?

One way is to start with an ordered binary decision diagram such as (26) or the right-hand example in (2), and to reduce it so that it becomes a true BDD. The following algorithm, based on ideas of D. Sieling and I. Wegener [*Information Processing Letters* **48** (1993), 139–144], shows that an arbitrary N-node binary decision diagram whose branches are properly ordered can be reduced to a BDD in $O(N+n)$ steps when there are n variables.

Of course we need some extra memory space in order to decide whether two nodes are equivalent, when doing such a reduction. Having only the three fields (V, LO, HI) in each node, as in (1), would give us no room to maneuver. Fortunately, only one additional pointer-size field, called AUX, is needed, together with two additional state bits. We will assume for convenience that the state bits are implicitly present in the *signs* of the LO and AUX fields, so that the algorithm needs to deal with only four fields: (V, LO, HI, AUX). The fact that the sign is preempted does mean that a 28-bit LO field will accommodate only 2^{27} nodes at most — about 134 million — instead of 2^{28}. (On a computer like MMIX, we might prefer to assume that all node addresses are even, and to add 1 to a field instead of complementing it as done here.)

Algorithm R (*Reduction to a BDD*). Given a binary decision diagram that is ordered but not necessarily reduced, this algorithm transforms it into a valid BDD by removing unnecessary nodes and rerouting all pointers appropriately. Each node is assumed to have four fields (V, LO, HI, AUX) as described above, and ROOT points to the diagram's top node. The AUX fields are initially irrelevant, except that they must be nonnegative; they will again be nonnegative at the end of the process. All deleted nodes are pushed onto a stack addressed by AVAIL, linked together by the HI fields of its nodes. (The LO fields of these nodes will be negative; their complements point to equivalent nodes that have *not* been deleted.)

bitwise complement
depth-first search
reachable
Bucket sort

The `V` fields of branch nodes are assumed to run from `V(ROOT)` up to $v_{\max}$, in increasing order from the top downwards in the given dag. The sink nodes $\boxed{\bot}$ and $\boxed{\top}$ are assumed to be nodes 0 and 1, respectively, with nonnegative `LO` and `HI` fields. They are never deleted; in fact, they are left untouched except for their `AUX` fields. An auxiliary array of pointers, $\mathtt{HEAD}[v]$ for $\mathtt{V(ROOT)} \le v \le v_{\max}$, is used to create temporary lists of all nodes that have a given value of `V`.

R1. [Initialize.] Terminate immediately if $\mathtt{ROOT} \le 1$. Otherwise, set $\mathtt{AUX}(0) \leftarrow \mathtt{AUX}(1) \leftarrow \mathtt{AUX(ROOT)} \leftarrow -1$, and $\mathtt{HEAD}[v] \leftarrow -1$ for $\mathtt{V(ROOT)} \le v \le v_{\max}$. (We use the fact that $-1 = {\sim}0$ is the bitwise complement of 0.) Then set $s \leftarrow \mathtt{ROOT}$ and do the following operations while $s \ne 0$:

> Set $p \leftarrow s$, $s \leftarrow {\sim}\mathtt{AUX}(p)$, $\mathtt{AUX}(p) \leftarrow \mathtt{HEAD[V}(p)\mathtt{]}$, $\mathtt{HEAD[V}(p)\mathtt{]} \leftarrow {\sim}p$.
> If $\mathtt{AUX(LO}(p)\mathtt{)} \ge 0$, set $\mathtt{AUX(LO}(p)\mathtt{)} \leftarrow {\sim}s$ and $s \leftarrow \mathtt{LO}(p)$.
> If $\mathtt{AUX(HI}(p)\mathtt{)} \ge 0$, set $\mathtt{AUX(HI}(p)\mathtt{)} \leftarrow {\sim}s$ and $s \leftarrow \mathtt{HI}(p)$.

(We've essentially done a depth-first search of the dag, temporarily marking all nodes reachable from `ROOT` by making their `AUX` fields negative.)

R2. [Loop on v.] Set $\mathtt{AUX}(0) \leftarrow \mathtt{AUX}(1) \leftarrow 0$, and $v \leftarrow v_{\max}$.

R3. [Bucket sort.] (At this point all remaining nodes whose `V` field exceeds v have been properly reduced, and their `AUX` fields are nonnegative.) Set $p \leftarrow {\sim}\mathtt{HEAD}[v]$, $s \leftarrow 0$, and do the following steps while $p \ne 0$:

> Set $p' \leftarrow {\sim}\mathtt{AUX}(p)$.
> Set $q \leftarrow \mathtt{HI}(p)$; if $\mathtt{LO}(q) < 0$, set $\mathtt{HI}(p) \leftarrow {\sim}\mathtt{LO}(q)$.
> Set $q \leftarrow \mathtt{LO}(p)$; if $\mathtt{LO}(q) < 0$, set $\mathtt{LO}(p) \leftarrow {\sim}\mathtt{LO}(q)$ and $q \leftarrow \mathtt{LO}(p)$.
> If $q = \mathtt{HI}(p)$, set $\mathtt{LO}(p) \leftarrow {\sim}q$, $\mathtt{HI}(p) \leftarrow \mathtt{AVAIL}$, $\mathtt{AUX}(p) \leftarrow 0$, $\mathtt{AVAIL} \leftarrow p$;
> otherwise if $\mathtt{AUX}(q) \ge 0$, set $\mathtt{AUX}(p) \leftarrow s$, $s \leftarrow {\sim}q$, and $\mathtt{AUX}(q) \leftarrow {\sim}p$;
> otherwise set $\mathtt{AUX}(p) \leftarrow \mathtt{AUX}({\sim}\mathtt{AUX}(q))$ and $\mathtt{AUX}({\sim}\mathtt{AUX}(q)) \leftarrow p$.
> Then set $p \leftarrow p'$.

R4. [Clean up.] (Nodes with $\mathtt{LO} = x \ne \mathtt{HI}$ have now been linked together via their `AUX` fields, beginning with ${\sim}\mathtt{AUX}(x)$.) Set $r \leftarrow {\sim}s$, $s \leftarrow 0$, and do the following while $r \ge 0$:

> Set $q \leftarrow {\sim}\mathtt{AUX}(r)$ and $\mathtt{AUX}(r) \leftarrow 0$.
> If $s = 0$ set $s \leftarrow q$; otherwise set $\mathtt{AUX}(p) \leftarrow q$.
> Set $p \leftarrow q$; then while $\mathtt{AUX}(p) > 0$, set $p \leftarrow \mathtt{AUX}(p)$.
> Set $r \leftarrow {\sim}\mathtt{AUX}(p)$.

R5. [Loop on p.] Set $p \leftarrow s$. Go to step R9 if $p = 0$. Otherwise set $q \leftarrow p$.

R6. [Examine a bucket.] Set $s \leftarrow \mathtt{LO}(p)$. (At this point $p = q$.)

R7. [Remove duplicates.] Set $r \leftarrow \mathtt{HI}(q)$. If $\mathtt{AUX}(r) \ge 0$, set $\mathtt{AUX}(r) \leftarrow {\sim}q$; otherwise set $\mathtt{LO}(q) \leftarrow \mathtt{AUX}(r)$, $\mathtt{HI}(q) \leftarrow \mathtt{AVAIL}$, and $\mathtt{AVAIL} \leftarrow q$. Then set $q \leftarrow \mathtt{AUX}(q)$. If $q \ne 0$ and $\mathtt{LO}(q) = s$, repeat step R7.

R8. [Clean up again.] If $\mathtt{LO}(p) \ge 0$, set $\mathtt{AUX(HI}(p)\mathtt{)} \leftarrow 0$. Then set $p \leftarrow \mathtt{AUX}(p)$, and repeat step R8 until $p = q$.

R9. [Done?] If $p \ne 0$, return to R6. Otherwise, if $v > \mathtt{V(ROOT)}$, set $v \leftarrow v - 1$ and return to R3. Otherwise, if $\mathtt{LO(ROOT)} < 0$, set $\mathtt{ROOT} \leftarrow {\sim}\mathtt{LO(ROOT)}$. ▮

The intricate link manipulations of Algorithm R are easier to program than to explain, but they are highly instructive and not really difficult. The reader is urged to work through the example in exercise 53.

Algorithm R can also be used to compute the BDD for any *restriction* of a given function, namely for any function obtained by "hardwiring" one or more variables to a constant value. The idea is to do a little extra work between steps R1 and R2, setting $\mathtt{HI}(p) \leftarrow \mathtt{LO}(p)$ if variable $\mathtt{V}(p)$ is supposed to be fixed at 0, or $\mathtt{LO}(p) \leftarrow \mathtt{HI}(p)$ if $\mathtt{V}(p)$ is to be fixed at 1. We also need to recycle all nodes that become inaccessible after restriction. Exercise 57 fleshes out the details.

restriction of a Boolean function
restriction, see also subfunctions
replacement of variables by constants
substitution of constants for variables
melding+
notation $\alpha \diamond \alpha'$

Synthesis of BDDs. We're ready now for the most important algorithm on binary decision diagrams, which takes the BDD for one function, f, and combines it with the BDD for another function, g, in order to obtain the BDD for further functions such as $f \wedge g$ or $f \oplus g$. Synthesis operations of this kind are the principal way to build up the BDDs for complex functions, and the fact that they can be done efficiently is the main reason why BDD data structures have become popular. We will discuss several approaches to the synthesis problem, beginning with a simple method and then speeding it up in various ways.

The basic notion that underlies synthesis is a product operation on BDD structures that we shall call *melding*. Suppose $\alpha = (v, l, h)$ and $\alpha' = (v', l', h')$ are BDD nodes, each containing the index of a variable together with LO and HI pointers. The "meld" of α and α', written $\alpha \diamond \alpha'$, is defined as follows when α and α' are not both sinks:

$$\alpha \diamond \alpha' \;=\; \begin{cases} (v,\ l \diamond l',\ h \diamond h'), & \text{if } v = v'; \\ (v,\ l \diamond \alpha',\ h \diamond \alpha'), & \text{if } v < v'; \\ (v',\ \alpha \diamond l',\ \alpha \diamond h'), & \text{if } v > v'. \end{cases} \tag{37}$$

For example, Fig. 24 shows how two small but typical BDDs are melded. The one on the left, with branch nodes $(\alpha, \beta, \gamma, \delta)$, represents $f(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$; the one in the middle, with branch nodes $(\omega, \psi, \chi, \varphi, \upsilon, \tau)$, represents $g(x_1, x_2, x_3, x_4) = (x_1 \oplus x_2) \vee (x_3 \oplus x_4)$. Nodes δ and τ are essentially the same, so we would have $\delta = \tau$ if f and g were part of a single BDD base; but melding can be applied also to BDDs that do not have common nodes. At the right of Fig. 24, $\alpha \diamond \omega$ is the root of a decision diagram that has eleven branch nodes, and it essentially represents the *ordered pair* (f, g).

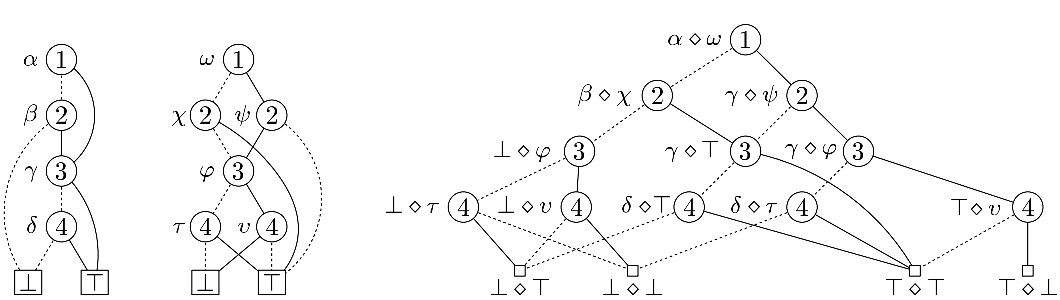
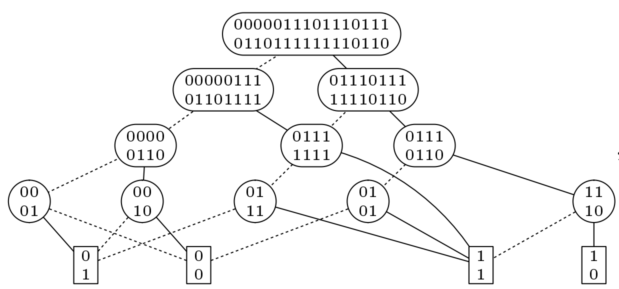


Fig. 24. Two BDDs can be melded together with the $\diamond$ operation (37).

An ordered pair of two Boolean functions can be visualized by placing the truth table of one above the truth table of the other. With this interpretation, $\alpha\diamond\omega$ stands for the ordered pair $\substack{0000011101110111\\0110111111110110}$, and $\beta\diamond\chi$ stands for $\substack{00000111\\01101111}$, etc. The melded BDD of Fig. 24 corresponds to the diagram



, (38)

which is analogous to (5) except that each node denotes an ordered pair of functions instead of a single function. Beads and subtables are defined on ordered pairs just as before. But now we have four possible sinks instead of two, namely

$$\bot\diamond\bot,\qquad \bot\diamond\top,\qquad \top\diamond\bot,\qquad \text{and}\qquad \top\diamond\top, \tag{39}$$

corresponding to the ordered pairs $\substack{0\\0}$, $\substack{0\\1}$, $\substack{1\\0}$, and $\substack{1\\1}$.

To compute the conjunction $f\wedge g$, we AND together the truth tables of f and g. This operation corresponds to replacing $\substack{0\\0}$, $\substack{0\\1}$, $\substack{1\\0}$, and $\substack{1\\1}$ by 0, 0, 0, and 1, respectively; so we get the BDD for $f\wedge g$ from $f\diamond g$ by replacing the respective sink nodes of (39) by $\boxed{\bot}$, $\boxed{\bot}$, $\boxed{\bot}$, and $\boxed{\top}$, then reducing the result. Similarly, the BDD for $f\oplus g$ is obtained if we replace the sinks (39) by $\boxed{\bot}$, $\boxed{\top}$, $\boxed{\top}$, and $\boxed{\bot}$. (In this particular case $f\oplus g$ turns out to be the symmetric function $S_{1,4}(x_1,x_2,x_3,x_4)$, as computed in Fig. 9 of Section 7.1.2.) The melded diagram $f\diamond g$ contains all the information needed to compute *any* Boolean combination of f and g; and the BDD for every such combination has at most $B(f\diamond g)$ nodes.

Clearly $B(f\diamond g)\le B(f)B(g)$, because each node of $f\diamond g$ corresponds to a node of f and a node of g. Therefore the meld of small BDDs cannot be extremely large. Usually, in fact, melding produces a result that is considerably smaller than this worst-case upper bound, with something like $B(f)+B(g)$ nodes instead of $B(f)B(g)$. Exercise 60 discusses a sharper bound that sheds some light on why melds often turn out to be small. But exercises 59(b) and 63 present interesting examples where quadratic growth does occur.

Melding suggests a simple algorithm for synthesis: We can form an array of $B(f)B(g)$ nodes, with node $\alpha\diamond\alpha'$ in row α and column α' for every α in the BDD for f and every α' in the BDD for g. Then we can convert the four sink nodes (39) to $\boxed{\bot}$ or $\boxed{\top}$ as desired, and apply Algorithm R to the root node $f\diamond g$. Voilà—we've got the BDD for $f\wedge g$ or $f\oplus g$ or $f\vee\bar{g}$ or whatever.

The running time of this algorithm is clearly of order $B(f)B(g)$. We can reduce it to order $B(f\diamond g)$, because there's no need to fill in all of the matrix entries $\alpha\diamond\alpha'$; only the nodes that are reachable from $f\diamond g$ are relevant, and we can generate them on the fly when necessary. But even with this improvement in the

ordered pair of two Boolean functions
truth table
Beads
subtables
sinks
conjunction
symmetric function

running time, the simple algorithm is unsatisfactory because of the requirement for $B(f)B(g)$ nodes in memory. When we deal with BDDs, time is cheap but space is expensive: Attempts to solve large problems tend to fail more often because of "spaceout" than because of "timeout." That's why Algorithm R was careful to perform its machinations with only one auxiliary link field per node.

time versus space
space versus time
Boolean function calculator
sequential stack
pool
templates+++
binary operators
truth table
op+

The following algorithm solves the synthesis problem with working space of order $B(f \diamond g)$; in fact, it needs only about sixteen bytes per element of the BDD for $f \diamond g$. The algorithm is designed to be used as the main engine of a "Boolean function calculator," which represents functions as BDDs in compressed form on a sequential stack. The stack is maintained at the lower end of a large array called the *pool*. Each BDD on the stack is a sequence of *nodes*, which each have three fields $(\mathtt{V}, \mathtt{LO}, \mathtt{HI})$. The rest of the pool is available to hold temporary results called *templates*, which each have four fields $(\mathtt{L}, \mathtt{H}, \mathtt{LEFT}, \mathtt{RIGHT})$. A node typically occupies one octabyte of memory, while a template occupies two.

The purpose of Algorithm S is to examine the top two Boolean functions on the stack, f and g, and to replace them by the Boolean combination $f \circ g$, where $\circ$ is one of the 16 possible binary operators. This operator is identified by its 4-bit truth table, *op*. For example, Algorithm S will form the BDD for $f \oplus g$ when *op* is $(0110)_2 = 6$; it will deliver $f \wedge g$ when $op = 1$.

When the algorithm begins, operand f appears in locations $[f_0 \mathinner{..} g_0)$ of the pool, and operand g appears in locations $[g_0 \mathinner{..} \mathtt{NTOP})$. All higher locations $[\mathtt{NTOP} \mathinner{..} \mathtt{POOLSIZE})$ are available for storing the templates that the algorithm needs. Those templates will appear in locations $[\mathtt{TBOT} \mathinner{..} \mathtt{POOLSIZE})$ at the high end of the pool; the boundary markers NTOP and TBOT will change dynamically as the algorithm proceeds. The resulting BDD for $f \circ g$ will eventually be placed in locations $[f_0 \mathinner{..} \mathtt{NTOP})$, taking over the space formerly occupied by f and g. We assume that a template occupies the space of two nodes. Thus, the assignments "$t \leftarrow \mathtt{TBOT} - 2$, $\mathtt{TBOT} \leftarrow t$" allocate space for a new template, pointed to by t; the assignments "$p \leftarrow \mathtt{NTOP}$, $\mathtt{NTOP} \leftarrow p + 1$" allocate a new node p. For simplicity of exposition, Algorithm S does not check that the condition $\mathtt{NTOP} \le \mathtt{TBOT}$ remains valid throughout the process; but of course such tests are essential in practice. Exercise 69 remedies this oversight.

The input functions f and g are specified to Algorithm S as sequences of instructions $(I_{s-1}, \ldots, I_1, I_0)$ and $(I'_{s'-1}, \ldots, I'_1, I'_0)$, as in Algorithms B and C above. The lengths of these sequences are $s = B^+(f)$ and $s' = B^+(g)$, where

$$B^+(f) \;=\; B(f) + [f \text{ is identically } 1] \tag{40}$$

is the number of BDD nodes when the sink $\boxed{\perp}$ is forced to be present. For example, the two BDDs at the left of Fig. 24 could be specified by the instructions

$$\begin{matrix} I_5 = (\bar{1}?\ 4\colon 3), & I_3 = (\bar{3}?\ 2\colon 1), \\ I_4 = (\bar{2}?\ 0\colon 3), & I_2 = (\bar{4}?\ 0\colon 1); \end{matrix} \qquad \begin{matrix} I'_7 = (\bar{1}?\ 5\colon 6), & I'_4 = (\bar{3}?\ 2\colon 3), \\ I'_6 = (\bar{2}?\ 1\colon 4), & I'_3 = (\bar{4}?\ 1\colon 0), \\ I'_5 = (\bar{2}?\ 4\colon 1), & I'_2 = (\bar{4}?\ 0\colon 1); \end{matrix} \tag{41}$$

as usual, I_1, I_0, I'_1, and I'_0 are the sinks. These instructions are packed into nodes, so that if $I_k = (\bar{v}_k?\ l_k\colon h_k)$ we have $\mathtt{V}(f_0 + k) = v_k$, $\mathtt{LO}(f_0 + k) = l_k$, and

clone
trick
hash table++
binary search trees

$\mathtt{HI}(f_0+k) = h_k$ for $2 \le k < s$ when Algorithm S begins. Similar conventions apply to the instructions I'_k that define g. Furthermore

$$\mathtt{V}(f_0) = \mathtt{V}(f_0+1) = \mathtt{V}(g_0) = \mathtt{V}(g_0+1) = v_{\max}+1, \tag{42}$$

where we assume that f and g depend only on the variables x_v for $1 \le v \le v_{\max}$.

Like the simple but space-hungry algorithm described earlier, Algorithm S proceeds in two phases: First it builds the BDD for $f \diamond g$, constructing templates so that every important meld $\alpha \diamond \alpha'$ is represented as a template t for which

$$\mathtt{LEFT}(t) = \alpha,\ \mathtt{RIGHT}(t) = \alpha',\ \ \mathtt{L}(t) = \mathrm{LO}(\alpha \diamond \alpha'),\ \mathtt{H}(t) = \mathrm{HI}(\alpha \diamond \alpha'). \tag{43}$$

(The L and H fields point to templates, not nodes.) Then the second phase reduces these templates, using a procedure similar to Algorithm R; it changes template t from (43) to

$$\begin{gathered}\mathtt{LEFT}(t) = {\sim}\kappa(t),\ \ \mathtt{RIGHT}(t) = \tau(t),\\ \mathtt{L}(t) = \tau(\mathrm{LO}(\alpha \diamond \alpha')),\ \mathtt{H}(t) = \tau(\mathrm{HI}(\alpha \diamond \alpha')),\end{gathered} \tag{44}$$

where $\tau(t)$ is the unique template to which t has been reduced, and where $\kappa(t)$ is the "clone" of t if $\tau(t) = t$. Every reduced template t corresponds to an instruction node in the BDD of $f \circ g$, and $\kappa(t)$ is the index of this node relative to position f_0 in the stack. (Setting $\mathtt{LEFT}(t)$ to ${\sim}\kappa(t)$ instead of $\kappa(t)$ is a sneaky trick that makes steps S7–S10 run faster.) Special overlapping templates are permanently reserved for sinks at the *bottom* of the pool, so that we always have

$$\mathtt{LEFT}(0) = {\sim}0,\ \ \mathtt{RIGHT}(0) = 0,\ \ \mathtt{LEFT}(1) = {\sim}1,\ \ \mathtt{RIGHT}(1) = 1, \tag{45}$$

in accord with the conventions of (42) and (44).

We needn't make a template for $\alpha \diamond \alpha'$ when the value of $\alpha \circ \alpha'$ is obviously constant. For example, if we're computing $f \wedge g$, we know that $\alpha \diamond \alpha'$ will eventually reduce to $\boxed{\perp}$ if $\alpha = 0$ or $\alpha' = 0$. Such simplifications are discovered by a subroutine called *find_level*(f,g), which returns the positive integer j if the root of $f \diamond g$ begins with the branch (j), unless $f \circ g$ clearly has a constant value; in the latter case, *find_level*(f,g) returns the value $-(f \circ g)$, which is 0 or -1. The procedure is slightly technical, but simple, using the global truth table *op*:

Subroutine *find_level*(f,g), with local variable t:
If $f \le 1$ and $g \le 1$, return $-((op \gg (3-2f-g))\ \&\ 1)$, which is $-(f \circ g)$.
If $f \le 1$ and $g > 1$, set $t \leftarrow (f?\ op\ \&\ 3\colon\ op \gg 2)$; return 0 if $t = 0$, -1 if $t = 3$.
If $f > 1$ and $g \le 1$, set $t \leftarrow (g?\ op\colon\ op \gg 1)\ \&\ 5$; return 0 if $t = 0$, -1 if $t = 5$.
Otherwise return $\min(\mathtt{V}(f_0+f), \mathtt{V}(g_0+g))$. (46)

The main difficulty that faces us, when generating a template for a descendant of $\alpha \diamond \alpha'$ according to (37), is to decide whether or not such a template already exists — and if so, to link to it. The best way to solve such problems is usually to use a hash table; but then we must decide where to put such a table, and how much extra space to devote to it. Alternatives such as binary search trees would be much easier to adapt to our purposes, but they would add an unwanted factor of $\log B(f \diamond g)$ to the running time. The synthesis problem can actually be solved in worst-case time and space $O(B(f \diamond g))$ by using a bucket sort method analogous to Algorithm R (see exercise 72); but that solution is complicated and somewhat awkward.

bucket sort
chaining
breadth-first synthesis+
locality of reference

Fortunately there's a nice way out of this dilemma, requiring almost no extra memory and only modestly complex code, if we generate the templates one level at a time. Before generating the templates for level l, we'll know the number N_l of templates to be requested on that level. So we can temporarily allocate space for 2^b templates at the top of the currently free area, where $b = \lceil \lg N_l \rceil$, and put new templates there while hashing into the same area. The idea is to use chaining with separate lists, as in Fig. 38 of Section 6.4; the `H` and `L` fields of our templates and potential templates play the roles of heads and links in that illustration, while the keys appear in (`LEFT`, `RIGHT`). Here's the logic, in detail:

Subroutine *make_template*(f,g), with local variable t:
Set $h \leftarrow \texttt{HBASE} + 2(((314159257f + 271828171g) \bmod 2^d) \gg (d-b))$, where d is a convenient upper bound on the size of a pointer (usually $d = 32$). Then set $t \leftarrow \texttt{H}(h)$. While $t \neq \Lambda$ and either $\texttt{LEFT}(t) \neq f$ or $\texttt{RIGHT}(t) \neq g$, set $t \leftarrow \texttt{L}(t)$. If $t = \Lambda$, set $t \leftarrow \texttt{TBOT} - 2$, $\texttt{TBOT} \leftarrow t$, $\texttt{LEFT}(t) \leftarrow f$, $\texttt{RIGHT}(t) \leftarrow g$, $\texttt{L}(t) \leftarrow \texttt{H}(h)$, and $\texttt{H}(h) \leftarrow t$. Finally, return the value t. (47)

The calling routine in steps S4 and S5 ensures that $\texttt{NTOP} \le \texttt{HBASE} \le \texttt{TBOT}$.

This breadth-first, level-at-a-time strategy for constructing the templates has an added payoff, because it promotes "locality of reference": Memory accesses tend to be confined to nearby locations that have recently been seen, hence controlled in such a way that cache misses and page faults are significantly reduced. Furthermore, the eventual BDD nodes placed on the stack will also appear in order, so that all branches on the same variable appear consecutively.

Algorithm S (*Breadth-first synthesis of BDDs*). This algorithm computes the BDD for $f \circ g$ as described above, using subroutines (46) and (47). Auxiliary arrays $\texttt{LSTART}[l]$, $\texttt{LCOUNT}[l]$, $\texttt{LLIST}[l]$, and $\texttt{HLIST}[l]$ are used for $0 \le l \le v_{\max}$.

S1. [Initialize.] Set $f \leftarrow g_0 - 1 - f_0$, $g \leftarrow \texttt{NTOP} - 1 - g_0$, and $l \leftarrow \mathit{find\_level}(f,g)$. See exercise 66 if $l \le 0$. Otherwise set $\texttt{LSTART}[l-1] \leftarrow \texttt{POOLSIZE}$, and $\texttt{LLIST}[k] \leftarrow \texttt{HLIST}[k] \leftarrow \Lambda$, $\texttt{LCOUNT}[k] \leftarrow 0$ for $l < k \le v_{\max}$. Set $\texttt{TBOT} \leftarrow \texttt{POOLSIZE} - 2$, $\texttt{LEFT(TBOT)} \leftarrow f$, and $\texttt{RIGHT(TBOT)} \leftarrow g$.

S2. [Scan the level-l templates.] Set $\texttt{LSTART}[l] \leftarrow \texttt{TBOT}$ and $t \leftarrow \texttt{LSTART}[l-1]$. While $t > \texttt{TBOT}$, schedule requests for future levels by doing the following:

Set $t \leftarrow t-2$, $f \leftarrow \texttt{LEFT}(t)$, $g \leftarrow \texttt{RIGHT}(t)$, $vf \leftarrow \texttt{V}(f_0+f)$, $vg \leftarrow \texttt{V}(g_0+g)$,
$ll \leftarrow \mathit{find\_level}((vf \le vg\,?\ \texttt{LO}(f_0+f) : f), (vf \ge vg\,?\ \texttt{LO}(g_0+g) : g))$,
$lh \leftarrow \mathit{find\_level}((vf \le vg\,?\ \texttt{HI}(f_0+f) : f), (vf \ge vg\,?\ \texttt{HI}(g_0+g) : g))$.

If $ll \le 0$, set $\texttt{L}(t) \leftarrow -ll$; otherwise set $\texttt{L}(t) \leftarrow \texttt{LLIST}[ll]$, $\texttt{LLIST}[ll] \leftarrow t$, $\texttt{LCOUNT}[ll] \leftarrow \texttt{LCOUNT}[ll] + 1$. If $lh \le 0$, set $\texttt{H}(t) \leftarrow -lh$; otherwise set $\texttt{H}(t) \leftarrow \texttt{HLIST}[lh]$, $\texttt{HLIST}[lh] \leftarrow t$, $\texttt{LCOUNT}[lh] \leftarrow \texttt{LCOUNT}[lh] + 1$.

S3. [Done with phase one?] Go to S6 if $l = v_{\max}$. Otherwise set $l \leftarrow l + 1$, and return to S2 if $\texttt{LCOUNT}[l] = 0$.

monotone-function function+
truth table

S4. [Initialize for hashing.] Set $b \leftarrow \lceil \lg \texttt{LCOUNT[}l\texttt{]} \rceil$, $\texttt{HBASE} \leftarrow \texttt{TBOT} - 2^{b+1}$, and $\texttt{H(HBASE} + 2k\texttt{)} \leftarrow \Lambda$ for $0 \le k < 2^b$.

S5. [Make the level-l templates.] Set $t \leftarrow \texttt{LLIST[}l\texttt{]}$. While $t \ne \Lambda$, set $s \leftarrow \texttt{L(}t\texttt{)}$, $f \leftarrow \texttt{LEFT(}t\texttt{)}$, $g \leftarrow \texttt{RIGHT(}t\texttt{)}$, $vf \leftarrow \texttt{V(}f_0 + f\texttt{)}$, $vg \leftarrow \texttt{V(}g_0 + g\texttt{)}$, $\texttt{L(}t\texttt{)} \leftarrow \mathit{make\_template}((vf \le vg?\ \texttt{LO(}f_0{+}f\texttt{)}\colon f), (vf \ge vg?\ \texttt{LO(}g_0{+}g\texttt{)}\colon g))$, $t \leftarrow s$. (We're half done.) Then set $t \leftarrow \texttt{HLIST[}l\texttt{]}$. While $t \ne \Lambda$, set $s \leftarrow \texttt{H(}t\texttt{)}$, $f \leftarrow \texttt{LEFT(}t\texttt{)}$, $g \leftarrow \texttt{RIGHT(}t\texttt{)}$, $vf \leftarrow \texttt{V(}f_0 + f\texttt{)}$, $vg \leftarrow \texttt{V(}g_0 + g\texttt{)}$, $\texttt{H(}t\texttt{)} \leftarrow \mathit{make\_template}((vf \le vg?\ \texttt{HI(}f_0{+}f\texttt{)}\colon f), (vf \ge vg?\ \texttt{HI(}g_0{+}g\texttt{)}\colon g))$, $t \leftarrow s$. (Now the other half is done.) Go back to step S2.

S6. [Prepare for phase two.] (At this point it's safe to obliterate the nodes of f and g, because we've built all the templates (43). Now we'll convert them to form (44). Note that $\texttt{V(}f_0\texttt{)} = \texttt{V(}f_0+1\texttt{)} = v_{\max}+1$.) Set $\texttt{NTOP} \leftarrow f_0 + 2$.

S7. [Bucket sort.] Set $t \leftarrow \texttt{LSTART[}l-1\texttt{]}$. Do the following while $t > \texttt{LSTART[}l\texttt{]}$:

Set $t \leftarrow t-2$, $\texttt{L(}t\texttt{)} \leftarrow \texttt{RIGHT(L(}t\texttt{))}$, and $\texttt{H(}t\texttt{)} \leftarrow \texttt{RIGHT(H(}t\texttt{))}$.
If $\texttt{L(}t\texttt{)} = \texttt{H(}t\texttt{)}$, set $\texttt{RIGHT(}t\texttt{)} \leftarrow \texttt{L(}t\texttt{)}$. (This branch is redundant.)
Otherwise set $\texttt{RIGHT(}t\texttt{)} \leftarrow -1$, $\texttt{LEFT(}t\texttt{)} \leftarrow \texttt{LEFT(L(}t\texttt{))}$, $\texttt{LEFT(L(}t\texttt{))} \leftarrow t$.

S8. [Restore clone addresses.] If $t = \texttt{LSTART[}l-1\texttt{]}$, set $t \leftarrow \texttt{LSTART[}l\texttt{]} - 2$ and go to S9. Otherwise, if $\texttt{LEFT(}t\texttt{)} < 0$, set $\texttt{LEFT(L(}t\texttt{))} \leftarrow \texttt{LEFT(}t\texttt{)}$. Set $t \leftarrow t+2$ and repeat step S8.

S9. [Done with level?] Set $t \leftarrow t{+}2$. If $t = \texttt{LSTART[}l-1\texttt{]}$, go to S12. Otherwise, if $\texttt{RIGHT(}t\texttt{)} \ge 0$ repeat step S9.

S10. [Examine a bucket.] (Suppose $\texttt{L(}t_1\texttt{)} = \texttt{L(}t_2\texttt{)} = \texttt{L(}t_3\texttt{)}$, where $t_1 > t_2 > t_3 = t$ and no other templates on level l have this L value. Then at this point we have $\texttt{LEFT(}t_3\texttt{)} = t_2$, $\texttt{LEFT(}t_2\texttt{)} = t_1$, $\texttt{LEFT(}t_1\texttt{)} < 0$, and $\texttt{RIGHT(}t_1\texttt{)} = \texttt{RIGHT(}t_2\texttt{)} = \texttt{RIGHT(}t_3\texttt{)} = -1$.) Set $s \leftarrow t$. While $s > 0$, do the following: Set $r \leftarrow \texttt{H(}s\texttt{)}$, $\texttt{RIGHT(}s\texttt{)} \leftarrow \texttt{LEFT(}r\texttt{)}$; if $\texttt{LEFT(}r\texttt{)} < 0$, set $\texttt{LEFT(}r\texttt{)} \leftarrow s$; and set $s \leftarrow \texttt{LEFT(}s\texttt{)}$. Finally set $s \leftarrow t$ again.

S11. [Make clones.] If $s < 0$, go back to step S9. Otherwise if $\texttt{RIGHT(}s\texttt{)} \ge 0$, set $s \leftarrow \texttt{LEFT(}s\texttt{)}$. Otherwise set $r \leftarrow \texttt{LEFT(}s\texttt{)}$, $\texttt{LEFT(H(}s\texttt{))} \leftarrow \texttt{RIGHT(}s\texttt{)}$, $\texttt{RIGHT(}s\texttt{)} \leftarrow s$, $q \leftarrow \texttt{NTOP}$, $\texttt{NTOP} \leftarrow q+1$, $\texttt{LEFT(}s\texttt{)} \leftarrow {\sim}(q - f_0)$, $\texttt{LO(}q\texttt{)} \leftarrow {\sim}\texttt{LEFT(L(}s\texttt{))}$, $\texttt{HI(}q\texttt{)} \leftarrow {\sim}\texttt{LEFT(H(}s\texttt{))}$, $\texttt{V(}q\texttt{)} \leftarrow l$, $s \leftarrow r$. Repeat step S11.

S12. [Loop on l.] Set $l \leftarrow l-1$. Return to S7 if $\texttt{LSTART[}l\texttt{]} < \texttt{POOLSIZE}$. Otherwise, if $\texttt{RIGHT(POOLSIZE} - 2\texttt{)} = 0$, set $\texttt{NTOP} \leftarrow \texttt{NTOP} - 1$ (because $f \circ g$ is identically 0). ▮

As usual, the best way to understand an algorithm like this is to trace through an example. Exercise 67 discusses what Algorithm S does when it is asked to compute $f \wedge g$, given the BDDs in (41).

Algorithm S can be used, for example, to construct the BDDs for interesting functions such as the "monotone-function function" $\mu_n(x_1, \ldots, x_{2^n})$, which is true if and only if $x_1 \ldots x_{2^n}$ is the truth table of a monotone function:

$$\mu_n(x_1, \ldots, x_{2^n}) = \bigwedge_{0 \le i \subseteq j < 2^n} [x_{i+1} \le x_{j+1}]. \tag{48}$$

mems
combinatorial explosion

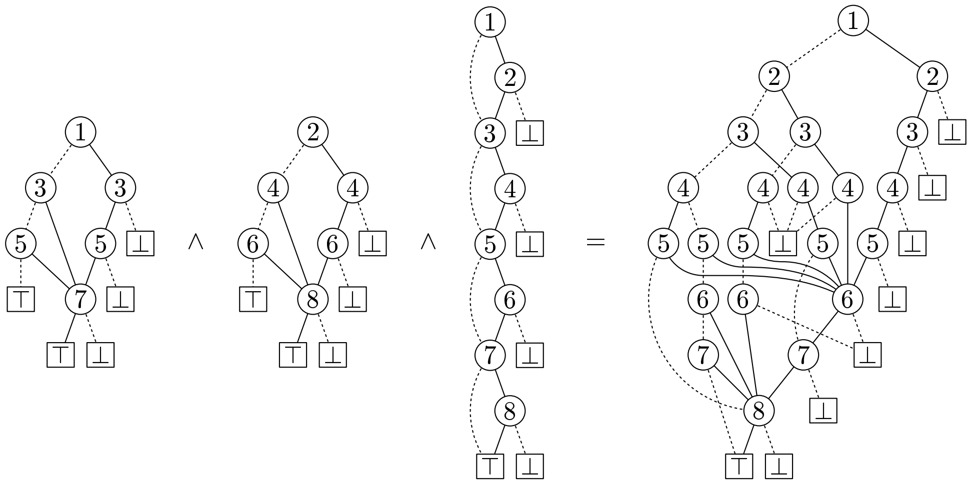


Fig. 25. $\mu_2(x_1,x_3,x_5,x_7) \wedge \mu_2(x_2,x_4,x_6,x_8) \wedge G_8(x_1,\ldots,x_8) \;=\; \mu_3(x_1,\ldots,x_8)$, as computed by Algorithm S.

Starting with $\mu_0(x_1) = 1$, this function satisfies the recursion relation

$$\mu_n(x_1,\ldots,x_{2^n}) \;= \\ \mu_{n-1}(x_1,x_3,\ldots,x_{2^n-1}) \wedge \mu_{n-1}(x_2,x_4,\ldots,x_{2^n}) \wedge G_{2^n}(x_1,\ldots,x_{2^n}), \quad (49)$$

where $G_{2^n}(x_1,\ldots,x_{2^n}) \;=\; [x_1 \le x_2] \wedge [x_3 \le x_4] \wedge \cdots \wedge [x_{2^n-1} \le x_{2^n}]$. So its BDD is easy to obtain with a BDD calculator like Algorithm S: The BDDs for $\mu_{n-1}(x_1,x_3,\ldots,x_{2^n-1})$ and $\mu_{n-1}(x_2,x_4,\ldots,x_{2^n})$ are simple variants of the one for $\mu_{n-1}(x_1,x_2,\ldots,x_{2^{n-1}})$, and G_{2^n} has an extremely simple BDD (see Fig. 25).

Repeating this process six times will produce the BDD for μ_6, which has 103,924 nodes. There are exactly 7,828,354 monotone Boolean functions of six variables (see exercise 5.3.4–31); this BDD nicely characterizes them all, and we need only about 4.8 million memory accesses to compute it with Algorithm S. Furthermore, 6.7 billion mems will suffice to compute the BDD for μ_7, which has 155,207,320 nodes and characterizes 2,414,682,040,998 monotone functions.

We must stop there, however; the size of the next case, $B(\mu_8)$, turns out to be a whopping 69,258,301,585,604 (see exercise 77).

Synthesis in a BDD base. Another approach is called for when we're dealing with many functions at once instead of computing a single BDD on the fly. The functions of a BDD base often share common subfunctions, as in (36). Algorithm S is designed to take disjoint BDDs and to combine them efficiently, afterwards destroying the originals; but in many cases we would rather form combinations of functions whose BDDs overlap. Furthermore, after forming a new function $f \wedge g$, say, we might want to keep f and g around for future use; indeed, the new function might well share nodes with f or g or both.

Let's therefore consider the design of a general-purpose toolkit for manipulating a collection of Boolean functions. BDDs are especially attractive for this purpose because most of the necessary operations have a simple recursive formulation. We know that every nonconstant Boolean function can be written

$$f(x_1, x_2, \ldots, x_n) \;=\; (\bar{x}_v?\ f_l\colon f_h), \tag{50}$$

where $v = f_v$ indexes the first variable on which f depends, and where we have

$$f_l = f(0, \ldots, 0, x_{v+1}, \ldots, x_n); \quad f_h = f(1, \ldots, 1, x_{v+1}, \ldots, x_n). \tag{51}$$

This rule corresponds to branch node (v) at the top of the BDD for f; and the rest of the BDD follows by using (50) and (51) recursively, until we reach constant functions that correspond to $\boxed{\bot}$ or $\boxed{\top}$. A similar recursion defines any combination of two functions, $f \circ g$: For if f and g aren't both constant, we have

$$f(x_1, \ldots, x_n) = (\bar{x}_v?\ f_l\colon f_h) \quad \text{and} \quad g(x_1, \ldots, x_n) = (\bar{x}_v?\ g_l\colon g_h), \tag{52}$$

where $v = \min(f_v, g_v)$ and where f_l, f_h, g_l, g_h are given by (51). Then, presto,

$$f \circ g \;=\; (\bar{x}_v?\ f_l \circ g_l\colon f_h \circ g_h). \tag{53}$$

This important formula is another way of stating the rule by which we defined melding, Eq. (37).

Caution: The notations above need to be understood carefully, because the subfunctions f_l and f_h in (50) might not be the same as the f_l and f_h in (52). Suppose, for example, that $f = x_2 \vee x_3$ while $g = x_1 \oplus x_3$. Then Eq. (50) holds with $f_v = 2$ and $f = (\bar{x}_2?\ f_l\colon f_h)$, where $f_l = x_3$ and $f_h = 1$. We also have $g_v = 1$ and $g = (\bar{x}_1?\ x_3\colon \bar{x}_3)$. But in (52) we use the same branch variable x_v for both functions, and $v = \min(f_v, g_v) = 1$ in our example; so Eq. (52) holds with $f = (\bar{x}_1?\ f_l\colon f_h)$ and $f_l = f_h = x_2 \vee x_3$.

Every node of a BDD base represents a Boolean function. Furthermore, a BDD base is reduced; therefore two of its functions or subfunctions are equal if and only if they correspond to exactly the same node. (This convenient uniqueness property was *not* true in Algorithm S.)

Formulas (51)–(53) immediately suggest a recursive way to compute $f \wedge g$:

$$\mathrm{AND}(f,g) = \begin{cases} \text{If } f \wedge g \text{ has an obvious value, return it.} \\ \text{Otherwise represent } f \text{ and } g \text{ as in (52);} \\ \text{compute } r_l \leftarrow \mathrm{AND}(f_l, g_l) \text{ and } r_h \leftarrow \mathrm{AND}(f_h, g_h); \\ \text{return the function } (\bar{x}_v?\ r_l\colon r_h). \end{cases} \tag{54}$$

(Recursions always need to terminate when a sufficiently simple case arises. The "obvious" values in the first line correspond to the terminal cases $f \wedge 1 = f$, $1 \wedge g = g$, $f \wedge 0 = 0 \wedge g = 0$, and $f \wedge g = f$ when $f = g$.) When f and g are the functions in our example above, (54) reduces $f \wedge g$ to the computation of $(x_2 \vee x_3) \wedge x_3$ and $(x_2 \vee x_3) \wedge \bar{x}_3$. Then $(x_2 \vee x_3) \wedge x_3$ reduces to $x_3 \wedge x_3$ and $1 \wedge x_3$; etc.

But (54) is problematic if we simply implement it as stated, because every nonterminal step launches two more instances of the recursion. The computation explodes, with 2^k instances of AND when we're k levels deep!

Fortunately there's a good way to avoid that blowup. Since f has only $B(f)$ different subfunctions, at most $B(f)B(g)$ distinctly different calls of AND can

recursive formulation
depth-first synthesis–
melding
reduced
equality testing of Boolean functions
sorcerer's apprentice
exponential growth

arise. To keep a lid on the computations, we just need to remember what we've done before, by making a *memo* of the fact that $f \wedge g = r$ just before returning r as the computed value. Then when the same subproblem occurs later, we can retrieve the memo and say, "Hey, we've already been there and done that." Previously solved cases thereby become terminal; only distinct subproblems can generate new ones. (Chapter 8 will discuss this memoization technique in detail.)

The algorithm in (54) also glosses over another problem: It's not so easy to "return the function $(\bar{x}_v?\ r_l\colon r_h)$," because we must keep the BDD base reduced. If $r_l = r_h$, we should return the node r_l; and if $r_l \neq r_h$, we need to decide whether the branch node $(\bar{x}_v?\ r_l\colon r_h)$ already exists, before creating a new one.

Thus we need to maintain additional information, besides the BDD nodes themselves. We need to keep memos of problems already solved; we also need to be able to find a node by its content, instead of by its address. The search algorithms of Chapter 6 now come to our rescue by telling us how to do both of these things, for example by hashing. To record a memo that $f \wedge g = r$, we can hash the key '$(f, \wedge, g)$' and associate it with the value r; to record the existence of an existing node $(\mathrm{V}, \mathrm{LO}, \mathrm{HI})$, we can hash the key '$(\mathrm{V}, \mathrm{LO}, \mathrm{HI})$' and associate it with that node's memory address.

The dictionary of all existing nodes $(\mathrm{V}, \mathrm{LO}, \mathrm{HI})$ in a BDD base is traditionally called the *unique table*, because we use it to enforce the all-important uniqueness criterion that forbids duplication. Instead of putting all that information into one giant dictionary, however, it turns out to be better to maintain a collection of smaller unique tables, one for each variable V. With such separate tables we can efficiently find all nodes that branch on a particular variable.

The memos are handy, but they aren't as crucial as the unique table entries. If we happen to forget the isolated fact that $f \wedge g = r$, we can always recompute it again later. Exponential blowup won't be worrisome, if the answers to the subproblems $f_l \wedge g_l$ and $f_h \wedge g_h$ are still remembered with high probability. Therefore we can use a less expensive method to store memos, designed to do a pretty-good-but-not-perfect job of retrieval: After hashing the key '$(f, \wedge, g)$' to a table position p, we need look for a memo only in that one position, not bothering to consider collisions with other keys. If several keys all share the same hash address, position p will record only the most recent relevant memo. This simplified scheme will still be adequate in practice, as long as the hash table is large enough. We shall call such a near-perfect table the *memo cache*, because it is analogous to the hardware caches by which a computer tries to remember significant values that it has dealt with in relatively slow storage units.

Okay, let's flesh out algorithm (54) by explicitly stating how it interacts with the unique tables and the memo cache:

$$\mathrm{AND}(f, g) = \begin{cases} \text{If } f \wedge g \text{ has an obvious value, return it.} \\ \text{Otherwise, if } f \wedge g = r \text{ is in the memo cache, return } r. \\ \text{Otherwise represent } f \text{ and } g \text{ as in (52);} \\ \text{compute } r_l \leftarrow \mathrm{AND}(f_l, g_l) \text{ and } r_h \leftarrow \mathrm{AND}(f_h, g_h); \\ \text{set } r \leftarrow \mathrm{UNIQUE}(v, r_l, r_h), \text{ using Algorithm U;} \\ \text{put `} f \wedge g = r \text{' into the memo cache, and return } r. \end{cases} \tag{55}$$

memoization
reduced
hashing
dictionary
unique table+
collisions
memo cache+
cache memory
computed table, see memo cache

Algorithm U (*Unique table lookup*). Given (v, p, q), where v is an integer while p and q point to nodes of a BDD base with variable rank $> v$, this algorithm returns a pointer to a node UNIQUE(v, p, q) that represents the function $(\bar{x}_v?\ p{:}\ q)$. A new node is added to the base if that function wasn't already present.

U1. [Easy case?] If $p = q$, return p.

U2. [Check the table.] Search variable x_v's unique table using the key (p, q). If the search successfully finds the value r, return r.

U3. [Create a node.] Allocate a new node r, and set `V`$(r) \leftarrow v$, `LO`$(r) \leftarrow p$, `HI`$(r) \leftarrow q$. Put r into x_v's unique table using the key (p, q). Return r. ▮

Notice that we needn't zero out the memo cache after finishing a top-level computation of AND(f, g). Each memo that we have made states a relationship between nodes of the structure; those facts are still valid, and they might be useful later when we want to compute AND(f, g) for new functions f and g.

A refinement of (55) will enhance that method further, namely to swap $f \leftrightarrow g$ if we discover that $f > g$ when $f \wedge g$ isn't obvious. Then we won't have to waste time computing $f \wedge g$ when we've already computed $g \wedge f$.

With simple changes to (55), the other binary operators OR(f, g), XOR(f, g), BUTNOT(f, g), NOR(f, g), ... can also be computed readily; see exercise 81.

The combination of (55) and Algorithm U looks considerably simpler than Algorithm S. Thus one might well ask, why should anybody bother to learn the other method? Its breadth-first approach seems quite complex by comparison with the "depth-first" order of computation in the recursive structure of (55); yet Algorithm S is able to deal only with BDDs that are disjoint, while Algorithm U and recursions like (55) apply to any BDD base.

Appearances can, however, be deceiving: Algorithm S has been described at a low level, with every change to every element of its data structures spelled out explicitly. By contrast, the high-level descriptions in (55) and Algorithm U assume that a substantial infrastructure exists behind the scenes. The memo cache and the unique tables need to be set up, and their sizes need to be carefully adjusted as the BDD base grows or contracts. When all is said and done, the total length of a program that implements Algorithms (55) and U properly "from scratch" is roughly ten times the length of a similar program for Algorithm S.

Indeed, the maintenance of a BDD base involves interesting questions of dynamic storage allocation, because we want to free up memory space when nodes are no longer accessible. Algorithm S solves this problem in a last-in-first-out manner, by simply keeping its nodes and templates on sequential stacks, and by making do with a single small hash table that can easily be integrated with the other data. A general BDD base, however, requires a more intricate system.

The best way to maintain a dynamic BDD base is probably to use *reference counters*, as discussed in Section 2.3.5, because BDDs are acyclic by definition. Therefore let's assume that every BDD node has a REF field, in addition to V, LO, and HI. The REF field tells us how many references exist to this node, either from LO or HI pointers in other nodes or from external root pointers F_j as in (36). For example, the REF fields for the nodes labeled 3 in (36) are respectively 4,

dead
monotone-function function
product
multiplication, binary

1, and 2; and all of the nodes labeled (2) or (4) or (6) in that example have REF = 1. Exercise 82 discusses the somewhat tricky issue of how to increase and decrease REF counts properly in the midst of a recursive computation.

A node becomes *dead* when its reference count becomes zero. When that happens, we should decrease the REF fields of the two nodes below it; and then they too might die in the same manner, recursively spreading the plague.

But a dead node needn't be removed from memory immediately. It still represents a potentially useful Boolean function, and we might discover that we need that function again as our computation proceeds. For example, we might find a dead node in step U2, because pointers from the unique table don't get counted as references. Likewise, in (55), we might accidentally stumble across a cache memo telling us that $f \wedge g = r$, when r is currently dead. In such cases, node r comes back to life. (And we must increase the REF counts of its LO and HI descendants, possibly resurrecting them recursively in the same fashion.)

Periodically, however, we will want to reclaim memory space by removing the deadbeats. Then we must do two things: We must purge all memos from the cache for which either f, g, or r is dead; and we must remove all dead nodes from memory and from their unique tables. See exercise 84 for typical heuristic strategies by which an automated system might decide when to invoke such cleanups and when to resize the tables dynamically.

Because of the extra machinery that is needed to support a BDD base, Algorithm U and top-down recursions like (55) cannot be expected to match the efficiency of Algorithm S on one-shot examples such as the monotone-function function μ_n in (49). The running time is approximately quadrupled when the more general approach is applied to this example, and the memory requirement grows by a factor of about 2.4.

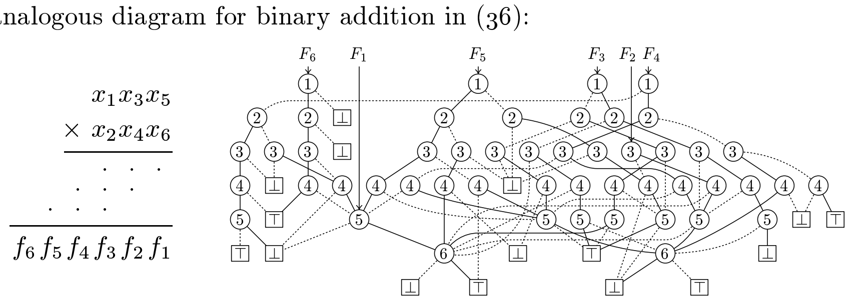
But a BDD base really begins to shine in numerous other applications. Suppose, for example, that we want the formulas for each bit of the product of two binary numbers,

$$(z_1 \ldots z_{m+n})_2 \;=\; (x_1 \ldots x_m)_2 \times (y_1 \ldots y_n)_2. \qquad (56)$$

Clearly $z_1 \ldots z_m = 0 \ldots 0$ when $n = 0$, and the simple recurrence

$$(x_1 \ldots x_m)_2 \times (y_1 \ldots y_n y_{n+1})_2 \;=\; (z_1 \ldots z_{m+n}0)_2 + (x_1 \ldots x_m)_2 y_{n+1} \qquad (57)$$

allows us to increase n by 1. This recurrence is easy to code for a BDD base. Here's what we get when $m = n = 3$, with subscripts chosen to match the analogous diagram for binary addition in (36):



(58)

Clearly multiplication is much more complicated than addition, bitwise. (Indeed, if it weren't, factorization wouldn't be so hard.) The corresponding BDD base for binary multiplication when $m = n = 16$ is huge, with $B(f_1, \ldots, f_{32}) =$ 136,398,751 nodes. It can be found after doing about 56 gigamems of calculation with Algorithm U, in 6.3 gigabytes of memory — including some 1.9 billion invocations of recursive subroutines, with hundreds of dynamic resizings of the unique tables and the memo cache, plus dozens of timely garbage collections. A similar calculation with Algorithm S would be almost unthinkable, although the individual functions in this particular example do not share many common subfunctions: It turns out that $B(f_1) + \cdots + B(f_{32}) = 168{,}640{,}131$, with the maximum occurring at the "middle bit," $B(f_{16}) = 38{,}174{,}143$.

gigamems
common subfunctions
middle bit
ternary operations+
multiplexing

***Ternary operations.** Given three Boolean functions $f = f(x_1, \ldots, x_n)$, $g = g(x_1, \ldots, x_n)$, and $h = h(x_1, \ldots, x_n)$, not all constant, we can generalize (52) to

$$f = (\bar{x}_v?\ f_l\colon f_h) \quad \text{and} \quad g = (\bar{x}_v?\ g_l\colon g_h) \quad \text{and} \quad h = (\bar{x}_v?\ h_l\colon h_h), \tag{59}$$

by taking $v = \min(f_v, g_v, h_v)$. Then, for example, (53) generalizes to

$$\langle fgh \rangle \;=\; \bigl(\bar{x}_v?\ \langle f_l g_l h_l \rangle\colon \langle f_h g_h h_h \rangle\bigr); \tag{60}$$

and similar formulas hold for *any* ternary operation on f, g, and h, including

$$(\bar{f}?\ g\colon h) \;=\; \bigl(\bar{x}_v?\ (\bar{f}_l?\ g_l\colon h_l)\colon (\bar{f}_h?\ g_h\colon h_h)\bigr). \tag{61}$$

(The reader of these formulas will please forgive the two meanings of 'h' in 'h_h'.)

Now it's easy to generalize (55) to ternary combinations like multiplexing:

$$\mathrm{MUX}(f,g,h) = \begin{cases} \text{If } (\bar{f}?\ g\colon h) \text{ has an obvious value, return it.} \\ \text{Otherwise, if } (\bar{f}?\ g\colon h) = r \text{ is in the memo cache, return } r. \\ \text{Otherwise represent } f,\ g, \text{ and } h \text{ as in (59);} \\ \text{compute } r_l \leftarrow \mathrm{MUX}(f_l, g_l, h_l) \text{ and } r_h \leftarrow \mathrm{MUX}(f_h, g_h, h_h); \\ \text{set } r \leftarrow \mathrm{UNIQUE}(v, r_l, r_h), \text{ using Algorithm U;} \\ \text{put `}(\bar{f}?\ g\colon h) = r\text{' into the memo cache, and return } r. \end{cases} \tag{62}$$

(See exercises 86 and 87.) The running time is $O\bigl(B(f)B(g)B(h)\bigr)$. The memo cache must now be consulted with a more complex key than before, including *three* pointers (f, g, h) instead of two, together with a code for the relevant operation. But each memo (op, f, g, h, r) can still be represented conveniently in, say, two octabytes, if the number of distinct pointer addresses is at most 2^{31}.

The ternary operation $f \wedge g \wedge h$ is an interesting special case. We could compute it with two invocations of (55), either as $\mathrm{AND}(f, \mathrm{AND}(g,h))$ or as $\mathrm{AND}(g, \mathrm{AND}(h,f))$ or as $\mathrm{AND}(h, \mathrm{AND}(f,g))$; or we could use a ternary subroutine, $\mathrm{ANDAND}(f,g,h)$, analogous to (62). This ternary routine first sorts the operands so that the pointers satisfy $f \le g \le h$. Then if $f = 0$, it returns 0; if $f = 1$ or $f = g$, it returns $\mathrm{AND}(g,h)$; if $g = h$ it returns $\mathrm{AND}(f,g)$; otherwise $1 < f < g < h$ and the operation remains ternary at the current level of recursion.

Suppose, for example, that $f = \mu_5(x_1, x_3, \ldots, x_{63})$, $g = \mu_5(x_2, x_4, \ldots, x_{64})$, and $h = G_{64}(x_1, \ldots, x_{64})$, as in Eq. (49). The computation $\mathrm{AND}(f, \mathrm{AND}(g,h))$

costs $0.2+6.8=7.0$ megamems in the author's experimental implementation; $\mathrm{AND}(g,\mathrm{AND}(h,f))$ costs $0.1+7.0=7.1$; $\mathrm{AND}(h,\mathrm{AND}(f,g))$ costs $24.4+5.6=30.0\,(!)$; and $\mathrm{ANDAND}(f,g,h)$ costs 7.5. So in this instance the all-binary approach wins, if we don't choose a bad order of computation. But sometimes ternary ANDAND beats all three of its binary competitors (see exercise 88).

ternary ANDAND
quantified formulas++
existential
universal quantification
notation: $\forall$
notation: $\exists$
Boolean matrix
Rudell

***Quantifiers.** If $f=f(x_1,\ldots,x_n)$ is a Boolean function and $1\le j\le n$, logicians traditionally define *existential and universal quantification* by the formulas

$$\exists x_j\, f(x_1,\ldots,x_n)=f_0\vee f_1 \quad\text{and}\quad \forall x_j\, f(x_1,\ldots,x_n)=f_0\wedge f_1, \tag{63}$$

where $f_c = f(x_1,\ldots,x_{j-1},c,x_{j+1},\ldots,x_n)$. Thus the quantifier '$\exists x_j$', pronounced "there exists x_j," changes f to the function of the remaining variables $(x_1,\ldots,x_{j-1},x_{j+1},\ldots,x_n)$ that is true if and only if at least one value of x_j satisfies $f(x_1,\ldots,x_n)$; the quantifier '$\forall x_j$', pronounced "for all x_j," changes f to the function that is true if and only if *both* values of x_j satisfy f.

Several quantifiers are often applied simultaneously. For example, the formula $\exists x_2\,\exists x_3\,\exists x_6\, f(x_1,\ldots,x_n)$ stands for the OR of eight terms, representing the eight functions of $(x_1,x_4,x_5,x_7,\ldots,x_n)$ that are obtained when we plug the values 0 or 1 into the variables x_2, x_3, and x_6 in all possible ways. Similarly, $\forall x_2\,\forall x_3\,\forall x_6\, f(x_1,\ldots,x_n)$ stands for the AND of those same eight terms.

One common application arises when the function $f(i_1,\ldots,i_l;j_1,\ldots,j_m)$ denotes the value in row $(i_1\ldots i_l)_2$ and column $(j_1\ldots j_m)_2$ of a $2^l\times 2^m$ Boolean matrix F. Then the function $h(i_1,\ldots,i_l;k_1,\ldots,k_n)$ given by

$$\exists j_1\ldots\exists j_m\big(f(i_1,\ldots,i_l;j_1,\ldots,j_m)\wedge g(j_1,\ldots,j_m;k_1,\ldots,k_n)\big) \tag{64}$$

represents the matrix H that is the Boolean product $F\,G$.

A convenient way to implement multiple quantification in a BDD base has been suggested by R. L. Rudell: Let $g=x_{j_1}\wedge\cdots\wedge x_{j_m}$ be a conjunction of positive literals. Then we can regard $\exists x_{j_1}\ldots\exists x_{j_m} f$ as the binary operation $f \mathbin{\mathrm{E}} g$, implemented by the following variant of (55):

$$\mathrm{EXISTS}(f,g)=\begin{cases}\text{If } f\mathbin{\mathrm{E}}g \text{ has an obvious value, return it.}\\ \text{Otherwise represent } f \text{ and } g \text{ as in (52);}\\ \text{if } v\ne f_v\text{, return EXISTS}(f,g_h).\\ \text{Otherwise, if } f\mathbin{\mathrm{E}}g=r \text{ is in the memo cache, return } r.\\ \text{Otherwise, } r_l\leftarrow\mathrm{EXISTS}(f_l,g_h) \text{ and } r_h\leftarrow\mathrm{EXISTS}(f_h,g_h);\\ \text{if } v\ne g_v\text{, set } r\leftarrow\mathrm{UNIQUE}(v,r_l,r_h) \text{ using Algorithm U,}\\ \text{otherwise compute } r\leftarrow\mathrm{OR}(r_l,r_h);\\ \text{put `} f\mathbin{\mathrm{E}}g=r\text{' into the memo cache, and return } r.\end{cases} \tag{65}$$

(See exercise 94.) The E operation is undefined when g does *not* have the stated form. Notice how the memo cache nicely remembers existential computations that have gone before.

The running time of (65) is highly variable — not like (55) where we know that $O(B(f)B(g))$ is the worst possible case — because m OR operations are invoked when g specifies m-fold quantification. The worst case now can be as bad as order $B(f)^{2^m}$, if all of the quantification occurs near the root of the BDD for f; this is only $O(B(f)^2)$ if $m = 1$, but it might become unbearably large as m grows. On the other hand, if all of the quantification occurs near the sinks, the running time is simply $O(B(f))$, regardless of the size of m. (See exercise 97.)

Several other quantifiers are worthy of note, and equally easy, although they aren't as famous as $\exists$ and $\forall$. The *Boolean difference* and the *yes/no quantifiers* are defined by formulas analogous to (63):

$$\text{ᗡ}x_j\, f = f_0 \oplus f_1; \qquad \text{⅄}x_j\, f = \bar{f}_0 \wedge f_1; \qquad \mathsf{N}x_j\, f = f_0 \wedge \bar{f}_1. \tag{66}$$

The Boolean difference, ᗡ, is the most important of these: $\text{ᗡ}x_j\, f$ is true for all values of $\{x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n\}$ such that f depends on x_j. If the multilinear representation of f is $f = (x_j g + h) \bmod 2$, where g and h are multilinear polynomials in $\{x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n\}$, then $\text{ᗡ}x_j\, f = g \bmod 2$. (See Eq. 7.1.1–(19).) Thus ᗡ acts like a derivative in calculus, over a finite field.

A Boolean function $f(x_1, \ldots, x_n)$ is monotone (nondecreasing) if and only if $\bigvee_{j=1}^{n} \mathsf{N}x_j\, f = 0$, which is the same as saying that $\mathsf{N}x_j\, f = 0$ for all j. However, exercise 105 presents a faster way to test a BDD for monotonicity.

Let's consider now a detailed example of existential quantification that is particularly instructive. If G is any graph, we can form Boolean functions $\mathrm{IND}(x)$ and $\mathrm{KER}(x)$ for its independent sets and kernels as follows, where x is a bit vector with one entry x_v for each vertex v of G:

$$\mathrm{IND}(x) = \neg \bigvee_{u \,-\!\!-\, v} (x_u \wedge x_v); \qquad \mathrm{KER}(x) = \mathrm{IND}(x) \wedge \bigwedge_{v} \Bigl(x_v \vee \bigvee_{u \,-\!\!-\, v} x_u\Bigr). \tag{67}$$

We can form a new graph $\mathcal{G}$ whose vertices are the kernels of G, namely the vectors x such that $\mathrm{KER}(x) = 1$. Let's say that two kernels x and y are *adjacent* in $\mathcal{G}$ if they differ in just the two entries for u and v, where $(x_u, x_v) = (1, 0)$ and $(y_u, y_v) = (0, 1)$ and $u \,-\!\!-\, v$. In other words, kernels can be considered as certain ways to place markers on vertices of G; moving a marker from one vertex to a neighboring vertex produces an adjacent kernel. Formally we define

$$a(x) = [\nu(x) = 2] \wedge \neg\mathrm{IND}(x); \tag{68}$$

$$\mathrm{ADJ}(x, y) = a(x \oplus y) \wedge \mathrm{KER}(x) \wedge \mathrm{KER}(y). \tag{69}$$

Then $x \,-\!\!-\, y$ in $\mathcal{G}$ if and only if $\mathrm{ADJ}(x, y) = 1$.

Notice that, if $x = x_1 \ldots x_n$, the function $[\nu(x) = 2]$ is the symmetric function $S_2(x_1, \ldots, x_n)$. Furthermore $a(x \oplus y)$ has at most 3 times as many nodes as $a(x)$, if we interleave the variables zipperwise so that the branching order is $(x_1, y_1, \ldots, x_n, y_n)$. Thus $B(a)$ and $B(\mathrm{ADJ})$ will not be extremely large unless $B(\mathrm{IND})$ or $B(\mathrm{KER})$ is large. It's now easy to express the condition that x is an *isolated vertex* of $\mathcal{G}$ (a vertex of degree 0):

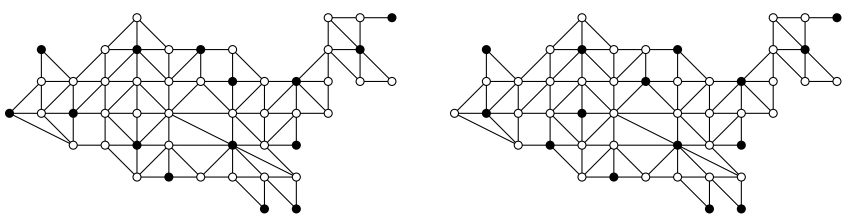
$$\mathrm{ISO}(x) \;=\; \mathrm{KER}(x) \wedge \neg\exists y\, \mathrm{ADJ}(x, y). \tag{70}$$

For example, suppose G is the graph of contiguous states in the USA, as in (18). Then each kernel vector x has 49 entries x_v for $v \in \{\mathtt{ME}, \mathtt{NH}, \ldots, \mathtt{CA}\}$. The graph $\mathcal{G}$ has 266,137 vertices, and we have observed earlier that the BDD sizes

Boolean difference
yes/no quantifiers
depends on
multilinear representation
monotone
independent sets+
kernels+
adjacent subsets of vertices
symmetric function
interleave
zipperwise
isolated vertices+
contiguous states+

implicit graphs
functional composition+
composition of functions+

for $\mathrm{IND}(x)$ and $\mathrm{KER}(x)$ are respectively 428 and 780 (see (17)). In this case the BDD sizes for $a(x)$ and $\mathrm{ADJ}(x,y)$ in (68) and (69) turn out to be only 286 and 7260, respectively, even though $\mathrm{ADJ}(x,y)$ is a function of 98 Boolean variables. The BDD for $\exists y\,\mathrm{ADJ}(x,y)$, which describes all kernels x of G that have at least one neighbor, turns out to have 842 nodes; and the one for $\mathrm{ISO}(x)$ has only 77. The latter BDD proves that graph G has exactly three isolated kernels, namely



(71)

and another that is a blend of these two. Using the algorithms above, this entire calculation, starting from a list of the vertices and edges of G (not $\mathcal{G}$), can be carried out with a total cost of about 4 megamems, in about 1.6 megabytes of memory; that's only about 15 memory accesses per kernel of G.

In a similar fashion we can use BDDs to work with other "implicit graphs," which have more vertices than could possibly be represented in memory, if those vertices can be characterized as the solution vectors of Boolean functions. When the functions aren't too complicated, we can answer queries about those graphs that could never be answered by representing the vertices and arcs explicitly.

***Functional composition.** The *pièce de résistance* of recursive BDD algorithms is a general procedure to compute $f(g_1, g_2, \ldots, g_n)$, where f is a given function of $\{x_1, x_2, \ldots, x_n\}$ and so is each argument g_j. Suppose we know a number $m \ge 0$ such that $g_j = x_j$ for $m < j \le n$; then the procedure can be expressed as follows:

$$\mathrm{COMPOSE}(f, g_1, \ldots, g_n) = \begin{cases} \text{If } f = 0 \text{ or } f = 1, \text{ return } f. \\ \text{Otherwise suppose } f = (\bar{x}_v?\ f_l\colon f_h), \text{ as in (50);} \\ \text{if } v > m, \text{ return } f; \text{ otherwise, if } f(g_1, \ldots, g_n) = r \\ \quad \text{is in the memo cache, return } r. \\ \text{Compute } r_l \leftarrow \mathrm{COMPOSE}(f_l, g_1, \ldots, g_n) \\ \quad \text{and } r_h \leftarrow \mathrm{COMPOSE}(f_h, g_1, \ldots, g_n); \\ \text{set } r \leftarrow \mathrm{MUX}(g_v, r_l, r_h) \text{ using (62);} \\ \text{put `} f(g_1, \ldots, g_n) = r \text{' into the cache, and return } r. \end{cases} \qquad (72)$$

The representation of cache memos like '$f(g_1, \ldots, g_n) = r$' in this algorithm is a bit tricky; we will discuss it momentarily.

Although the computations here look basically the same as those we've been seeing in previous recursions, there is in fact a huge difference: The functions r_l and r_h in (72) can now involve *all* variables $\{x_1, \ldots, x_n\}$, not just the x's near the bottom of the BDDs. So the running time of (72) might actually be huge. But there also are many cases when everything works together harmoniously and efficiently. For example, the computation of $a(x \oplus y)$ in (69) is no problem.

The key of a memo like '$f(g_1,\ldots,g_n) = r$' should not be a completely detailed specification of $(f, g_1, \ldots, g_n)$, because we want to hash it efficiently. Therefore we store only '$f[G] = r$', where G is an identification number for the sequence of functions $(g_1,\ldots,g_n)$. Whenever that sequence changes, we can use a new number G; and we can remember the G's for special sequences of functions that occur repeatedly in a particular computation, as long as the individual functions g_j don't die. (See also the alternative scheme in exercise 102.)

Let's return to the graph of contiguous states for one more example. That graph is planar; suppose we want to color it with four colors. Since the colors can be given 2-bit codes $\{00, 01, 10, 11\}$, it's easy to express the valid colorings as a Boolean function of 98 variables that is true if and only if the color codes ab are different for each pair of adjacent states:

$$\begin{aligned}\mathrm{COLOR}(a_{\mathtt{ME}}, b_{\mathtt{ME}}, \ldots, a_{\mathtt{CA}}, b_{\mathtt{CA}}) &= \\ &\mathrm{IND}(a_{\mathtt{ME}} \wedge b_{\mathtt{ME}}, \ldots, a_{\mathtt{CA}} \wedge b_{\mathtt{CA}}) \wedge \mathrm{IND}(a_{\mathtt{ME}} \wedge \bar b_{\mathtt{ME}}, \ldots, a_{\mathtt{CA}} \wedge \bar b_{\mathtt{CA}}) \\ &\quad \wedge \mathrm{IND}(\bar a_{\mathtt{ME}} \wedge b_{\mathtt{ME}}, \ldots, \bar a_{\mathtt{CA}} \wedge b_{\mathtt{CA}}) \wedge \mathrm{IND}(\bar a_{\mathtt{ME}} \wedge \bar b_{\mathtt{ME}}, \ldots, \bar a_{\mathtt{CA}} \wedge \bar b_{\mathtt{CA}}).\end{aligned} \tag{73}$$

Each of the four INDs has a BDD of 854 nodes, which can be computed via (72) with a cost of about 70 kilomems. The COLOR function turns out to have only 25,579 BDD nodes. Algorithm C now quickly establishes that the total number of ways to 4-color this graph is exactly 25,623,183,458,304 — or, if we divide by 4! to remove symmetries, about 1.1 trillion. The total time needed for this computation, starting from a description of the graph, is less than 3.5 megamems, in 2.2 megabytes of memory. (We can also find *random* 4-colorings, etc.)

Nasty functions. Of course there also are functions of 98 variables that aren't nearly so nice as COLOR. Indeed, the total number of 98-variable functions is $2^{2^{98}}$; exercise 108 proves that at most $2^{2^{46}}$ of them have a BDD size less than a trillion, and that almost all Boolean functions of 98 variables actually have $B(f) \approx 2^{98}/98 \approx 3.2 \times 10^{27}$. There's just no way to compress 2^{98} bits of data into a small space, unless that data happens to be highly redundant.

What's the worst case? If f is a Boolean function of n variables, how large can $B(f)$ be? The answer isn't hard to discover, if we consider the *profile* of a given BDD, which is the sequence $(b_0, \ldots, b_{n-1}, b_n)$ when there are b_k nodes that branch on variable x_{k+1} and b_n sinks. Clearly

$$B(f) = b_0 + \cdots + b_{n-1} + b_n. \tag{74}$$

We also have $b_0 \le 1$, $b_1 \le 2$, $b_2 \le 4$, $b_3 \le 8$, and in general

$$b_k \;\le\; 2^k, \tag{75}$$

because each node has only two branches. Furthermore $b_n = 2$ whenever f isn't constant; and $b_{n-1} \le 2$, because there are only two legal choices for the LO and HI branches of n. Indeed, we know that b_k is the number of *beads* of order $n-k$ in the truth table for f, namely the number of distinct subfunctions of $(x_{k+1}, \ldots, x_n)$ that depend on x_{k+1} after the values of $(x_1, \ldots, x_k)$ have been specified. Only $2^{2^m} - 2^{2^{m-1}}$ beads of order m are possible, so we must have

$$b_k \;\le\; 2^{2^{n-k}} - 2^{2^{n-k-1}}, \qquad \text{for } 0 \le k < n. \tag{76}$$

contiguous states
planar
colorings
4-color
–depth-first synthesis
random
profile+++
analysis of algorithms++
beads
truth table

When $n = 11$, for instance, (75) and (76) tell us that $(b_0, \ldots, b_{11})$ is at most

$$(1, 2, 4, 8, 16, 32, 64, 128, 240, 12, 2, 2). \tag{77}$$

Thus $B(f) \le 1 + 2 + \cdots + 128 + 240 + \cdots + 2 = 255 + 256 = 511$ when $n = 11$.

This upper bound is in fact obtained with the truth table

$$00000000\ 00000001\ 00000010\ \ldots\ 11111110\ 11111111, \tag{78}$$

or with any string of length 2^{11} that is a permutation of the 256 possible 8-bit bytes, because all of the 8-bit beads are clearly present, and because all of the subtables of lengths 16, 32, ..., 2^{11} are clearly beads. Similar examples can be constructed for all n (see exercise 110). Therefore the worst case is known:

truth table
subtables
beads
quasi-BDD
QDD

Theorem U. *Every Boolean function $f(x_1, \ldots, x_n)$ has $B(f) \le U_n$, where*

$$U_n = 2 + \sum_{k=0}^{n-1} \min(2^k, 2^{2^{n-k}} - 2^{2^{n-k-1}}) = 2^{n-\lambda(n-\lambda n)} + 2^{2^{\lambda(n-\lambda n)}} - 1. \tag{79}$$

Furthermore, explicit functions f_n with $B(f_n) = U_n$ exist for all n. ▮

If we replace λ by lg, the right-hand side of (79) becomes $2^n/(n - \lg n) + 2^n/n - 1$. In general, U_n is u_n times $2^n/n$, where the factor u_n lies between 1 and $2 + O(\frac{\log n}{n})$. A BDD with about $2^{n+1}/n$ nodes needs about $n + 1 - \lg n$ bits for each of two pointers in every node, plus $\lg n$ bits to indicate the variable for branching. So the total amount of memory space taken up by the BDD for any function $f(x_1, \ldots, x_n)$ is never more than about 2^{n+2} bits, which is four times the number of bits in its truth table, even if f happens to be one of the worst possible functions from the standpoint of BDD representation.

The average case turns out to be almost the same as the worst case, if we choose the truth table for f at random from among all 2^{2^n} possibilities. Again the calculations are straightforward: The average number of $\boxed{k+1}$ nodes is exactly

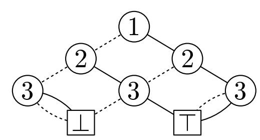
$$\hat{b}_k \;=\; \bigl(2^{2^{n-k}} - 2^{2^{n-k-1}}\bigr)\bigl(2^{2^n} - (2^{2^{n-k}} - 1)^{2^k}\bigr)/2^{2^n}, \tag{80}$$

because there are $2^{2^{n-k}} - 2^{2^{n-k-1}}$ beads of order $n - k$ and $(2^{2^{n-k}} - 1)^{2^k}$ truth tables in which any particular bead does not occur. Exercise 112 shows that this complicated-looking quantity $\hat{b}_k$ always lies extremely close to the worst-case estimate $\min(2^k,\ 2^{2^{n-k}} - 2^{2^{n-k-1}})$, except for two values of k. The exceptional levels occur when $k \approx 2^{n-k}$ and the "min" has little effect. For example, the average profile $(\hat{b}_0, \ldots, \hat{b}_{n-1}, \hat{b}_n)$ when $n = 11$ is approximately

$$(1.0, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0, 127.4, 151.9, 12.0, 2.0, 2.0) \tag{81}$$

when rounded to one decimal place, and these values are virtually indistinguishable from the worst case (77) except when $k = 7$ or 8.

A related concept called a *quasi-BDD*, or "QDD," is also important. Every function has a unique QDD, which is similar to its BDD except that the root node is always (1), and every (k) node for $k < n$ branches to two $\boxed{k+1}$ nodes; thus every path from the root to a sink has length n. To make this possible, we allow the LO and HI pointers of a QDD node to be identical. But the QDD must still be reduced, in the sense that different nodes cannot have the same two pointers (LO, HI). For example, the QDD for $\langle x_1 x_2 x_3 \rangle$ is



; (82)

it has two more nodes than the corresponding BDD in Fig. 21. Notice that the `V` fields are redundant in a QDD, so they needn't be present in memory.

The *quasi-profile* of a function is $(q_0, \ldots, q_{n-1}, q_n)$, where q_{k-1} is the number of (k) nodes in the QDD. It's easy to see that q_k is also the number of distinct *subtables* of order $n-k$ in the truth table, just as b_k is the number of distinct beads. Every bead is a subtable, so we have

$$q_k \;\geq\; b_k, \qquad \text{for } 0 \leq k \leq n. \tag{83}$$

Furthermore, exercise 115 proves that

$$q_k \leq 1 + b_0 + \cdots + b_{k-1} \text{ and } q_k \leq b_k + \cdots + b_n, \quad \text{for } 0 \leq k \leq n. \tag{84}$$

Consequently each element of the quasi-profile is a lower bound on the BDD size:

$$B(f) \;\geq\; 2q_k - 1, \qquad \text{for } 0 \leq k \leq n. \tag{85}$$

Let $Q(f) = q_0 + \cdots + q_{n-1} + q_n$ be the total size of the QDD for f. We obviously have $Q(f) \geq B(f)$, by (83). On the other hand $Q(f)$ can't be too much bigger than $B(f)$, because (84) implies that

$$Q(f) \;\leq\; \frac{n+1}{2}\bigl(B(f)+1\bigr). \tag{86}$$

Exercises 116 and 117 explore other basic properties of quasi-profiles.

The worst-case truth table (78) actually corresponds to a familiar function that we've already seen, the 8-way multiplexer

$$M_3(x_9, x_{10}, x_{11}; x_1, \ldots, x_8) \;=\; x_{1+(x_9 x_{10} x_{11})_2}. \tag{87}$$

But we've renumbered the variables perversely so that the multiplexing now occurs with respect to the *last* three variables (x_9, x_{10}, x_{11}), instead of the first three as in Eq. (30). This simple change to the ordering of the variables raises the BDD size of M_3 from 17 to 511; and an analogous change when $n = 2^m + m$ would cause $B(M_m)$ to make a colossal leap from $2n - 2m + 1$ to $2^{n-m+1} - 1$.

R. E. Bryant has introduced an interesting "navel-gazing" multiplexer called the *hidden weighted bit function*, defined as follows:

$$h_n(x_1, \ldots, x_n) \;=\; x_{x_1 + \cdots + x_n} \;=\; x_{\nu x}, \tag{88}$$

with the understanding that $x_0 = 0$. For example, $h_4(x_1, x_2, x_3, x_4)$ has the truth table 0000 0111 1001 1011. He proved [*IEEE Trans.* **C-40** (1991), 208–210] that h_n has a large BDD, regardless of how we might try to renumber its variables.

reduced
quasi-profile
subtables
truth table
beads
$Q(f)$
multiplexer
Bryant
omphaloskepsis
sideways addition

Fibonacci sequence
ϕ
exponential growth
plastic constant
permutation
notation f^π
nonstandard ordering

With the standard ordering of variables, the profile $(b_0, \ldots, b_{11})$ of h_{11} is

$$(1, 2, 4, 8, 15, 27, 46, 40, 18, 7, 2, 2); \tag{89}$$

hence $B(h_{11}) = 172$. The first half of this profile is actually the Fibonacci sequence in slight disguise, with $b_k = F_{k+4} - k - 2$. In general, h_n always has this value of b_k for $k < n/2$; thus its initial profile counts grow with order ϕ^k instead of the worst-case rate of 2^k. This growth rate slackens after k surpasses $n/2$, so that, for example, $B(h_{32})$ is only a modest 86,636. But exponential growth eventually takes over, and $B(h_{100})$ is out of sight: 17,530,618,296,680. (When $n = 100$, the maximum profile element is $b_{59} = 2{,}947{,}635{,}944{,}748$, which dwarfs $b_0 + \cdots + b_{49} = 139{,}583{,}861{,}115$.) Exercise 125 proves that $B(h_n)$ is asymptotically $c\chi^n + O(n^2)$, where

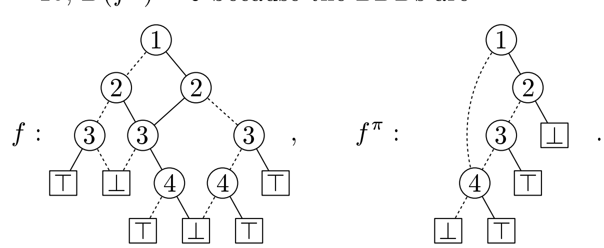
$$\begin{aligned}\chi &= \frac{\sqrt[3]{27-\sqrt{621}} + \sqrt[3]{27+\sqrt{621}}}{\sqrt[3]{54}}\\ &= 1.32471\,79572\,44746\,02596\,09088\,54478\,09734\,07344+\end{aligned} \tag{90}$$

is the so-called "plastic constant," the positive root of $\chi^3 = \chi + 1$, and the coefficient c is $7\chi - 1 + 14/(3 + 2\chi) \approx 10.75115$.

On the other hand we can do substantially better if we change the order in which the variables are tested in the BDD. If $f(x_1, \ldots, x_n)$ is any Boolean function and if π is any permutation of $\{1, \ldots, n\}$, let us write

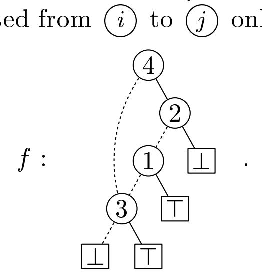
$$f^\pi(x_1, \ldots, x_n) \;=\; f(x_{1\pi}, \ldots, x_{n\pi}). \tag{91}$$

For example, if $f(x_1, x_2, x_3, x_4) = (x_3 \vee (x_1 \wedge x_4)) \wedge (\bar{x}_2 \vee \bar{x}_4)$ and if $(1\pi, 2\pi, 3\pi, 4\pi) = (3, 2, 4, 1)$, then $f^\pi(x_1, x_2, x_3, x_4) = (x_4 \vee (x_3 \wedge x_1)) \wedge (\bar{x}_2 \vee \bar{x}_1)$; and we have $B(f) = 10$, $B(f^\pi) = 6$ because the BDDs are



(92)

The BDD for f^π corresponds to a BDD for f that has a nonstandard ordering, in which a branch is permitted from (i) to (j) only if $i\pi < j\pi$:



(93)

The root is (i), where $i = 1\pi^-$ is the index for which $i\pi = 1$. When the branch variables are listed from the top down, we have $(4\pi, 2\pi, 1\pi, 3\pi) = (1, 2, 3, 4)$.

Applying these ideas to the hidden weighted bit function, we have

$$h_n^\pi(x_1,\ldots,x_n) \;=\; x_{(x_1+\cdots+x_n)\pi}, \tag{94}$$

with the understanding that $0\pi=0$ and $x_0=0$. For example, $h_3^\pi(0,0,1)=1$ if $(1\pi,2\pi,3\pi)=(3,1,2)$, because $x_{(x_1+x_2+x_3)\pi}=x_3=1$. (See exercise 120.)

Element q_k of the quasi-profile counts the number of distinct subfunctions that arise when the values of x_1 through x_k are known. Using (94), we can represent all such subfunctions by means of a *slate of options* $[r_0,\ldots,r_{n-k}]$, where r_j is the result of the subfunction when $x_{k+1}+\cdots+x_n=j$. Suppose $x_1=c_1,\ldots,x_k=c_k$, and let $s=c_1+\cdots+c_k$. Then $r_j=c_{(s+j)\pi}$ if $(s+j)\pi\le k$; otherwise $r_j=x_{(s+j)\pi}$. However, we set $r_0\leftarrow 0$ if $s\pi>k$, and $r_{n-k}\leftarrow 1$ if $(s+n-k)\pi>k$, so that the first and last options of every slate are constant.

quasi-profile
slate of options
Bryant

For example, calculations show that the following permutation $1\pi\ \ldots\ 100\pi$ reduces the BDD size of h_{100} from 17.5 trillion to $B(h_{100}^\pi)=1{,}124{,}432{,}105$:

2	4	6	8	10	12	14	16	18	20	97	57	77	37	87	47	67	27	92	52
72	32	82	42	62	22	100	60	80	40	90	50	70	30	95	55	75	35	85	45
65	25	98	58	78	38	88	48	68	28	93	53	73	33	83	43	63	23	99	59
79	39	89	49	69	29	94	54	74	34	84	44	64	24	96	56	76	36	86	46
66	26	91	51	71	31	81	41	61	21	19	17	15	13	11	9	7	5	3	1

(95)

Such calculations can be based on an enumeration of all slates that can arise, for $0\le s\le k\le n$. Suppose we've tested $x_1,\ldots,x_{83}$ and found that $x_j=[j\le 42]$, say, for $1\le j\le 83$. Then $s=42$; and the subfunction of the remaining 17 variables $(x_{84},\ldots,x_{100})$ is given by the slate $[r_0,\ldots,r_{17}]=[c_{25},x_{98},c_{58},c_{78},c_{38},x_{88},c_{48},c_{68},c_{28},x_{93},c_{53},c_{73},c_{33},c_{83},c_{43},c_{63},c_{23},x_{99}]$, which reduces to

$$[1,x_{98},0,0,1,x_{88},0,0,1,x_{93},0,0,1,0,0,0,1,1]. \tag{96}$$

This is one of the 2^{14} subfunctions counted by q_{83} when $s=42$. Exercise 124 explains how to deal similarly with the other values of k and s.

We're ready now to prove Bryant's theorem:

Theorem B. *The BDD size of h_n^π exceeds $2^{\lfloor n/5\rfloor}$, for all permutations π.*

Proof. Observe first that two subfunctions of h_n^π are equal if and only if they have the same slate. For if $[r_0,\ldots,r_{n-k}]\ne[r'_0,\ldots,r'_{n-k}]$, suppose $r_j\ne r'_j$. If both r_j and r'_j are constant, the subfunctions differ when $x_{k+1}+\cdots+x_n=j$. If r_j is constant but $r'_j=x_i$, we have $0<j<n-k$; the subfunctions differ because $x_{k+1}+\cdots+x_n$ can equal j with $x_i\ne r_j$. And if $r_j=x_i$ but $r'_j=x_{i'}$ with $i\ne i'$, we can have $x_{k+1}+\cdots+x_n=j$ with $x_i\ne x_{i'}$. (The latter case can arise only when the slates correspond to different offsets s and s'.)

Therefore q_k is the number of different slates $[r_0,\ldots,r_{n-k}]$. Exercise 123 proves that this number, for any given k, n, and s as described above, is exactly

$$\binom{w}{w-s}+\binom{w}{w-s+1}+\cdots+\binom{w}{k-s}=\binom{w}{s+w-k}+\cdots+\binom{w}{s-1}+\binom{w}{s}, \tag{97}$$

where w is the number of indices j such that $s\le j\le s+n-k$ and $j\pi\le k$.

Now consider the case $k=\lfloor 3n/5\rfloor+1$, and let $s=k-\lceil n/2\rceil$, $s'=\lfloor n/2\rfloor+1$. (Think of $n=100$, $k=61$, $s=11$, $s'=51$. We may assume that $n\ge 10$.) Then

$w + w' = k - w''$, where w'' counts the indices with $j\pi \le k$ and either $j < s$ or $j > s' + n - k$. Since $w'' \le (s-1) + (k-s') = 2k - 2 - n$, we must have $w + w' \ge n + 2 - k = \lceil 2n/5 \rceil + 1$. Hence either $w > \lfloor n/5 \rfloor$ or $w' > \lfloor n/5 \rfloor$; and in both cases (97) exceeds $2^{\lfloor n/5 \rfloor - 1}$. The theorem follows from (85). ▌

Conversely, there's always a permutation π such that $B(h_n^\pi) = O(2^{0.2029n})$, although the constant hidden by O-notation is quite large. This result was proved by B. Bollig, M. Löbbing, M. Sauerhoff, and I. Wegener, *Theoretical Informatics and Applications* **33** (1999), 103–115, using a permutation like (95): The first indices, with $j\pi \le n/5$, come alternately from $j > 9n/10$ and $j \le n/10$; the others are ordered by reading the binary representation of $9n/10 - j$ from right to left (*colex order*).

Let's also look briefly at a much simpler example, the *permutation function* $P_m(x_1, \ldots, x_{m^2})$, which equals 1 if and only if the binary matrix with $x_{(i-1)m+j}$ in row i and column j is a permutation matrix:

$$P_m(x_1, \ldots, x_{m^2}) \;=\; \bigwedge_{i=1}^{m} S_1(x_{(i-1)m+1}, x_{(i-1)m+2}, \ldots, x_{(i-1)m+m}) \;\wedge\; \bigwedge_{j=1}^{m} S_1(x_j, x_{m+j}, \ldots, x_{m^2-m+j}). \qquad (98)$$

In spite of its simplicity, this function cannot be represented with a small BDD, under any reordering of its variables:

Theorem K. *The BDD size of P_m^π exceeds $m2^{m-1}$, for all permutations π.*

Proof. [See I. Wegener, *Branching Programs and Binary Decision Diagrams* (SIAM, 2000), Theorem 4.12.3.] Given the BDD for P_m^π, notice that each of the $m!$ vectors x such that $P_m^\pi(x) = 1$ traces a path of length $n = m^2$ from the root to $\boxed{\top}$; every variable must be tested. Let $v_k(x)$ be the node from which the path for x takes its kth HI branch. This node branches on the value in row i and column j of the given matrix, for some pair $(i,j) = (i_k(x), j_k(x))$.

Suppose $v_k(x) = v_{k'}(x')$, where $x \ne x'$. Construct x'' by letting it agree with x up to $v_k(x)$ and with x' thereafter. Then $f(x'') = 1$; consequently we must have $k = k'$. In fact, this argument shows that we must also have

$$\{(i_1(x), j_1(x)), (i_2(x), j_2(x)), \ldots, (i_{k-1}(x), j_{k-1}(x))\} = \{(i_1(x'), j_1(x')), (i_2(x'), j_2(x')), \ldots, (i_{k-1}(x'), j_{k-1}(x'))\}. \qquad (99)$$

Imagine m colors of tickets, with $m!$ tickets of each color. Place a ticket of color k on node $v_k(x)$, for all k and all x. Then no node gets tickets of different colors; and no node of color k gets more than $(k-1)!\,(m-k)!$ tickets altogether, by Eq. (99). Therefore at least $m!/((k-1)!\,(m-k)!) = k\binom{m}{k}$ different nodes must receive tickets of color k. Summing over k gives $m2^{m-1}$ non-sink nodes. ▌

Exercise 184 shows that $B(P_m)$ is less than $m2^{m+1}$, so the lower bound in Theorem K is nearly optimum except for a factor of 4. Although the size grows exponentially, the behavior isn't hopelessly bad, because $m = \sqrt{n}$. For example, $B(P_{20})$ is only 38,797,317, even though P_{20} is a Boolean function of 400 variables.

O-notation
Bollig
Löbbing
Sauerhoff
Wegener
colex order
permutation function
0–1 matrices
permutation matrix
Wegener
exponential growth

optimizing the order of variables+
2^m-way multiplexer
subset function
organ-pipe order
master profile chart

Optimizing the order. Let $B_{\min}(f)$ and $B_{\max}(f)$ denote the smallest and largest values of $B(f^\pi)$, taken over all permutations π that can prescribe an ordering of the variables. We've seen several cases where $B_{\min}$ and $B_{\max}$ are dramatically different; for example, the 2^m-way multiplexer has $B_{\min}(M_m) \approx 2n$ and $B_{\max}(M_m) \approx 2^n/n$, when $n = 2^m + m$. And indeed, simple functions for which a good ordering is crucial are not at all unusual. Consider, for instance,

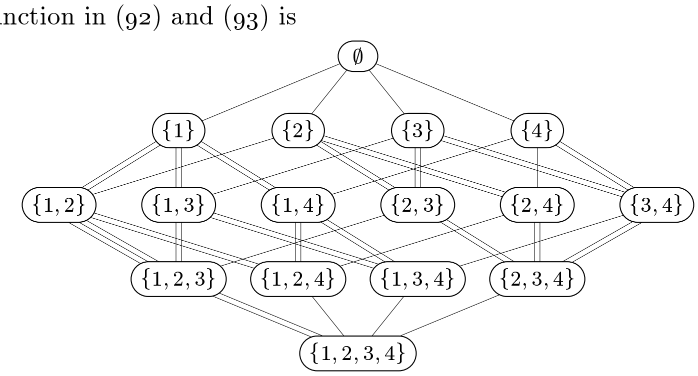
$$f(x_1, x_2, \ldots, x_n) \;=\; (\bar{x}_1 \vee x_2) \wedge (\bar{x}_3 \vee x_4) \wedge \cdots \wedge (\bar{x}_{n-1} \vee x_n), \quad n \text{ even}; \qquad (100)$$

this is the important *subset function* $[x_1 x_3 \ldots x_{n-1} \subseteq x_2 x_4 \ldots x_n]$, and we have $B(f) = B_{\min}(f) = n + 2$. But the BDD size explodes to $B(f^\pi) = B_{\max}(f) = 2^{n/2+1}$ when π is "organ-pipe order," namely the ordering for which

$$f^\pi(x_1, x_2, \ldots, x_n) \;=\; (\bar{x}_1 \vee x_n) \wedge (\bar{x}_2 \vee x_{n-1}) \wedge \cdots \wedge (\bar{x}_{n/2} \vee x_{n/2+1}). \qquad (101)$$

And the same bad behavior occurs for the ordering $[x_1 \ldots x_{n/2} \subseteq x_{n/2+1} \ldots x_n]$. In these orderings the BDD must "remember" the states of $n/2$ variables, while the original formulation (100) needs very little memory.

Every Boolean function f has a *master profile chart*, which encapsulates the set of all its possible sizes $B(f^\pi)$. If f has n variables, this chart has 2^n vertices, one for each subset of the variables; and it has $n2^{n-1}$ edges, one for each pair of subsets that differ in just one element. For example, the master profile chart for the function in (92) and (93) is



. (102)

Every edge has a weight, illustrated here by the number of lines; for example, the weight between $\{1,2\}$ and $\{1,2,3\}$ is 3. The chart has the following interpretation: *If X is a subset of k variables, and if $x \notin X$, then the weight between X and $X \cup x$ is the number of subfunctions of f that depend on x when the variables of X have been replaced by constants in all 2^k possible ways.* For example, if $X = \{1,2\}$, we have $f(0,0,x_3,x_4) = x_3$, $f(0,1,x_3,x_4) = f(1,1,x_3,x_4) = x_3 \wedge \bar{x}_4$, and $f(1,0,x_3,x_4) = x_3 \vee x_4$; all three of these subfunctions depend on x_3, but only two of them depend on x_4, as shown in the weights below $\{1,2\}$.

There are $n!$ paths of length n from $\emptyset$ to $\{1,\ldots,n\}$, and we can let the path $\emptyset \to \{a_1\} \to \{a_1,a_2\} \to \cdots \to \{a_1,\ldots,a_n\}$ correspond to the permutation π if $a_1\pi = 1$, $a_2\pi = 2$, …, $a_n\pi = n$. Then the sum of the weights on path π is $B(f^\pi)$, if we add 2 for the sink nodes. For example, the path $\emptyset \to \{4\} \to \{2,4\} \to \{1,2,4\} \to \{1,2,3,4\}$ yields the only way to achieve $B(f^\pi) = 6$ as in (93).

Notice that the master profile chart is a familiar graph, the n-cube, whose edges have been decorated so that they count the number of beads in various sets of subfunctions. The graph has exponential size, $n2^{n-1}$; yet it is much smaller than the total number of permutations, $n!$. When n is, say, 25 or less, exercise 138 shows that the entire chart can be computed without great difficulty, and we can find an optimum permutation for any given function. For example, the hidden weighted bit function turns out to have $B_{\min}(h_{25}) = 2090$ and $B_{\max}(h_{25}) = 35441$; the minimum is achieved with $(1\pi, \ldots, 25\pi) = (3, 5, 7, 9, 11, 13, 15, 17, 25, 24, 23, 22, 21, 20, 19, 18, 16, 14, 12, 10, 8, 6, 4, 2, 1)$, while the maximum results from a strange permutation $(22, 19, 17, 25, 15, 13, 11, 10, 9, 8, 7, 24, 6, 5, 4, 3, 2, 12, 1, 14, 23, 16, 18, 20, 21)$ that tests many "middle" variables first.

Instead of computing the entire master profile chart, we can sometimes save time by learning just enough about it to determine a path of least weight. (See exercise 140.) But when n grows and functions get more weird, we are unlikely to be able to determine $B_{\min}(f)$ exactly, because the problem of finding the best ordering is NP-complete (see exercise 137).

We've defined the profile and quasi-profile of a single Boolean function f, but the same ideas apply also to an arbitrary BDD base that contains m functions $\{f_1, \ldots, f_m\}$. Namely, the profile is $(b_0, \ldots, b_n)$ when there are b_k nodes on level k, and the quasi-profile is $(q_0, \ldots, q_n)$ when there are q_k nodes on level k of the corresponding QDD base; the truth tables of the functions have b_k different beads of order $n - k$, and q_k different subtables. For example, the profile of the $(4+4)$-bit addition functions $\{f_1, f_2, f_3, f_4, f_5\}$ in (36) is $(2, 4, 3, 6, 3, 6, 3, 2, 2)$, and the quasi-profile is worked out in exercise 144. Similarly, the concept of master profile chart applies to m functions whose variables are reordered simultaneously; and we can use it to find $B_{\min}(f_1, \ldots, f_m)$ and $B_{\max}(f_1, \ldots, f_m)$, the minimum and maximum of $b_0 + \cdots + b_n$ taken over all profiles.

***Local reordering.** What happens to a BDD base when we decide to branch on x_2 first, then on x_1, x_3, …, x_n? Figure 26 shows that the structure of the top two levels can change dramatically, but all other levels remain the same.

A closer analysis reveals, in fact, that this level-swapping process isn't difficult to understand or to implement. The 1 nodes before swapping can be divided into two kinds, "tangled" and "solitary," depending on whether they have 2 nodes as descendants; for example, there are three tangled nodes at the left of Fig. 26, pointed to by s_1, s_2, and s_3, while s_4 points to a solitary node. Similarly, the 2 nodes before swapping are either "visible" or "hidden," depending on whether they are independent source functions or accessible only from 1 nodes; all four of the 2 nodes at the left of Fig. 26 are hidden.

After swapping, the solitary 1 nodes simply move down one level, but the tangled nodes are transmogrified according to a process that we shall explain shortly. The hidden 2 nodes disappear, and the visible ones simply move up to the top level. Additional nodes might also arise during the transmogrification process; such nodes, labeled 1, are called "newbies." For example, two newbies appear at the right of Fig. 26. This process decreases the total number of nodes if and only if the hidden nodes outnumber the newbies.

n-cube
hidden weighted bit function
master profile chart
NP-complete
profile
quasi-profile
beads
subtables
$B_{\min}(f_1, \ldots, f_m)$
$B_{\max}(f_1, \ldots, f_m)$
swapping adjacent levels–
interchanging adjacent variables–
tangled nodes++
solitary nodes++
visible nodes++
hidden nodes++
transmogrification+
newbies++

sinks
extended truth table
Rudell

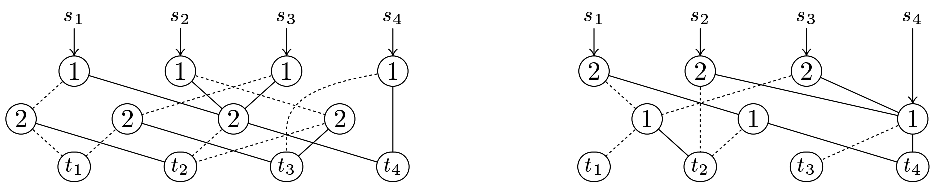


Fig. 26. Interchanging the top two levels of a BDD base. Here (s_1, s_2, s_3, s_4) are source functions; (t_1, t_2, t_3, t_4) are target nodes, representing subfunctions at lower levels.

The reverse of a swap is, of course, the same as a swap, but with the roles of (1) and (2) interchanged. If we begin with the diagram at the right of Fig. 26, we see that it has three tangled nodes (labeled (2)) and one that's visible (labeled (1)); two of its nodes are hidden, none are solitary. The swapping process in general sends (tangled, solitary, visible, hidden) nodes into (tangled, visible, solitary, newbie) nodes, respectively — after which newbies would become hidden in a reverse swap, and the originally hidden nodes would reappear as newbies.

Transmogrification is easiest to understand if we treat all nodes below the top two levels as if they were sinks, having constant values. Then every source function $f(x_1, x_2)$ depends only on x_1 and x_2; hence it takes on four values $a = f(0,0)$, $b = f(0,1)$, $c = f(1,0)$, and $d = f(1,1)$, where a, b, c, and d represent sinks. We may suppose that there are q sinks, $\boxed{1}$, $\boxed{2}$, ..., $\boxed{q}$, and that $1 \le a, b, c, d \le q$. Then $f(x_1, x_2)$ is fully described by its *extended truth table*, $f(0,0)f(0,1)f(1,0)f(1,1) = abcd$. And after swapping, we're left with $f(x_2, x_1)$, which has the extended truth table $acbd$. For example, Fig. 26 can be redrawn as follows, using extended truth tables to label its nodes:

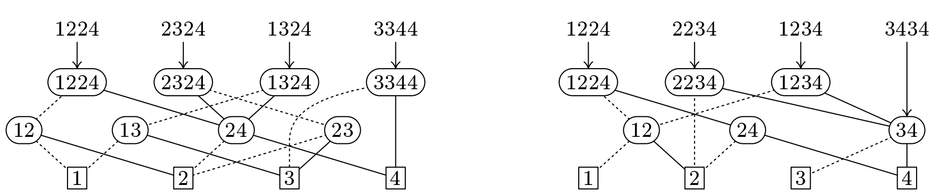


Fig. 27. Another way to represent the transformations in Fig. 26.

In these terms, the source function $abcd$ points to a solitary node when $a = b \ne c = d$, and to a visible node when $a = c \ne b = d$; otherwise it points to a tangled node (unless $a = b = c = d$, when it points directly to a sink). The tangled node $abcd$ usually has LO $= ab$ and HI $= cd$, unless $a = b$ or $c = d$; in the exceptional cases, LO or HI is a sink. After transmogrification it will have LO $= ac$ and HI $= bd$ in a similar way, where latter nodes will be either newbies or visibles or sinks (but not both sinks). One interesting case is 1224, whose children 12 and 24 on the left are hidden nodes, while the 12 and 24 on the right are newbies.

Exercise 147 discusses an efficient implementation of this transformation, which was introduced by Richard Rudell in *IEEE/ACM International Conf. Computer-Aided Design* **CAD-93** (1993), 42–47. It has the important property that no pointers need to change, except within the nodes on the top two levels:

All source nodes s_j still point to the same place in computer memory, and all sinks retain their previous identity. We have described it as a swap between ①s and ②s, but in fact the same transformation will swap (j)s and (k)s whenever the variables x_j and x_k correspond to branching on adjacent levels. The reason is that the upper levels of any BDD base essentially define source functions for the lower levels, which constitute a BDD base in their own right.

sorting
adjacent interchanges
jump-up
exponentially
sheep-and-goats
jump-down
zipper function
melding operation

We know from our study of sorting that *any* reordering of the variables of a BDD base can be produced by a sequence of swaps between adjacent levels. In particular, we can use adjacent swaps to do a "jump-up" transformation, which brings a given variable x_k to the top level without disturbing the relative order of the other variables. It's easy, for instance, to jump x_4 up to the top: We simply swap ④ $\leftrightarrow$ ③, then ④ $\leftrightarrow$ ②, then ④ $\leftrightarrow$ ①, because x_4 will be adjacent to x_1 after it has jumped past x_2.

Since repeated swaps can produce any ordering, they are sometimes able to make a BDD base grow until it is too big to handle. How bad can a single swap be? If exactly (s, t, v, h, ν) nodes are solitary, tangled, visible, hidden, and newbie, the top two levels end up with $s + t + v + \nu$ nodes; and this is at most $m + \nu \le m + 2t$ when there are m source functions, because $m \ge s + t + v$. Thus the new size can't exceed twice the original, plus the number of sources.

If a single swap can double the size, a jump-up for x_k threatens to increase the size exponentially, because it does $k-1$ swaps. Fortunately, however, jump-ups are no worse than single swaps in this regard:

Theorem J$^+$. *$B(f_1^\pi, \ldots, f_m^\pi) < m + 2B(f_1, \ldots, f_m)$ after a jump-up operation.*

Proof. Let $a_1a_2 \ldots a_{2^k-1}a_{2^k}$ be the extended truth table for a source function $f(x_1, \ldots, x_k)$, with lower-level nodes regarded as sinks. After the jump-up, the extended truth table for $f^\pi(x_1, \ldots, x_k) = f(x_{1\pi}, \ldots, x_{k\pi}) = f(x_2, \ldots, x_k, x_1)$ is $a_1a_3 \ldots a_{2^k-1}a_2a_4 \ldots a_{2^k}$, which incidentally can be written $a_1 \ldots a_{2^k} \mathbin{\cdot\!|\!\cdot} \mu_{k,0}$ in the "sheep-and-goats" notation of 7.1.3–(81). Thus we can see that each bead on level j of f^π is derived from some bead on level $j-1$ of f, for $1 \le j < k$; but every such bead spawns at most two beads of half the size in f^π. Therefore, if the respective profiles of $\{f_1, \ldots, f_m\}$ and $\{f_1^\pi, \ldots, f_m^\pi\}$ are $(b_0, \ldots, b_n)$ and $(b'_0, \ldots, b'_n)$, we must have $b'_0 \le m$, $b'_1 \le 2b_0$, $\ldots$, $b'_{k-1} \le 2b_{k-2}$, $b'_k = b_k$, $\ldots$, $b'_n = b_n$. The total is therefore $\le m + B(f_1, \ldots, f_m) + b_0 + \cdots + b_{k-2} - b_{k-1}$. ∎

The opposite of a jump-up is a "jump-down," which demotes the topmost variable by $k-1$ levels. As before, this operation can be implemented with $k-1$ swaps. But we have to settle for a much weaker upper bound on the resulting size:

Theorem J$^-$. *$B(f_1^\pi, \ldots, f_m^\pi) < B(f_1, \ldots, f_m)^2$ after a jump-down operation.*

Proof. Now the extended truth table in the previous proof changes from $a_1 \ldots a_{2^k}$ to $a_1 \ldots a_{2^{k-1}} \ddagger a_{2^{k-1}+1} \ldots a_{2^k} = a_1a_{2^{k-1}+1} \ldots a_{2^{k-1}}a_{2^k}$, the "zipper function" 7.1.3–(76). In this case we can identify every bead after the jump with an ordered pair of original subfunctions, as in the melding operation (37) and (38). For example, when $k = 3$ the truth table 12345678 becomes 15263748, whose bead 1526 can be regarded as the meld $12 \diamond 56$. ∎

2^m-way mux
Bollig
Löbbing
Wegener
dynamic reordering of variables+++
Rudell
sifting

This proof indicates why quadratic growth might occur. If, for example,

$$f(x_1,\ldots,x_n) \;=\; x_1?\; M_m(x_2,\ldots,x_{m+1};x_{2m+2},\ldots,x_n){:}\; M_m(x_{m+2},\ldots,x_{2m+1};\bar{x}_{2m+2},\ldots,\bar{x}_n), \quad (103)$$

where $n = 1 + 2m + 2^m$, a jump-down of $2m$ levels changes $B(f) = 4n - 8m - 3$ to $B(f^\pi) \;=\; 2n^2 - 8m(n-m) - 2(n-2m) + 1 \;\approx\; \frac{1}{2}B(f)^2$.

Since jump-up and jump-down are inverse operations, we can also use Theorems J^+ and J^- in reverse: *A jump-up operation might conceivably decrease the BDD size to something like its square root, but a jump-down cannot reduce the size to less than about half.* That's bad news for fans of jump-down, although they can take comfort from the knowledge that jump-downs are sometimes the only decent way to get from a given ordering to an optimum one.

Theorems J^+ and J^- are due to B. Bollig, M. Löbbing, and I. Wegener, *Inf. Processing Letters* **59** (1996), 233–239. (See also exercise 149.)

***Dynamic reordering.** In practice, a natural way to order the variables often suggests itself, based on the modules-in-a-row perspective of Fig. 23 and Theorem M. But sometimes no suitable ordering is apparent, and we can only hope to be lucky; perhaps the computer will come to our rescue and find one. Furthermore, even if we do know a good way to begin a computation, the ordering of variables that works best in the first stages of the work might turn out to be unsatisfactory in later stages. Therefore we can get better results if we don't insist on a fixed ordering. Instead, we can try to tune up the current order of branching whenever a BDD base becomes unwieldy.

For example, we might try to swap $x_{j-1} \leftrightarrow x_j$ in the order, for $1 < j \le n$, undoing the swap if it increases the total number of nodes but letting it ride otherwise; we could keep this up until no such swap makes an improvement. That method is easy to implement, but unfortunately it's too weak; it doesn't give much of a reduction. A much better reordering technique was proposed by Richard Rudell at the same time as he introduced the swap-in-place algorithm of exercise 147. His method, called "sifting," has proved to be quite successful. The idea is simply to take a variable x_k and to try jumping it up or down to all other levels — that is, essentially to remove x_k from the ordering and then to insert it again, choosing a place for insertion that keeps the BDD size as small as possible. All of the necessary work can be done with a sequence of elementary swaps:

Algorithm J (*Sifting a variable*). This algorithm moves variable x_k into an optimum position with respect to the current ordering of the other variables $\{x_1,\ldots,x_{k-1},x_{k+1},\ldots,x_n\}$ in a given BDD base. It works by repeatedly calling the procedure of exercise 147 to swap adjacent variables $x_{j-1} \leftrightarrow x_j$. Throughout this algorithm, S denotes the current size of the BDD base (the total number of nodes); the swapping operation usually changes S.

J1. [Initialize.] Set $p \leftarrow 0$, $j \leftarrow k$, and $s \leftarrow S$. If $k > n/2$, go to J5.

J2. [Sift up.] While $j > 1$, swap $x_{j-1} \leftrightarrow x_j$ and set $j \leftarrow j-1$, $s \leftarrow \min(S,s)$.

Rudell
contiguous USA++

J3. [End the pass.] If $p = 1$, go to J4. Otherwise, while $j \neq k$, set $j \leftarrow j+1$ and swap $x_{j-1} \leftrightarrow x_j$; then set $p \leftarrow 1$ and go to J5.

J4. [Finish downward.] While $s \neq S$, set $j \leftarrow j+1$ and swap $x_{j-1} \leftrightarrow x_j$. Stop.

J5. [Sift down.] While $j < n$, set $j \leftarrow j+1$, swap $x_{j-1} \leftrightarrow x_j$, and set $s \leftarrow \min(S, s)$.

J6. [End the pass.] If $p = 1$, go to J7. Otherwise, while $j \neq k$, swap $x_{j-1} \leftrightarrow x_j$ and set $j \leftarrow j-1$; then set $p \leftarrow 1$ and go to J2.

J7. [Finish upward.] While $s \neq S$, swap $x_{j-1} \leftrightarrow x_j$ and set $j \leftarrow j-1$. Stop. ▮

Whenever Algorithm J swaps $x_{j-1} \leftrightarrow x_j$, the variable that is currently called x_j is the original variable x_k. The total number of swaps varies from about n to about $2.5n$, depending on k and the optimum final position of x_k. But we can improve the running time substantially, without seriously affecting the outcome, if steps J2 and J5 are modified to proceed immediately to J3 and J6, respectively, whenever S becomes larger than, say, $1.2s$ or even $1.1s$ or even $1.05s$. In such cases, further sifting in the same direction is unlikely to decrease s.

Rudell's sifting procedure consists of applying Algorithm J exactly n times, once for each variable that is present; see exercise 151. We could continue sifting again and again until there is no more improvement; but the additional gain is usually not worth the extra effort.

Let's look at a detailed example, in order to make these ideas concrete. We've observed that when the contiguous United States are arranged in the order

$$\begin{matrix}\texttt{ME NH VT MA RI CT NY NJ PA DE MD DC VA NC SC GA FL AL TN KY WV OH MI IN}\\ \texttt{IL WI MN IA MO AR MS LA TX OK KS NE SD ND MT WY CO NM AZ UT ID WA OR NV CA}\end{matrix} \qquad (104)$$

as in (17), they lead to a BDD of size 428 for the independent-set function

$$\neg\big((x_{\texttt{AL}} \wedge x_{\texttt{FL}}) \vee (x_{\texttt{AL}} \wedge x_{\texttt{GA}}) \vee (x_{\texttt{AL}} \wedge x_{\texttt{MS}}) \vee \cdots \vee (x_{\texttt{UT}} \wedge x_{\texttt{WY}}) \vee (x_{\texttt{VA}} \wedge x_{\texttt{WV}})\big). \quad (105)$$

The author chose the ordering (104) by hand, starting with the historical/geographical listing of states that he had been taught as a child, then trying to minimize the size of the boundary between states-already-listed and states-to-come, so that the BDD for (105) would not need to "remember" too many partial results at any level. The resulting size, 428, is pretty good for a function of 49 variables; but sifting is able to make it even better. For example, consider WV: Some of the possibilities for altering its position, with varying sizes S, are

$$\underset{424}{|}\texttt{RI}\underset{422}{|}\texttt{CT}\underset{417}{|}\texttt{NY}\underset{415}{|}\texttt{NJ}\underset{414}{|}\texttt{PA}\underset{412}{|}\texttt{DE}\underset{411}{|}\texttt{MD}\underset{410}{|}\texttt{DC}\underset{412}{|}\texttt{VA}\underset{412}{|}\texttt{NC}\underset{415}{|}\texttt{SC}\underset{420}{|}\texttt{GA}\underset{421}{|}\texttt{FL}\underset{426}{|}\texttt{AL}\underset{425}{|}\texttt{TN}\underset{427}{|}\texttt{KY}\underset{428}{|}\texttt{OH}\underset{428}{|}\texttt{MI}\underset{436}{|}\texttt{IN}\underset{442}{|}\texttt{IL}\underset{453}{|}$$

so we can save $428 - 410 = 18$ nodes by jumping WV up to a position between MD and DC. By using Algorithm J to sift on all the variables — first on ME, then on NH, then ..., then on CA — we end up with the ordering

$$\begin{matrix}\texttt{VT MA ME NH CT RI NY NJ DE PA MD WV VA DC KY OH NC GA SC AL FL MS TN IN}\\ \texttt{IL MI AR TX LA OK MO IA WI MN CO NE KS MT ND WY SD UT AZ NM ID CA OR WA NV}\end{matrix} \qquad (106)$$

and the BDD size has been reduced to 345(!). That sifting process involves a total of 4663 swaps, requiring less than 4 megamems of computation altogether.

Instead of choosing an ordering carefully, let's consider a lazier alternative: We might begin with the states in alphabetic order

$$\begin{matrix}\texttt{AL AR AZ CA CO CT DC DE FL GA IA ID IL IN KS KY LA MA MD ME MI MN MO MS}\\ \texttt{MT NC ND NE NH NJ NM NV NY OH OK OR PA RI SC SD TN TX UT VA VT WA WI WV WY}\end{matrix} \qquad (107)$$

and proceed from there. Then the BDD for (105) turns out to have 306,214 nodes; it can be computed either via Algorithm S (with about 380 megamems of machine time) or via (55) and Algorithm U (with about 565 megamems). In this case sifting makes a dramatic difference: Those 306,214 nodes become only 2871, at a cost of 430 additional megamems. Furthermore, the sifting cost goes down from 430 Mμ to 210 Mμ if the loops of Algorithm J are aborted when $S > 1.1s$. (The more radical choice of aborting when $S > 1.05s$ would reduce the cost of sifting to 155 Mμ; but the BDD size would be reduced only to 2946 in that case.)

And we can actually do much, much better, if we sift the variables *while* evaluating (105), instead of waiting until that whole long sequence of disjunctions been entirely computed. For example, suppose we invoke sifting automatically whenever the BDD size surpasses twice the number of nodes that were present after the previous sift. Then the evaluation of (105), starting from the alphabetic ordering (107), runs like a breeze: It automatically churns out a BDD that has only 419 nodes, after only about 60 megamems of calculation! Neither human ingenuity nor "geometric understanding" are needed to discover the ordering

$$\begin{matrix}\texttt{NV OR ID WA AZ CA UT NM WY CO MT OK TX NE MO KS LA AR MS TN IA ND MN SD}\\ \texttt{GA FL AL NC SC KY WI MI IL OH IN WV MD VA DC PA NJ DE NY CT RI NH ME VT MA}\end{matrix} \qquad (108)$$

which beats the author's (104). For this one, the computer just decided to invoke autosifting 39 times, on smaller BDDs.

What is the *best* ordering of states for the function (105)? The answer to that question will probably never be known for sure, but we can make a pretty good guess. First of all, a few more sifts of (108) will yield a still-better ordering

$$\begin{matrix}\texttt{OR ID NV WA AZ CA UT NM WY CO MT SD MN ND IA NE OK KS TX MO LA AR MS TN}\\ \texttt{GA FL AL NC SC KY WI MI IL OH IN WV MD DC VA PA NJ DE NY CT RI NH ME VT MA}\end{matrix} \qquad (109)$$

with BDD size 354. Sifting will not improve (109) further; but sifting has only limited power, because it explores only $(n-1)^2$ alternative orderings, out of $n!$ possibilities. (Indeed, exercise 134 exhibits a function of only four variables whose BDD cannot be improved by sifting, even though the ordering of its variables is not optimum.) There is, however, another arrow in our quiver: We can use *master profile charts* to optimize every window of, say, 16 consecutive levels in the BDD. There are 34 such windows; and the algorithm of exercise 139 optimizes each of them rather quickly. After about 9.6 gigamems of computation, that algorithm discovers a new champion

$$\begin{matrix}\texttt{OR ID NV WA AZ CA UT NM WY CO MT SD MN ND IA NE OK KS TX MO LA AR MS WI}\\ \texttt{KY MI IN IL AL TN FL NC SC GA WV OH MD DC VA PA NJ DE NY CT RI NH ME VT MA}\end{matrix} \qquad (110)$$

by cleverly rearranging 16 states within (109). This ordering, for which the BDD size is only 339, might well be optimum, because it cannot be improved either by sifting or by optimizing any window of width 25. However, such a conjecture

megamems
Mμ
sifting, partial
sifting automatically
autosifting
master profile charts
window

rests on shaky ground: The ordering

```
 AL GA FL TN NC SC VA MS AR TX LA OK KY MO NM WV MD DC PA NJ DE OH IL MI
IN IA NE KS WI SD WY ND MN MT UT CO ID CA AZ OR WA NV NY CT RI NH ME VT MA
```
(111)

also happens to be unimprovable by sifting and by width-25 window optimization, yet its BDD has 606 nodes and is far from optimum.

With the improved ordering (110), the 98-variable COLOR function of (73) needs only 22037 BDD nodes, instead of 25579. Sifting reduces it to 16098.

optimal versus optimum
coloring
4-coloring
read-once funcs++
Fibonacci numbers
Sauerhoff
Wegener
Werchner

***Read-once functions.** Boolean functions such as $(x_1 \supset x_2) \oplus ((x_3 \equiv x_4) \wedge x_5)$, which can be expressed as formulas in which each variable occurs exactly once, form an important class for which optimum orderings of variables can easily be computed. Formally, let us say that $f(x_1, \ldots, x_n)$ is a *read-once function* if either (i) $n = 1$ and $f(x_1) = x_1$; or (ii) $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_k) \circ h(x_{k+1}, \ldots, x_n)$, where $\circ$ is one of the binary operators $\{\wedge, \vee, \bar\wedge, \bar\vee, \supset, \subset, \bar\supset, \bar\subset, \oplus, \equiv\}$ and where both g and h are read-once functions. In case (i) we obviously have $B(f) = 3$. And in case (ii), exercise 163 proves that

$$B(f) = \begin{cases} B(g) + B(h) - 2, & \text{if } \circ \in \{\wedge, \vee, \bar\wedge, \bar\vee, \supset, \subset, \bar\supset, \bar\subset\}; \\ B(g) + B(h, \bar h) - 2, & \text{if } \circ \in \{\oplus, \equiv\}. \end{cases} \tag{112}$$

In order to get a recurrence, we also need the similar formulas

$$B(f, \bar f) = \begin{cases} 4, & \text{if } n = 1; \\ 2B(g) + B(h, \bar h) - 4, & \text{if } \circ \in \{\wedge, \vee, \bar\wedge, \bar\vee, \supset, \subset, \bar\supset, \bar\subset\}; \\ B(g, \bar g) + B(h, \bar h) - 2, & \text{if } \circ \in \{\oplus, \equiv\}. \end{cases} \tag{113}$$

A particularly interesting family of read-once functions arises when we define

$$\begin{aligned} u_{m+1}(x_1, \ldots, x_{2^{m+1}}) &= v_m(x_1, \ldots, x_{2^m}) \wedge v_m(x_{2^m+1}, \ldots, x_{2^{m+1}}), \\ v_{m+1}(x_1, \ldots, x_{2^{m+1}}) &= u_m(x_1, \ldots, x_{2^m}) \oplus u_m(x_{2^m+1}, \ldots, x_{2^{m+1}}), \end{aligned} \tag{114}$$

and $u_0(x_1) = v_0(x_1) = x_1$; for example, $u_3(x_1, \ldots, x_8) = \bigl((x_1 \wedge x_2) \oplus (x_3 \wedge x_4)\bigr) \wedge \bigl((x_5 \wedge x_6) \oplus (x_7 \wedge x_8)\bigr)$. Exercise 165 shows that the BDD sizes for these functions, calculated via (112) and (113), involve Fibonacci numbers:

$$\begin{aligned} B(u_{2m}) &= 2^m F_{2m+2} + 2, & \qquad B(u_{2m+1}) &= 2^{m+1} F_{2m+2} + 2; \\ B(v_{2m}) &= 2^m F_{2m+2} + 2, & \qquad B(v_{2m+1}) &= 2^m F_{2m+4} + 2. \end{aligned} \tag{115}$$

Thus u_m and v_m are functions of $n = 2^m$ variables whose BDD sizes grow as

$$\Theta(2^{m/2}\phi^m) = \Theta(n^\beta), \qquad \text{where } \beta = 1/2 + \lg\phi \approx 1.19424. \tag{116}$$

In fact, the BDD sizes in (115) are optimum for the u and v functions, under all permutations of the variables, because of a fundamental result due to M. Sauerhoff, I. Wegener, and R. Werchner:

Theorem W. *If $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_k) \circ h(x_{k+1}, \ldots, x_n)$ is a read-once function, there is a permutation π that minimizes $B(f^\pi)$ and $B(f^\pi, \bar f^\pi)$ simultaneously, and in which the variables $\{x_1, \ldots, x_k\}$ occur either first or last.*

Proof. Any permutation $(1\pi, \ldots, n\pi)$ leads naturally to an "unshuffled" permutation $(1\sigma, \ldots, n\sigma)$ in which the first k elements are $\{1, \ldots, k\}$ and the last $n-k$ elements are $\{k+1, \ldots, n\}$, retaining the π order within each group. For example, if $k = 7$, $n = 9$, and $(1\pi, \ldots, 9\pi) = (3, 1, 4, 5, 9, 2, 6, 8, 7)$, we have $(1\sigma, \ldots, 9\sigma) = (3, 1, 4, 5, 2, 6, 7, 9, 8)$. Exercise 166 proves that, in appropriate circumstances, we have $B(f^\sigma) \le B(f^\pi)$ and $B(f^\sigma, \bar{f}^\sigma) \le B(f^\pi, \bar{f}^\pi)$. ▌

Using this theorem together with (112) and (113), we can readily optimize the ordering of variables for the BDD of any given read-once function. Consider, for example, $(x_1 \vee x_2) \oplus (x_3 \wedge x_4 \wedge x_5) = g(x_1, x_2) \oplus h(x_3, x_4, x_5)$. We have $B(g) = 4$ and $B(g, \bar{g}) = 6$; $B(h) = 5$ and $B(h, \bar{h}) = 8$. For the overall formula $f = g \oplus h$, Theorem W says that there are two candidates for a best ordering $(1\pi, \ldots, 5\pi)$, namely $(1, 2, 3, 4, 5)$ and $(4, 5, 1, 2, 3)$. The first of these gives $B(f^\pi) = B(g) + B(h, \bar{h}) - 2 = 10$; the other one excels, with $B(f^\pi) = B(h) + B(g, \bar{g}) - 2 = 9$.

The algorithm in exercise 167 finds an optimum π for any read-once function $f(x_1, \ldots, x_n)$ in $O(n)$ steps. Moreover, a careful analysis proves that $B(f^\pi) = O(n^\beta)$ in the best ordering, where β is the constant in (116). (See exercise 168.)

***Multiplication.** Some of the most interesting Boolean functions, from a mathematical standpoint, are the $m + n$ bits that arise when an m-bit number is multiplied by an n-bit number:

$$(x_m \ldots x_2 x_1)_2 \times (y_n \ldots y_2 y_1)_2 \;=\; (z_{m+n} \ldots z_2 z_1)_2. \tag{117}$$

In particular, the "leading bit" z_{m+n}, and the "middle bit" z_n when $m = n$, are especially noteworthy. To remove the dependence of this notation on m and n, we can imagine that $m = n = \infty$ by letting $x_i = y_j = 0$ for all $i > m$ and $j > n$; then each z_k is a function of $2k$ variables, $z_k = Z_k(x_1, \ldots, x_k; y_1, \ldots, y_k)$, namely the middle bit of the product $(x_k \ldots x_1)_2 \times (y_k \ldots y_1)_2$.

The middle bit turns out to be difficult, BDDwise, even when y is constant. Let $Z_{n,a}(x_1, \ldots, x_n) = Z_n(x_1, \ldots, x_n; a_1, \ldots, a_n)$, where $a = (a_n \ldots a_1)_2$.

Theorem X. *There is a constant a such that $B_{\min}(Z_{n,a}) > \frac{5}{288} \cdot 2^{\lfloor n/2 \rfloor} - 2$.*

Proof. [P. Woelfel, *J. Computer and System Sci.* **71** (2005), 520–534.] We may assume that $n = 2t$ is even, since $Z_{2t+1,2a} = Z_{2t,a}$. Let $x = (x_n \ldots x_1)_2$ and $m = ([n\pi \le t] \ldots [1\pi \le t])_2$. Then $x = p + q$, where $q = x \mathbin{\&} m$ represents the "known" bits of x after t branches have been taken in a BDD for $Z_{n,y}$ with the ordering π, and $p = x \mathbin{\&} \overline{m}$ represents the bits yet unknown. Let

$$P \;=\; \{x \mathbin{\&} \overline{m} \mid 0 \le x < 2^n\} \quad \text{and} \quad Q \;=\; \{x \mathbin{\&} m \mid 0 \le x < 2^n\}. \tag{118}$$

For any fixed a, the function $Z_{n,a}$ has 2^t subfunctions

$$f_q(p) \;=\; \bigl((pa + qa) \gg (n-1)\bigr) \mathbin{\&} 1, \qquad q \in Q. \tag{119}$$

We want to show that some n-bit number a will make many of these subfunctions differ; in other words we want to find a large subset $Q^* \subseteq Q$ such that

$$q \in Q^* \text{ and } q' \in Q^* \text{ and } q \ne q' \text{ implies } f_q(p) \ne f_{q'}(p) \text{ for some } p \in P. \tag{120}$$

Exercise 176 shows in detail how this can be done. ▌

pi, as source of "random" data
multiplication++
middle bit++
leading bit
$Z_n(x,y)$
$Z_{n,a}$
Woelfel

Amano
Maruoka

Table 1

BEST AND WORST ORDERINGS FOR THE MIDDLE BIT z_n OF MULTIPLICATION

$\begin{array}{r} x_{11}x_{10}x_9x_7x_8x_6x_{13}x_{15} \\ \times\ x_{16}x_{14}x_{12}x_5x_4x_3x_2x_1 \end{array}$ $B_{\min}(Z_8) = 756$	$\begin{array}{r} x_{10}x_{11}x_9x_8x_7x_{16}x_6x_{15} \\ \times\ x_5x_4x_3x_{12}x_{13}x_2x_1x_{14} \end{array}$ $B_{\max}(Z_8) = 6791$
$\begin{array}{r} x_{24}x_{20}x_{18}x_{16}x_9x_8x_{10}x_{11}x_7x_{12}x_{14}x_{21} \\ \times\ x_{22}x_{19}x_{17}x_{15}x_6x_5x_4x_3x_2x_1x_{13}x_{23} \end{array}$ $B_{\min}(Z_{12}) = 21931$	$\begin{array}{r} x_{16}x_{17}x_{15}x_{14}x_{24}x_{13}x_{12}x_{11}x_{20}x_{10}x_9x_{23} \\ \times\ x_8x_7x_6x_5x_{18}x_4x_{22}x_3x_2x_{19}x_1x_{21} \end{array}$ $B_{\max}(Z_{12}) = 866283$

Table 2

BEST AND WORST ORDERINGS FOR ALL BITS $\{z_1, \ldots, z_{m+n}\}$ OF MULTIPLICATION

$\begin{array}{r} x_{11}x_{16}x_{15}x_{14}x_{13}x_{12}x_{10}x_9 \\ \times\ x_8x_7x_6x_5x_4x_3x_2x_1 \end{array}$ $B_{\min}(Z_{8,8}^{(1)}, \ldots, Z_{8,8}^{(16)}) = 9700$	$\begin{array}{r} x_{10}x_8x_9x_{13}x_2x_1x_{11}x_7 \\ \times\ x_{16}x_5x_{15}x_6x_4x_{14}x_3x_{12} \end{array}$ $B_{\max}(Z_{8,8}^{(1)}, \ldots, Z_{8,8}^{(16)}) = 28678$
$\begin{array}{r} x_{15}x_{17}x_{24}x_{23}x_{22}x_{21}x_{20}x_{19}x_{18}x_{16}x_{14}x_{13} \\ \times\ x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12} \end{array}$ $B_{\min}(Z_{12,12}^{(1)}, \ldots, Z_{12,12}^{(24)}) = 648957$	$\begin{array}{r} x_{17}x_{22}x_{14}x_{13}x_{16}x_{10}x_{20}x_3x_2x_1x_{19}x_{12} \\ \times\ x_{24}x_{15}x_9x_8x_{21}x_7x_6x_{11}x_{23}x_5x_4x_{18} \end{array}$ $B_{\max}(Z_{12,12}^{(1)}, \ldots, Z_{12,12}^{(24)}) = 4224195$
$\begin{array}{r} x_{17}x_{16}x_{10}x_9x_{11}x_{12}\ldots x_{15}x_{18}x_{19}x_{24}x_{23}\ldots x_{20} \\ \times\ x_1x_2x_3x_4x_5x_6x_7x_8 \end{array}$ $B_{\min}(Z_{16,8}^{(1)}, \ldots, Z_{16,8}^{(24)}) = 157061$	$\begin{array}{r} x_{13}x_{14}x_{12}x_{15}x_{16}x_{17}x_{22}x_{10}x_8x_7x_{18}x_9x_2x_1x_{19}x_6 \\ \times\ x_{24}x_{11}x_{21}x_5x_4x_{23}x_3x_{20} \end{array}$ $B_{\max}(Z_{16,8}^{(1)}, \ldots, Z_{16,8}^{(24)}) = 1236251$

A good upper bound for the BDD size of the middle bit function when neither operand is constant has been found by K. Amano and A. Maruoka, *Discrete Applied Math.* **155** (2007), 1224–1232:

Theorem A. *Let $f(x_1, \ldots, x_{2n}) = Z_n(x_1, x_3, \ldots, x_{2n-1}; x_2, x_4, \ldots, x_{2n})$. Then*

$$B(f) \;\le\; Q(f) \;<\; \tfrac{19}{7}\, 2^{\lceil 6n/5 \rceil}. \qquad (121)$$

Proof. Consider two n-bit numbers $x = 2^k x_h + x_l$ and $y = 2^k y_h + y_l$, with $n-k$ unknown bits in each of their high parts (x_h, y_h), while their k-bit low parts (x_l, y_l) are both known. Then the middle bit of xy is determined by adding together three $(n-k)$-bit quantities when $k \ge n/2$, namely $x_h y_l \bmod 2^{n-k}$, $x_l y_h \bmod 2^{n-k}$, and $(x_l y_l \gg k) \bmod 2^{n-k}$. Hence level $2k$ of the QDD needs to "remember" only the least significant $n-k$ bits of each of the prior quantities x_l, y_l, and $x_l y_l \gg k$, a total of $3n - 3k$ bits, and we have $q_{2k} \le 2^{3n-3k}$ in f's quasi-profile. Exercise 177 completes the proof. ▮

Amano and Maruoka also discovered another important upper bound. Let $Z_{m,n}^{(p)}(x_1, \ldots, x_m; y_1, \ldots, y_n)$ denote the pth bit z_p of the product (117).

Theorem Y. *For all constants $(a_m \ldots a_1)_2$ and for all p, the BDD and QDD for the function $Z_{m,n}^{(p)}(a_1, \ldots, a_m; x_1, \ldots, x_n)$ have fewer than $3 \cdot 2^{n/2}$ nodes.*

Proof. Exercise 180 proves that $q_k \le 2^{n+1-k}$ for this function. The theorem follows when we combine that result with the obvious upper bound $q_k \le 2^k$. ▮

Theorem Y shows that the lower bound of Theorem X is best possible, except for a constant factor. It also shows that the BDD base for all $m+n$ product functions $Z^{(p)}_{m,n}(x_1,\ldots,x_m;x_{m+1},\ldots,x_{m+n})$ is not nearly as large as $\Theta(2^{m+n})$, which we get for almost all instances of $m+n$ functions of $m+n$ variables:

Corollary Y. *If* $m \le n$, $B(Z^{(1)}_{m,n},\ldots,Z^{(m+n)}_{m,n}) < 3(m+n)2^{m+(n+1)/2}$. ▮

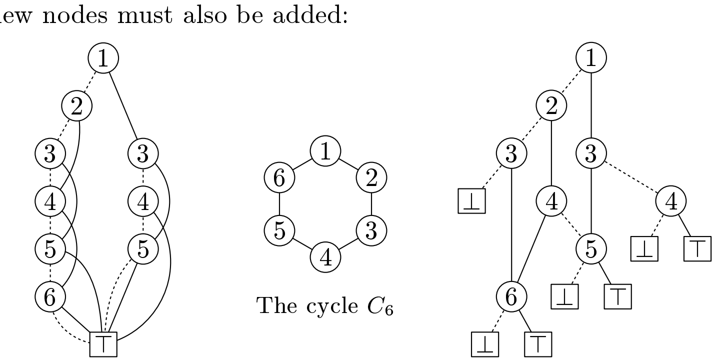
The best orderings of variables for the middle-bit function Z_n and for the complete BDD base remain mysterious, but empirical results for small m and n give reason to conjecture that the upper bounds of Theorem A and Corollary Y are not far from the truth; see Tables 1 and 2. Here, for example, are the optimum results of Z_n when $n \le 12$:

$n =$	1	2	3	4	5	6	7	8	9	10	11	12
$B_{\min}(Z_n) =$	4	8	14	31	63	136	315	756	1717	4026	9654	21931
$2^{6n/5} \approx$	2	5	12	28	64	147	338	776	1783	4096	9410	21619

The ratios $B_{\max}/B_{\min}$ with respect to the full BDD base $\{Z^{(1)}_{m,n},\ldots,Z^{(m+n)}_{m,n}\}$ are surprisingly small in Table 2. Therefore all orderings for that problem might turn out to be roughly equivalent.

Zero-suppressed BDDs: A combinatorial alternative. When BDDs are applied to combinatorial problems, a glance at the data in memory often reveals that most of the HI fields simply point to $\boxed{\bot}$. In such cases, we're better off using a variant data structure called a *zero-suppressed binary decision diagram*, or "ZDD" for short, introduced by Shin-ichi Minato [*ACM/IEEE Design Automation Conf.* **30** (1993), 272–277]. A ZDD has nodes like a BDD, but its nodes are interpreted differently: When an (i) node branches to a (j) node for $j > i{+}1$, it means that the Boolean function is false unless $x_{i+1} = \cdots = x_{j-1} = 0$.

For example, the BDDs for independent sets and kernels in (12) have many nodes with HI $= \boxed{\bot}$. Those nodes go away in the corresponding ZDDs, although a few new nodes must also be added:



Independent sets The cycle C_6 Kernels (122)

Notice that we might have LO = HI in a ZDD, because of the new conventions. Furthermore, the example on the left shows that a ZDD need not contain $\boxed{\bot}$ at all! About 40% of the nodes in (12) have been eliminated from each diagram.

zero-suppressed BDDs–
ZDDs–
Minato
independent sets
kernels

One good way to understand a ZDD is to regard it as a condensed representation of a *family of sets*. Indeed, the ZDDs in (122) represent respectively the families of all independent sets and all kernels of C_6. The root node of a ZDD names the smallest element that appears in at least one of the sets; its HI and LO branches represent the residual subfamilies that do and don't contain that element; and so on. At the bottom, $\boxed{\bot}$ represents the empty family '$\emptyset$', and $\boxed{\top}$ represents '$\{\emptyset\}$'. For example, the rightmost ZDD in (122) represents the family $\{\{1,3,5\}, \{1,4\}, \{2,4,6\}, \{2,5\}, \{3,6\}\}$, because the HI branch of the root represents $\{\{3,5\}, \{4\}\}$ and the LO branch represents $\{\{2,4,6\}, \{2,5\}, \{3,6\}\}$.

Every Boolean function $f(x_1, \ldots, x_n)$ is, of course, equivalent to a family of subsets of $\{1, \ldots, n\}$, and vice versa. But the family concept gives us a different perspective from the function concept. For example, the family $\{\{1,3\}, \{2\}, \{2,5\}\}$ has the same ZDD for all $n \geq 5$; but if, say, $n = 7$, the BDD for the function $f(x_1, \ldots, x_7)$ that defines this family needs additional nodes to ensure that $x_4 = x_6 = x_7 = 0$ when $f(x) = 1$.

Almost every notion that we've discussed for BDDs has a counterpart in the theory of ZDDs, although the actual data structures are often strikingly different. We can, for example, take the truth table for any given function $f(x_1, \ldots, x_n)$ and construct its unique ZDD in a straightforward way, analogous to the construction of its BDD as illustrated in (5). We know that the BDD nodes for f correspond to the "beads" of f's truth table; the ZDD nodes, similarly, correspond to *zeads*, which are binary strings of the form $\alpha\beta$ with $|\alpha| = |\beta|$ and $\beta \neq 0 \ldots 0$, or with $|\alpha| = |\beta| - 1$. Any binary string corresponds to a unique zead, obtained by lopping off the right half repeatedly, if necessary, until the string either has odd length or its right half is nonzero.

Dear reader, please take a moment now to work exercise 187. (Really.)

The *z-profile* of $f(x_1, \ldots, x_n)$ is $(z_0, \ldots, z_n)$, where z_k is the number of zeads of order $n-k$ in f's truth table, for $0 \leq k < n$, namely the number of $(k{+}1)$ nodes in the ZDD; also z_n is the number of sinks. We write $Z(f) = z_0 + \cdots + z_n$ for the total number of nodes. For example, the functions in (122) have z-profiles $(1,1,2,2,2,1,1)$ and $(1,1,2,2,1,1,2)$, respectively, so $Z(f) = 10$ in each case.

The basic relations (83)–(85) between profiles and quasi-profiles hold true also for z-profiles:

$$q_k \;\geq\; z_k, \qquad \text{for } 0 \leq k \leq n; \tag{123}$$

$$q_k \leq 1 + z_0 + \cdots + z_{k-1} \;\text{ and }\; q_k \leq z_k + \cdots + z_n, \quad \text{for } 0 \leq k \leq n; \tag{124}$$

$$Z(f) \;\geq\; 2q_k - 1, \qquad \text{for } 0 \leq k \leq n. \tag{125}$$

Consequently the BDD size and the ZDD size can never be wildly different:

$$Z(f) \;\leq\; \frac{n+1}{2}\bigl(B(f)+1\bigr) \qquad \text{and} \qquad B(f) \;\leq\; \frac{n+1}{2}\bigl(Z(f)+1\bigr). \tag{126}$$

On the other hand, a factor of 50 when $n = 100$ is nothing to sneeze at.

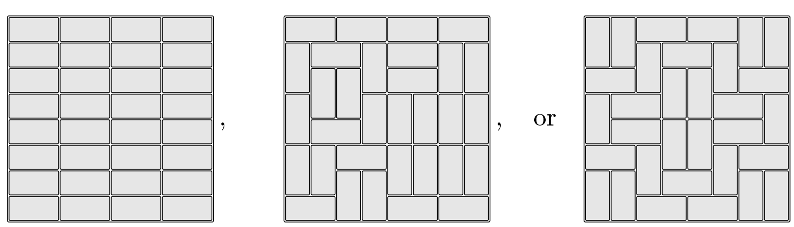
When ZDDs are used to find independent sets and kernels of the contiguous USA, using the original order of (17), the BDD sizes of 428 and 780 go down to 177 and 385, respectively. Sifting reduces these ZDD sizes to 160 and 335. Is anybody sneezing? That's amazingly good, for complicated functions of 49 variables.

family of sets
set of combinations, see family of sets
empty case
truth table
beads
zeads
z-profile
quasi-profiles
independent sets
kernels
contiguous USA
Sifting

When we know the ZDDs for f and g, we can synthesize them to obtain the ZDDs for $f \wedge g$, $f \vee g$, $f \oplus g$, etc., using algorithms that are very much like the methods we've used for BDDs. Furthermore we can count and/or optimize the solutions of f, with analogs of Algorithms C and B; in fact, ZDD-based techniques for counting and optimization turn out to be a bit easier than the corresponding BDD-based algorithms are. With slight modifications of BDD methods, we can also do dynamic variable reordering via sifting. Exercises 197–209 discuss the nuts and bolts of all the basic ZDD procedures.

In general, a ZDD tends to be better than a BDD when we're dealing with functions whose solutions are *sparse*, in the sense that νx tends to be small when $f(x) = 1$. And if $f(x)$ itself happens to be sparse, in the sense that it has comparatively few solutions, so much the better.

For example, ZDDs are well suited to *exact cover problems*, defined by an $m \times n$ matrix of 0s and 1s: We want to find all ways to choose rows that sum to $(1, 1, \ldots, 1)$. Our goal might be, say, to cover a chessboard with 32 dominoes, like



(127)

This is an exact cover problem whose matrix has $8 \times 8 = 64$ columns, one for each cell; there are $2 \times 7 \times 8 = 112$ rows, one for each pair of adjacent cells:

$$\begin{pmatrix} 1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \ldots 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0 \ldots 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0 \ldots 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0 \ldots 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ \vdots \qquad\qquad\qquad \vdots \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \ldots 0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \ldots 0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \ldots 0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1 \end{pmatrix}. \tag{128}$$

Let variable x_j represent the choice (or not) of row j. Thus the three solutions in (127) have $(x_1, x_2, x_3, x_4, \ldots, x_{110}, x_{111}, x_{112}) = (1, 0, 0, 0, \ldots, 1, 0, 1)$, $(1, 0, 0, 0, \ldots, 1, 0, 1)$, and $(0, 1, 0, 1, \ldots, 1, 0, 0)$, respectively. In general, the solutions to an exact cover problem are represented by the function

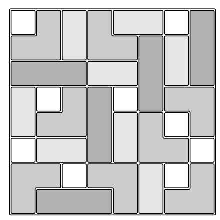
$$f(x_1, \ldots, x_m) = \bigwedge_{j=1}^{n} S_1(X_j) = \bigwedge_{j=1}^{n} [\nu X_j = 1], \tag{129}$$

where $X_j = \{x_i \mid a_{ij} = 1\}$ and (a_{ij}) is the given matrix.

The dominoes-on-a-chessboard ZDD turns out to have only $Z(f) = 2300$ nodes, even though f has $m = 112$ variables in this case. We can use it to prove that there are exactly 12,988,816 coverings such as (127).

synthesize
counting
optimization
dynamic variable reordering
sifting
solutions
sparse
tiling, see exact cover problems
exact cover problems+
matrix of 0s and 1s
chessboard

Similarly, we can investigate more exotic kinds of covering. In

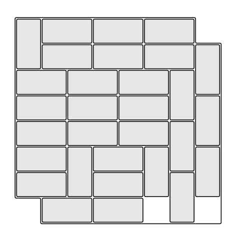


, (130)

for instance, a chessboard has been covered with monominoes, dominoes, and/or trominoes — that is, with rookwise-connected pieces that each have either one, two, or three cells. There are exactly 92,109,458,286,284,989,468,604 ways to do this(!); and we can compute that number almost instantly, doing only about 75 megamems of calculation, by forming a ZDD of size 512,227 on 468 variables.

A special algorithm could be devised to find the ZDD for any given exact cover problem; or we can synthesize the result using (129). See exercise 212.

Incidentally, the problem of domino covering as in (127) is equivalent to finding the perfect matchings of the grid graph $P_8 \,\square\, P_8$, which is bipartite. We will see in Section 7.5.1 that efficient algorithms are available by which perfect matchings can be studied on graphs that are far too large to be treated with BDD/ZDD techniques. In fact, there's even an explicit formula for the number of domino coverings of an $m \times n$ grid. By contrast, general coverings such as (130) fall into a wider category of hypergraph problems for which polynomial-time methods are unlikely to exist as $m, n \to \infty$.



An amusing variant of domino covering called the "mutilated chessboard" was considered by Max Black in his book *Critical Thinking* (1946), pages 142 and 394: Suppose we remove opposite corners of the chessboard, and try to cover the remaining cells with 31 dominoes. It's easy to place 30 of them, for example as shown here; but then we're stuck. Indeed, if we consider the corresponding 108×62 exact cover problem, but leave out the last two constraints of (129), we obtain a ZDD with 1224 nodes from which we can deduce that there are 324,480 ways to choose rows that sum to $(1, 1, \ldots, 1, 1, *, *)$. But each of those solutions has at least two 1s in column 61; therefore the ZDD reduces to $\boxed{\bot}$ after we AND in the constraint $[\nu X_{61} = 1]$. ("Critical thinking" explains why; see exercise 213.) This example reminds us that (i) the size of the final ZDD or BDD in a calculation can be much smaller than the time needed to compute it; and (ii) using our brains can save oodles of computer cycles.

ZDDs as dictionaries. Let's switch gears now, to note that ZDDs are advantageous also in applications that have an entirely different flavor. We can use them, for instance, to represent the *five-letter words of English*, the set `WORDS`(5757) from the Stanford GraphBase that is discussed near the beginning of this chapter. One way to do this is to consider the function $f(x_1, \ldots, x_{25})$ that is defined to be 1 if and only if the five numbers $(x_1 \ldots x_5)_2$, $(x_6 \ldots x_{10})_2$, …, $(x_{21} \ldots x_{25})_2$ encode the letters of an English word, where $\mathtt{a} = (00001)_2$, …, $\mathtt{z} = (11010)_2$.

monominoes
dominoes
polyominoes
trominoes
rookwise-connected
perfect matchings
grid graph
bipartite
matchings
hypergraph
mutilated chessboard
Black
five-letter words+
Stanford GraphBase

For example, $f(0,0,1,1,1,0,1,1,1,1,0,1,1,1,1,0,0,1,1,0,1,1,0,0,x_{25}) = x_{25}$. This function of 25 variables has $Z(f) = 6233$ nodes — which isn't bad, since it represents 5757 words.

Of course we've studied many other ways to represent 5757 words, in Chapter 6. The ZDD approach is no match for binary trees or tries or hash tables, when we merely want to do simple searches. But with ZDDs we can also retrieve data that is only partially specified, or data that is only supposed to match a key approximately; many complex queries can be handled with ease.

Furthermore, we don't need to worry very much about having lots of variables when ZDDs are being used. Instead of working with the 25 variables x_j considered above, we can also represent those five-letter words as a sparse function $F(a_1,\ldots,z_1,a_2,\ldots,z_2,\ldots,a_5,\ldots,z_5)$ that has $26\times 5 = 130$ variables, where variable a_2 (for example) controls whether the second letter is '`a`'. To indicate that `crazy` is a word, we make F true when $c_1 = r_2 = a_3 = z_4 = y_5 = 1$ and all other variables are 0. Equivalently, we consider F to be a family consisting of the 5757 subsets $\{\mathtt{w}_1,\mathtt{h}_2,\mathtt{i}_3,\mathtt{c}_4,\mathtt{h}_5\}$, $\{\mathtt{t}_1,\mathtt{h}_2,\mathtt{e}_3,\mathtt{r}_4,\mathtt{e}_5\}$, etc. With these 130 variables the ZDD size $Z(F)$ turns out to be only 5020 instead of 6233.

Incidentally, $B(F)$ is 46,189 — more than nine times as large as $Z(F)$. But $B(f)/Z(f)$ is only $8870/6233 \approx 1.4$ in the 25-variable case. The ZDD world is different from the BDD world in many ways, in spite of having similar algorithms and a similar theory.

One consequence of this difference is a need for new primitive operations by which complex families of subsets can readily be constructed from elementary families. Notice that the simple subset $\{\mathtt{f}_1,\mathtt{u}_2,\mathtt{n}_3,\mathtt{n}_4,\mathtt{y}_5\}$ is actually an extremely long-winded Boolean function:

$$\bar{a}_1 \wedge \cdots \wedge \bar{e}_1 \wedge f_1 \wedge \bar{g}_1 \wedge \cdots \wedge \bar{t}_2 \wedge u_2 \wedge \bar{v}_2 \wedge \cdots \wedge \bar{x}_5 \wedge y_5 \wedge \bar{z}_5, \qquad (131).$$

a minterm of 130 Boolean variables. Exercise 203 discusses an important *family algebra*, by which that subset is expressed more naturally as '$\mathtt{f}_1 \sqcup \mathtt{u}_2 \sqcup \mathtt{n}_3 \sqcup \mathtt{n}_4 \sqcup \mathtt{y}_5$'. With family algebra we can readily describe and compute many interesting collections of words and word fragments (see exercise 222).

ZDDs to represent simple paths. An important connection between arbitrary directed, acyclic graphs (dags) and a special class of ZDDs is illustrated in Fig. 28. When every source vertex of the dag has out-degree 1 and every sink vertex has in-degree 1, the ZDD for all oriented paths from a source to a sink has essentially the same "shape" as the original dag. The variables in this ZDD are the *arcs* of the dag, in a suitable topological order. (See exercise 224.)

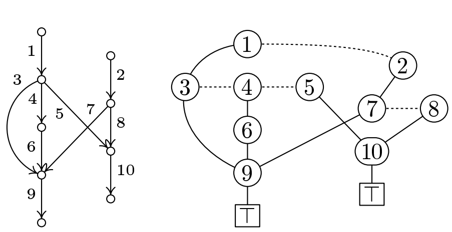
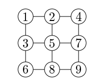
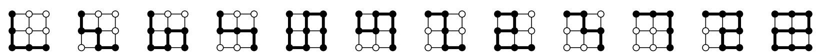


Fig. 28. A dag, and the ZDD for its source-to-sink paths. Arcs of the dag correspond to vertices of the ZDD. All branches to $\boxed{\bot}$ have been omitted from the ZDD in order to show the structural similarities more clearly.

tries
hash tables
sparse function
minterm
family algebra
simple paths++
acyclic graphs
dags
source vertex
sink vertex
oriented paths
topological order

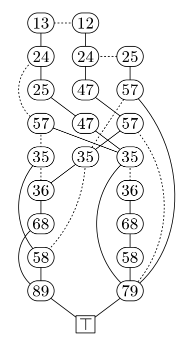
We can also use ZDDs to represent simple paths in an *undirected* graph. For example, there are 12 ways to go from the upper left corner of a 3×3 grid to the lower right corner, without visiting any point twice:





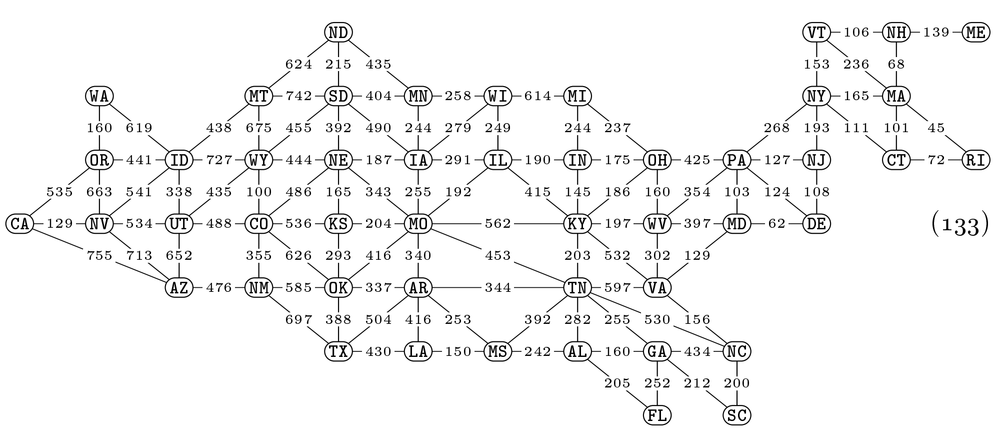
(132)

These paths can be represented by the ZDD shown at the right, which characterizes all sets of suitable edges. For example, we get the first path by taking the HI branches at 13, 36, 68, and 89 of the ZDD. (As in Fig. 28, this diagram has been simplified by omitting all of the uninteresting LO branches that merely go to $\boxed{\bot}$.) Of course this ZDD isn't a truly great way to represent (132), because that family of paths has only 12 members. But on the larger grid $P_8 \,\square\, P_8$, the number of simple paths from corner to corner turns out to be 789,360,053,252; and they can all be represented by a ZDD that has at most 33580 nodes. Exercise 225 explains how to construct such a ZDD quickly.



A similar algorithm, discussed in exercise 226, constructs a ZDD that represents all *cycles* of a given graph. With a ZDD of size 22275, we can deduce that $P_8 \,\square\, P_8$ has exactly 603,841,648,931 simple cycles. This ZDD may well provide the best way to represent all of those cycles within a computer, and the best way to generate them systematically if desired.

The same ideas work well with graphs from the "real world" that don't have a neat mathematical structure. For example, we can use them to answer a question posed to the author in 2008 by Randal Bryant: "Suppose I wanted to take a driving tour of the Continental U.S., visiting all of the state capitols, and passing through each state only once. What route should I take to minimize the total distance?" The following diagram shows the shortest distances between neighboring capital cities, when restricted to local itineraries that each cross only one state boundary:



(133)

The problem is to choose a subset of these edges that form a Hamiltonian path of smallest total length.

grid
self-avoiding walks
cycles
Cycles of a graph, generation of all
Knuth
Bryant
state capitols+
contiguous USA+
Capitol, Montana
Hamiltonian path
traveling salesrep problem

generating function
Hamiltonian
New England
articulation point
Bryant
prime implicants of monotone+
monotone Boolean function
two-level representation
disjunctive prime form
prime implicants
minimal solutions
family algebra
absorption+

Every Hamiltonian path in this graph must clearly either start or end at Augusta, Maine (`ME`). Suppose we start in Sacramento, California (`CA`). Proceeding as above, we can find a ZDD that characterizes all paths from `CA` to `ME`; this ZDD turns out to have only 7850 nodes, and it quickly tells us that exactly 437,525,772,584 simple paths from `CA` to `ME` are possible. In fact, the generating function by number of edges turns out to be

$$4z^{11} + 124z^{12} + 1539z^{13} + \cdots + 33385461z^{46} + 2707075z^{47}; \qquad (134)$$

so the longest such paths are Hamiltonian, and there are exactly 2,707,075 of them. Furthermore, exercise 227 shows how to construct a smaller ZDD, of size 4726, which describes just the Hamiltonian paths from `CA` to `ME`.

We could repeat this experiment for each of the states in place of California. (Well, the starting point had better be outside of New England, if we are going to get past New York, which is an articulation point of this graph.) For example, there are 483,194 Hamiltonian paths from `NJ` to `ME`. But exercise 228 shows how to construct a *single* ZDD of size 28808 for the family of all Hamiltonian paths from `ME` to *any* other final state — of which there are 68,656,026. The answer to Bryant's problem now pops out immediately, via Algorithm B. (The reader may like to try finding a minimum route by hand, before turning to exercise 230 and discovering the absolutely optimum answer.)

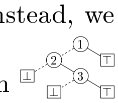
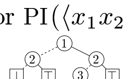
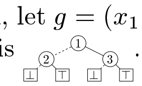
***ZDDs and prime implicants.** Finally, let's look at an instructive application in which BDDs and ZDDs are both used simultaneously.

According to Theorem 7.1.1Q, every monotone Boolean function f has a unique shortest two-level representation as an OR of ANDs, called its "disjunctive prime form" — the disjunction of all of its prime implicants. The prime implicants correspond to the minimal points where $f(x) = 1$, namely the binary vectors x for which we have $f(x') = 1$ and $x' \subseteq x$ if and only if $x' = x$. If

$$f(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge x_3), \qquad (135)$$

for example, the prime implicants of f are x_1 and $x_2 \wedge x_3$, while the minimal solutions are $x_1x_2x_3 = 100$ and 011. These minimal solutions can also be expressed conveniently as e_1 and $e_2 \sqcup e_3$, using family algebra (see exercise 203).

In general, $x_{i_1} \wedge \cdots \wedge x_{i_s}$ is a prime implicant of a monotone function f if and only if $e_{i_1} \sqcup \cdots \sqcup e_{i_s}$ is a minimal solution of f. Thus we can consider f's prime implicants $\mathrm{PI}(f)$ to be its family of minimal solutions. Notice, however, that $x_{i_1} \wedge \cdots \wedge x_{i_s} \subseteq x_{j_1} \wedge \cdots \wedge x_{j_t}$ if and only if $e_{i_1} \sqcup \cdots \sqcup e_{i_s} \supseteq e_{j_1} \sqcup \cdots \sqcup e_{j_t}$; so it's confusing to say that one prime implicant "contains" another. Instead, we say that the shorter one "absorbs" the longer one.

A curious phenomenon shows up in example (135): The diagram  is not only the BDD for f, it's also the ZDD for $\mathrm{PI}(f)$! Similarly, Fig. 21 at the beginning of this section illustrates not only the BDD for $\langle x_1x_2x_3\rangle$ but also the ZDD for $\mathrm{PI}(\langle x_1x_2x_3\rangle)$. On the other hand, let $g = (x_1 \wedge x_3) \vee x_2$. Then the BDD for g is  but the ZDD for $\mathrm{PI}(g)$ is . What's going on here?

The key to resolving this mystery lies in the recursive structure on which BDDs and ZDDs are based. Every Boolean function can be represented as

$$f(x_1,\ldots,x_n) \;=\; (\bar{x}_1?\; f_0\colon f_1) \;=\; (\bar{x}_1 \wedge f_0) \vee (x_1 \wedge f_1), \tag{136}$$

where f_c is the value of f when x_1 is replaced by c. When f is monotone we also have $f = f_0 \vee (x_1 \wedge f_1)$, because $f_0 \subseteq f_1$. If $f_0 \neq f_1$, the BDD for f is obtained by creating a node 1 whose LO and HI branches point to the BDDs for f_0 and f_1. Similarly, it's not difficult to see that the prime implicants of f are

$$\mathrm{PI}(f) \;=\; \mathrm{PI}(f_0) \cup \bigl(e_1 \sqcup (\mathrm{PI}(f_1) \setminus \mathrm{PI}(f_0))\bigr). \tag{137}$$

(See exercise 253.) This is the recursion that defines the ZDD for $\mathrm{PI}(f)$, when we add the termination conditions for constant functions: The ZDDs for $\mathrm{PI}(0)$ and $\mathrm{PI}(1)$ are $\boxed{\bot}$ and $\boxed{\top}$.

Let's say that a Boolean function is *sweet* if it is monotone and if the ZDD for $\mathrm{PI}(f)$ is exactly the same as the BDD for f. Constant functions are clearly sweet. And nonconstant sweetness is easily characterized:

Theorem S. *A Boolean function that depends on x_1 is sweet if and only if its prime implicants are $P \cup (x_1 \sqcup Q)$, where P and Q are sweet and independent of x_1, and every member of P is absorbed by some member of Q.*

Proof. See exercise 246. (To say that "P and Q are sweet" means that they each are families of prime implicants that define a sweet Boolean function.) ∎

Corollary S. *The connectedness function of any graph is sweet.*

Proof. The prime implicants of the connectedness function f are the spanning trees of the graph. Every spanning tree that does not include arc x_1 has at least one subtree that will be spanning when arc x_1 is added to it. Furthermore, all subfunctions of f are the connectedness functions of smaller graphs. ∎

Thus, for example, the BDD in Fig. 22, which defines all 431 of the connected subgraphs of $P_3 \,\square\, P_3$, also is the ZDD that defines all 192 of its spanning trees.

Whether f is sweet or not, we can use (137) to compute the ZDD for $\mathrm{PI}(f)$ whenever f is monotone. When we do this we can actually let the BDD nodes and the ZDD nodes *coexist* in the same big base of data: Two nodes with identical (V, LO, HI) fields might as well appear only once in memory, even though they might have complete different meanings in different contexts. We use one routine to synthesize $f \wedge \bar{g}$ when f and g point to BDDs, and another routine to form $f \setminus g$ when f and g point to ZDDs; no trouble will arise if these routines happen to share nodes, as long as the variables aren't being reordered. (Of course the cache memos must distinguish BDD facts from ZDD facts when we do this.)

For example, exercise 7.1.1–67 defines an interesting class of self-dual functions called the Y functions, and the BDD for Y_{12} (which is a function of 91 variables) has 748,416 nodes. This function has 2,178,889,774 prime implicants; yet $Z(\mathrm{PI}(Y_{12}))$ is only 217,388. (We can find this ZDD with a computational cost of about 13 gigamems and 660 megabytes.)

recursive
sweet
monotone
spanning trees
ZDDs mixed with BDDs
cache memos
Y functions
–zero-suppressed ZDDs

A brief history. The seeds of binary decision diagrams were implicitly planted by Claude Shannon [*Trans. Amer. Inst. Electrical Engineers* **57** (1938), 713–723], in his illustrations of relay-contact networks. Section 4 of that paper showed that any symmetric Boolean function of n variables has a BDD with at most $\binom{n+1}{2}$ branch nodes. Shannon preferred to work with Boolean algebra; but C. Y. Lee, in *Bell System Tech. J.* **38** (1959), 985–999, pointed out several advantages of what he called "binary-decision programs," because any n-variable function could be evaluated by executing at most n branch instructions in such a program.

S. Akers coined the name "binary decision diagrams" and pursued the ideas further in *IEEE Trans.* **C-27** (1978), 509–516. He showed how to obtain a BDD from a truth table by working bottom-up, or from algebraic subfunctions by working top-down. He explained how to count the paths from a root to $\boxed{\top}$ or $\boxed{\bot}$, and observed that these paths partition the n-cube into disjoint subcubes.

Meanwhile a very similar model of Boolean computation arose in theoretical studies of automata. For example, A. Cobham [*FOCS* **7** (1966), 78–87] related the minimum sizes of branching programs for a sequence of functions $f_n(x_1,\ldots,x_n)$ to the space complexity of nonuniform Turing machines that compute this sequence. More significantly, S. Fortune, J. Hopcroft, and E. M. Schmidt [*Lecture Notes in Comp. Sci.* **62** (1978), 227–240] considered "free B-schemes," now known as FBDDs, in which no Boolean variable is tested twice on any path (see exercise 35). Among other results, they gave a polynomial-time algorithm to test whether $f = g$, given FBDDs for f and g, provided that at least one of those FBDDs is ordered consistently as in a BDD. The theory of finite-state automata, which has intimate connections to BDD structure, was also being developed; thus several researchers worked on problems that are equivalent to analyzing the size, $B(f)$, for various functions f. (See exercise 261.)

All of this work was conceptual, not implemented in computer programs, although programmers had found good uses for binary tries and Patrician trees— which are similar to BDDs except that they are trees instead of dags (see Section 6.3). But then Randal E. Bryant discovered that binary decision diagrams are significantly important in practice when they are required to be both *reduced* and *ordered*. His introduction to the subject [*IEEE Trans.* **C-35** (1986), 677–691] became for many years the most cited paper in all of computer science, because it revolutionized the data structures used to represent Boolean functions.

In his paper, Bryant pointed out that the BDD for any function is essentially unique under his conventions, and that most of the functions encountered in practice had BDDs of reasonable size. He presented efficient algorithms to synthesize the BDDs for $f \wedge g$ and $f \oplus g$, etc., from the BDDs for f and g. He also showed how to compute the lexicographically least x such that $f(x) = 1$, etc.

Lee, Akers, and Bryant all noted that many functions can profitably coexist in a BDD base, sharing their common subfunctions. A high-performance "package" for BDD base operations, developed by K. S. Brace, R. L. Rudell, and R. E. Bryant [*ACM/IEEE Design Automation Conf.* **27** (1990), 40–45], has strongly influenced all subsequent implementations of BDD toolkits. Bryant summarized the early uses of BDDs in *Computing Surveys* **24** (1992), 293–318.

Shannon
relay-contact networks
symmetric Boolean function
Lee
Akers
truth table
subfunctions
count
n-cube
subcubes
automata
Cobham
branching programs
space complexity
nonuniform Turing machines
Turing machines
Fortune
Hopcroft
Schmidt
free
B-schemes
FBDDs
finite-state automata
tries
Patrician trees
dags
overlapping subtrees
Bryant
reduced
ordered
synthesize
lexicographically least
Lee
Akers
Bryant
BDD base
package
Brace
Rudell
Bryant
toolkits

Shin-ichi Minato introduced ZDDs in 1993, as noted above, to improve performance in combinatorial work. He gave a retrospective account of early ZDD applications in *Software Tools for Technology Transfer* **3** (2001), 156–170.

The use of Boolean methods in graph theory was pioneered by K. Maghout [*Comptes Rendus Acad. Sci.* **248** (Paris, 1959), 3522–3523], who showed how to express the maximal independent sets and the minimal dominating sets of any graph or digraph as the prime implicants of a monotone function. Then R. Fortet [*Cahiers du Centre d'Etudes Recherche Operationelle* **1**, 4 (1959), 5–44] considered Boolean approaches to a variety of other problems; for example, he introduced the idea of 4-coloring a graph by assigning two Boolean variables to each vertex, as we have done in (73). P. Camion, in that same journal [**2** (1960), 234–289], transformed integer programming problems into equivalent problems in Boolean algebra, hoping to resolve them via techniques of symbolic logic. This work was extended by others, notably P. L. Hammer and S. Rudeanu, whose book *Boolean Methods in Operations Research* (Springer, 1968) summarized the ideas. Unfortunately, however, their approach foundered, because no good techniques for Boolean calculation were available at the time. The proponents of Boolean methods had to wait until the advent of BDDs before the general Boolean programming problem (7) could be resolved, thanks to Algorithm B. The special case of Algorithm B in which all weights satisfying $w_i \geq 0$ was introduced by B. Lin and F. Somenzi [*IEEE/ACM International Conf. Computer-Aided Design* **CAD-90** (1990), 88–91]. S. Minato [*Formal Methods in System Design* **10** (1999), 221–242] developed software that automatically converts linear inequalities between integer variables into BDDs that can be manipulated conveniently, somewhat as the researcher of the 1960s had hoped would be possible.

The classic problem of finding a minimum size DNF for a given function also became spectacularly simpler when BDD methods became understood. The latest techniques for that problem are beyond the scope of this book, but Olivier Coudert has given an excellent overview in *Integration* **17** (1994), 97–140.

A fine book by Ingo Wegener, *Branching Programs and Binary Decision Diagrams* (SIAM, 2000), surveys the vast literature of the subject, develops the mathematical foundations carefully, and discusses many ways in which the basic ideas have been generalized and extended.

Caveat. We've seen dozens of examples in which the use of BDDs and/or ZDDs has made it possible to solve a wide variety of combinatorial problems with amazing efficiency, and the exercises below contain dozens of additional examples where such methods shine. But BDD and ZDD structures are by no means a panacea; they're only two of the weapons in our arsenal. They apply chiefly to problems that have more solutions than can readily be examined one by one, problems whose solutions have a local structure that allows our algorithms to deal with only relatively few subproblems at a time. In later sections of *The Art of Computer Programming* we shall be studying additional techniques by which other kinds of combinatorial problems can be tamed.

Minato
ZDDs
graph theory
Maghout
maximal independent sets
minimal dominating sets
kernels
prime implicants
monotone function
Fortet
4-coloring
Camion
integer programming problems
Hammer
Rudeanu
Boolean programming problem
Lin
Somenzi
Minato
linear inequalities
integer variables
DNF
Coudert
Wegener

EXERCISES

▶ **1.** [*20*] Draw the BDDs for all 16 Boolean functions $f(x_1, x_2)$. What are their sizes?

▶ **2.** [*21*] Draw a planar dag with sixteen vertices, each of which is the root of one of the 16 BDDs in exercise 1.

3. [*16*] How many Boolean functions $f(x_1, \ldots, x_n)$ have BDD size 3 or less?

4. [*21*] Suppose three fields $\boxed{\mathtt{V}\,|\quad\mathtt{LO}\quad|\quad\mathtt{HI}\quad}$ have been packed into a 64-bit word x, where `V` occupies 8 bits and the other two fields occupy 28 bits each. Show that five bitwise instructions will transform $x \mapsto x'$, where x' is equal to x except that a `LO` or `HI` value of 0 is changed to 1 and vice versa. (Repeating this operation on every branch node x of a BDD for f will produce the BDD for the complementary function, $\bar{f}$.)

5. [*20*] If you take the BDD for $f(x_1, \ldots, x_n)$ and interchange the LO and HI pointers of every node, and if you also swap the two sinks $\boxed{\bot} \leftrightarrow \boxed{\top}$, what do you get?

6. [*10*] Let $g(x_1, x_2, x_3, x_4) = f(x_4, x_3, x_2, x_1)$, where f has the BDD in (6). What is the truth table of g, and what are its beads?

7. [*21*] Given a Boolean function $f(x_1, \ldots, x_n)$, let

$$g_k(x_0, x_1, \ldots, x_n) = f(x_0, \ldots, x_{k-2}, x_{k-1} \vee x_k, x_{k+1}, \ldots, x_n) \qquad \text{for } 1 \le k \le n.$$

Find a simple relation between (a) the truth tables and (b) the BDDs of f and g_k.

8. [*22*] Solve exercise 7 with $x_{k-1} \oplus x_k$ in place of $x_{k-1} \vee x_k$.

9. [*16*] Given the BDD for a function $f(x) = f(x_1, \ldots, x_n)$, represented sequentially as in (8), explain how to determine the lexicographically largest x such that $f(x) = 0$.

▶ **10.** [*21*] Given two BDDs that define Boolean functions f and f', represented sequentially as in (8) and (10), design an algorithm that tests $f = f'$.

11. [*20*] Does Algorithm C give the correct answer if it is applied to a binary decision diagram that is (a) ordered but not reduced? (b) reduced but not ordered?

▶ **12.** [*M21*] A *kernel* of a digraph is a set of vertices K such that

$$\begin{aligned} v \in K \quad &\text{implies} \quad v \not\longrightarrow u \text{ for all } u \in K; \\ v \notin K \quad &\text{implies} \quad v \longrightarrow u \text{ for some } u \in K. \end{aligned}$$

a) Show that when the digraph is an ordinary graph (that is, when $u \longrightarrow v$ if and only if $v \longrightarrow u$), a kernel is the same as a maximal independent set.
b) Describe the kernels of the *oriented* cycle $C_n^{\rightarrow}$.
c) Prove that an acyclic digraph has a *unique* kernel.

13. [*M15*] How is the concept of a graph kernel related to the concept of (a) a maximal clique? (b) a minimal vertex cover?

14. [*M24*] How big, exactly, are the BDDs for (a) all independent sets of the cycle graph C_n, and (b) all kernels of C_n, when $n \ge 3$? (Number the vertices as in (12).)

15. [*M23*] How many (a) independent sets and (b) kernels does C_n have, when $n \ge 3$?

▶ **16.** [*22*] Design an algorithm that successively generates all vectors $x_1 \ldots x_n$ for which $f(x_1, \ldots, x_n) = 1$, when a BDD for f is given.

17. [*32*] If possible, improve the algorithm of exercise 16 so that its running time is $O(B(f)) + O(N)$ when there are N solutions.

18. [*13*] Play through Algorithm B with the BDD (8) and $(w_1, \ldots, w_4) = (1,-2,-3,4)$.

Binary Boolean operations
Two-variable functions
BDD base
bitwise instructions
broadword chains
truth table
beads
substituting an expression for a variable
lexicographically largest
sequential representation of BDDs
isomorphism of BDDs
equality testing of Boolean functions
counting solutions
satisfiability counting
ordered
reduced
kernel
maximal independent set
$C_n^{\rightarrow}$
oriented cycle
acyclic digraph
dag
maximal clique
minimal vertex cover
clique
vertex cover
cycle graph C_n
consecutive 1s forbidden
two-in-a-row function
cycle graph C_n
generating all solutions

19. [*20*] What are the largest and smallest possible values of variable m_k in Algorithm B, based only on the weights $(w_1, \ldots, w_n)$, not on any details of the function f?

20. [*15*] Devise a fast way to compute the Thue–Morse weights (15) for $1 \le j \le n$.

21. [*05*] Can Algorithm B *minimize* $w_1x_1 + \cdots + w_nx_n$, instead of maximizing it?

▶ **22.** [*M21*] Suppose step B3 has been simplified so that '$W_{v+1} - W_{v_l}$' and '$W_{v+1} - W_{v_h}$' are eliminated from the formulas. Prove that the algorithm will still work, when applied to BDDs that represent kernels of graphs.

▶ **23.** [*M20*] All paths from the root of the BDD in Fig. 22 to $\boxed{\top}$ have exactly eight solid arcs. Why is this not a coincidence?

24. [*M22*] Suppose twelve weights $(w_{12}, w_{13}, \ldots, w_{89})$ have been assigned to the edges of the grid in Fig. 22. Explain how to find a minimum spanning tree in that graph (namely, a spanning tree whose edges have minimum total weight), by applying Algorithm B to the BDD shown there.

25. [*M20*] Modify Algorithm C so that it computes the generating function for the solutions to $f(x_1, \ldots, x_n) = 1$, namely $G(z) = \sum_{x_1=0}^{1} \cdots \sum_{x_n=0}^{1} z^{x_1+\cdots+x_n} f(x_1, \ldots, x_n)$.

26. [*M20*] Modify Algorithm C so that it computes the reliability polynomial for given probabilities, namely

$$F(p_1, \ldots, p_n) = \sum_{x_1=0}^{1} \cdots \sum_{x_n=0}^{1} (1-p_1)^{1-x_1} p_1^{x_1} \ldots (1-p_n)^{1-x_n} p_n^{x_n} f(x_1, \ldots, x_n).$$

▶ **27.** [*M26*] Suppose $F(p_1, \ldots, p_n)$ and $G(p_1, \ldots, p_n)$ are the reliability polynomials for Boolean functions $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$, where $f \ne g$. Let q be a prime number, and choose independent random integers $q_1, \ldots, q_n$, uniformly distributed in the range $0 \le q_k < q$. Prove that $F(q_1, \ldots, q_n) \bmod q \ne G(q_1, \ldots, q_n) \bmod q$ with probability $\ge (1 - 1/q)^n$. (In particular, if $n = 1000$ and $q = 2^{31} - 1$, different functions lead to different "hash values" under this scheme with probability at least 0.9999995.)

28. [*M16*] Let $F(p)$ be the value of the reliability polynomial $F(p_1, \ldots, p_n)$ when $p_1 = \cdots = p_n = p$. Show that it's easy to compute $F(p)$ from the generating function $G(z)$.

29. [*HM20*] Modify Algorithm C so that it computes the reliability polynomial $F(p)$ of exercise 28 and also its derivative $F'(p)$, given p and the BDD for f.

▶ **30.** [*M21*] The reliability polynomial is the sum, over all solutions to $f(x_1, \ldots, x_n) = 1$, of contributions from all "minterms" $(1-p_1)^{1-x_1} p_1^{x_1} \ldots (1-p_n)^{1-x_n} p_n^{x_n}$. Explain how to find a solution $x_1 \ldots x_n$ whose contribution to the total reliability is maximum, given a BDD for f and a sequence of probabilities $(p_1, \ldots, p_n)$.

31. [*M21*] Modify Algorithm C so that it computes the fully elaborated truth table of f, formalizing the procedure by which (24) was obtained from Fig. 21.

▶ **32.** [*M20*] What interpretations of '$\circ$', '$\bullet$', '$\bot$', '$\top$', '$\bar{x}_j$', and 'x_j' will make the general algorithm of exercise 31 specialize to the algorithms of exercises 25, 26, 29, and 30?

▶ **33.** [*M22*] Specialize exercise 31 so that we can efficiently compute

$$\sum_{f(x)=1} (w_1x_1 + \cdots + w_nx_n) \qquad \text{and} \qquad \sum_{f(x)=1} (w_1x_1 + \cdots + w_nx_n)^2$$

from the BDD of a Boolean function $f(x) = f(x_1, \ldots, x_n)$.

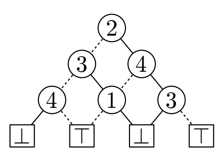
Thue–Morse
minimize
kernels
grid
minimum spanning tree
spanning tree
generating function
reliability polynomial
multilinear int rep
hash values
derivative
minterms
fully elaborated truth table

34. [*M25*] Specialize exercise 31 so that we can efficiently compute

$$\max\{\max_{1\le k\le n}(w_1x_1+\cdots+w_{k-1}x_{k-1}+w'_kx_k+w_{k+1}x_{k+1}+\cdots+w_nx_n+w''_k)\mid f(x)=1\}$$

from the BDD of f, given $3n$ arbitrary weights $(w_1,\ldots,w_n,w'_1,\ldots,w'_n,w''_1,\ldots,w''_n)$.

▸ **35.** [*22*] A *free binary decision diagram* (FBDD) is a binary decision diagram such as



where the branch variables needn't appear in any particular order, but no variable is allowed to occur more than once on any downward path from the root. (An FBDD is "free" in the sense that every path in the dag is possible: No branch constrains another.)

a) Design an algorithm to verify that a supposed FBDD is really free.

b) Show that it's easy to compute the reliability polynomial $F(p_1,\ldots,p_n)$ of a Boolean function $f(x_1,\ldots,x_n)$, given $(p_1,\ldots,p_n)$ and an FBDD that defines f, and to compute the number of solutions to $f(x_1,\ldots,x_n)=1$.

36. [*25*] By extending exercise 31, explain how to compute the elaborated truth table for any given FBDD, if the abstract operators $\circ$ and $\bullet$ are commutative as well as distributive and associative. (Thus we can find optimum solutions as in Algorithm B, or solve problems such as those in exercises 30 and 33, with FBDDs as well as with BDDs.)

37. [*M20*] (R. L. Rivest and J. Vuillemin.) A Boolean function $f(x_1,\ldots,x_n)$ is called *evasive* if every FBDD for f contains a downward path of length n. Let $G(z)$ be the generating function for f, as in exercise 25. Prove that f is evasive if $G(-1)\neq 0$.

▸ **38.** [*27*] Let I_{s-1}, …, I_0 be branch instructions that define a nonconstant Boolean function $f(x_1,\ldots,x_n)$ as in (8) and (10). Design an algorithm that computes the status variables $t_1\ldots t_n$, where

$$t_j \;=\; \begin{cases} +1, & \text{if } f(x_1,\ldots,x_n)=1 \text{ whenever } x_j=1;\\ -1, & \text{if } f(x_1,\ldots,x_n)=1 \text{ whenever } x_j=0;\\ 0, & \text{otherwise.}\end{cases}$$

(If $t_1\ldots t_n\neq 0\ldots 0$, the function f is therefore *canalizing* as defined in Section 7.1.1.) The running time of your algorithm should be $O(n+s)$.

39. [*M20*] What is the size of the BDD for the threshold function $[x_1+\cdots+x_n\ge k]$?

▸ **40.** [*22*] Let g be the "condensation" of f obtained by setting $x_{k+1}\leftarrow x_k$ as in (27).

a) Prove that $B(g)\le B(f)$. [*Hint:* Consider subtables and beads.]

b) Suppose h is obtained from f by setting $x_{k+2}\leftarrow x_k$. Is $B(h)\le B(f)$?

41. [*M25*] Assuming that $n\ge 4$, find the BDD size of the Fibonacci threshold functions (a) $\langle x_1^{F_1}x_2^{F_2}\ldots x_{n-2}^{F_{n-2}}x_{n-1}^{F_{n-1}}x_n^{F_{n-2}}\rangle$ and (b) $\langle x_n^{F_1}x_{n-1}^{F_2}\ldots x_3^{F_{n-2}}x_2^{F_{n-1}}x_1^{F_{n-2}}\rangle$.

42. [*22*] Draw the BDD base for all symmetric Boolean functions of 3 variables.

▸ **43.** [*22*] What is $B(f)$ when (a) $f(x_1,\ldots,x_{2n})=[x_1+\cdots+x_n=x_{n+1}+\cdots+x_{2n}]$? (b) $f(x_1,\ldots,x_{2n})=[x_1+x_3+\cdots+x_{2n-1}=x_2+x_4+\cdots+x_{2n}]$?

▸ **44.** [*M32*] Determine the maximum possible size, S_n, of $B(f)$ when f is a symmetric Boolean function of n variables.

free binary decision diagram
FBDD
read-once branching program, see FBDD
reliability polynomial
elaborated truth table
commutative
distributive
associative
Boolean programming
Rivest
Vuillemin
evasive
elusive function, see evasive
exhaustive function, see evasive
generating function
canalizing
threshold function
condensation
substituting one variable for another
subtables
beads
Fibonacci threshold functions
BDD base
symmetric Boolean functions

45. [*22*] Give precise specifications for the Boolean modules that compute the three-in-a-row function as in (33) and (34), and show that the network is well defined.

46. [*M23*] What is the true BDD size of the three-in-a-row function?

47. [*M21*] Devise and prove a *converse* of Theorem M: Every Boolean function f with a small BDD can be implemented by an efficient network of modules.

48. [*M22*] Implement the hidden weighted bit function with a network of modules like Fig. 23, using $a_k = 2 + \lambda k$ and $b_k = 1 + \lambda(n-k)$ connecting wires for $1 \le k < n$. Conclude from Theorem B that the upper bound in Theorem M cannot be improved to $\sum_{k=0}^{n} 2^{p(a_k,b_k)}$ for any polynomial p.

49. [*20*] Draw the BDD base for the following sets of symmetric Boolean functions: (a) $\{S_{\ge k}(x_1,x_2,x_3,x_4) \mid 1 \le k \le 4\}$; (b) $\{S_k(x_1,x_2,x_3,x_4) \mid 0 \le k \le 4\}$.

50. [*22*] Draw the BDD base for the functions of the 7-segment display (7.1.2–(42)).

51. [*22*] Describe the BDD base for binary addition when the input bits are numbered from right to left, namely $(f_{n+1}f_nf_{n-1}\ldots f_1)_2 = (x_{2n-1}\ldots x_3x_1)_2 + (x_{2n}\ldots x_4x_2)_2$, instead of from left to right as in (35) and (36).

52. [*20*] There's a sense in which the BDD base for m functions $\{f_1,\ldots,f_m\}$ isn't really very different from a BDD with just one root: Consider the *junction function* $J(u_1,\ldots,u_n;v_1,\ldots,v_n) = (u_1?\ v_1\colon u_2?\ v_2\colon \cdots u_n?\ v_n\colon 0)$, and let

$$f(t_1,\ldots,t_{m+1},x_1,\ldots,x_n) = J(t_1,\ldots,t_{m+1}; f_1(x_1,\ldots,x_n),\ldots,f_m(x_1,\ldots,x_n),1),$$

where $(t_1,\ldots,t_{m+1})$ are new "dummy" variables, placed ahead of $(x_1,\ldots,x_n)$ in the ordering. Show that $B(f)$ is almost the same as the size of the BDD base for $\{f_1,\ldots,f_m\}$.

▶ **53.** [*23*] Play through Algorithm R, when it is applied to the binary decision diagram with seven branch nodes in (2).

54. [*17*] Construct the BDD of $f(x_1,\ldots,x_n)$ from f's truth table, in $O(2^n)$ steps.

55. [*M30*] Explain how to construct the "connectedness BDD" of a graph (like Fig. 22).

56. [*20*] Modify Algorithm R so that, instead of pushing any unnecessary nodes onto an `AVAIL` stack, it creates a brand new BDD, consisting of consecutive instructions I_{s-1}, …, I_1, I_0 that have the compact form $(\bar v_k?\ l_k\colon h_k)$ assumed in Algorithms B and C. (The original nodes input to the algorithm can then all be recycled en masse.)

57. [*25*] Specify additional actions to be taken between steps R1 and R2 when Algorithm R is extended to compute the restriction of a function. Assume that `FIX[`v`]` $= t \in \{0,1\}$ if variable v is to be given the fixed value t; otherwise `FIX[`v`]` < 0.

58. [*20*] Prove that the "melded" diagram defined by recursive use of (37) is reduced.

▶ **59.** [*M28*] Let $h(x_1,\ldots,x_n)$ be a Boolean function. Describe the melded BDD $f \diamond g$ in terms of the BDD for h, when (a) $f(x_1,\ldots,x_{2n}) = h(x_1,\ldots,x_n)$ and $g(x_1,\ldots,x_{2n}) = h(x_{n+1},\ldots,x_{2n})$; (b) $f(x_1,x_2,\ldots,x_{2n}) = h(x_1,x_3,\ldots,x_{2n-1})$ and $g(x_1,x_2,\ldots,x_{2n}) = h(x_2,x_4,\ldots,x_{2n})$. [In both cases we obviously have $B(f) = B(g) = B(h)$.]

60. [*M22*] Suppose $f(x_1,\ldots,x_n)$ and $g(x_1,\ldots,x_n)$ have the profiles $(b_0,\ldots,b_n)$ and $(b'_0,\ldots,b'_n)$, respectively, and let their respective quasi-profiles be $(q_0,\ldots,q_n)$ and $(q'_0,\ldots,q'_n)$. Show that their meld $f \diamond g$ has $B(f \diamond g) \le \sum_{j=0}^{n}(q_jb'_j + b_jq'_j - b_jb'_j)$ nodes.

▶ **61.** [*M27*] If α and β are nodes of the respective BDDs for f and g, prove that

$$\text{in-degree}(\alpha \diamond \beta) \;\le\; \text{in-degree}(\alpha)\cdot\text{in-degree}(\beta)$$

in the melded BDD $f \diamond g$. (Imagine that the root of a BDD has in-degree 1.)

three-in-a-row
network model of computation
modules in a network
hidden weighted bit function
Notation: lambda n
BDD base
symmetric Boolean functions
threshold functions
seven-segment display
binary addition
BDD base
junction function
conditional expression, nested, see junction function
notation $J(u_1,\ldots,u_n;v_1,\ldots,v_n)$
truth table
connectedness
`AVAIL` stack
sequential representation of BDDs
restriction
melded
reduced
profiles
quasi-profiles
meld
in-degree

▶ **62.** [*M21*] If $f(x) = \bigvee_{j=1}^{\lfloor n/2 \rfloor}(x_{2j-1} \wedge x_{2j})$ and $g(x) = (x_1 \wedge x_n) \vee \bigvee_{j=1}^{\lceil n/2 \rceil - 1}(x_{2j} \wedge x_{2j+1})$, what are the asymptotic values of $B(f)$, $B(g)$, $B(f \diamond g)$, and $B(f \vee g)$ as $n \to \infty$?

63. [*M27*] Let $f(x_1, \ldots, x_n) = M_m(x_1 \oplus x_2, x_3 \oplus x_4, \ldots, x_{2m-1} \oplus x_{2m}; x_{2m+1}, \ldots, x_n)$ and $g(x_1, \ldots, x_n) = M_m(x_2 \oplus x_3, \ldots, x_{2m-2} \oplus x_{2m-1}, x_{2m}; \bar{x}_{2m+1}, \ldots, \bar{x}_n)$, where $n = 2m + 2^m$. What are $B(f)$, $B(g)$, and $B(f \wedge g)$?

64. [*M21*] We can compute the median $\langle f_1 f_2 f_3 \rangle$ of three Boolean functions by forming

$$f_4 = f_1 \vee f_2, \quad f_5 = f_1 \wedge f_2, \quad f_6 = f_3 \wedge f_4, \quad f_7 = f_5 \vee f_6.$$

Then $B(f_4) = O(B(f_1)B(f_2))$, $B(f_5) = O(B(f_1)B(f_2))$, $B(f_6) = O(B(f_3)B(f_4)) = O(B(f_1)B(f_2)B(f_3))$; therefore $B(f_7) = O(B(f_5)B(f_6)) = O(B(f_1)^2 B(f_2)^2 B(f_3))$. Prove, however, that $B(f_7)$ is actually only $O(B(f_1)B(f_2)B(f_3))$, and the running time to compute it from f_5 and f_6 is also $O(B(f_1)B(f_2)B(f_3))$.

▶ **65.** [*M25*] If $h(x_1, \ldots, x_n) = f(x_1, \ldots, x_{j-1}, g(x_1, \ldots, x_n), x_{j+1}, \ldots, x_n)$, prove that $B(h) = O(B(f)^2 B(g))$. Can this upper bound be improved to $O(B(f)B(g))$ in general?

66. [*20*] Complete Algorithm S by explaining what to do in step S1 if $f \circ g$ turns out to be trivially constant.

67. [*24*] Sketch the actions of Algorithm S when (41) defines f and g, and $op = 1$.

68. [*20*] Speed up step S10 by streamlining the common case when `LEFT(t)` < 0.

69. [*21*] Algorithm S ought to have one or more precautionary instructions such as "if `NTOP` $>$ `TBOT`, terminate the algorithm unsuccessfully," in case it runs out of room. Where are the best places to insert them?

70. [*21*] Discuss setting b to $\lfloor \lg$ `LCOUNT[l]` $\rfloor$ instead of $\lceil \lg$ `LCOUNT[l]` $\rceil$ in step S4.

71. [*20*] Discuss how to extend Algorithm S to ternary operators.

72. [*25*] Explain how to eliminate hashing from Algorithm S.

▶ **73.** [*25*] Discuss the use of "virtual addresses" instead of actual addresses as the links of a BDD: Each pointer p has the form $\pi(p)2^e + \sigma(p)$, where $\pi(p) = p \gg e$ is p's "page" and $\sigma(p) = p \bmod 2^e$ is p's "slot"; the parameter e can be chosen for convenience. Show that, with this approach, only two fields (`LO`, `HI`) are needed in BDD nodes, because the variable identifier $V(p)$ can be deduced from the virtual address p itself.

▶ **74.** [*M23*] Explain how to count the number of *self-dual* monotone Boolean functions of n variables, by modifying (49).

75. [*M20*] Let $\rho_n(x_1, \ldots, x_{2^n})$ be the Boolean function that is true if and only if $x_1 \ldots x_{2^n}$ is the truth table of a *regular* function (see exercise 7.1.1–110). Show that the BDD for ρ_n can be computed by a procedure similar to that of μ_n in (49).

▶ **76.** [*M22*] A "clutter" is a family $\mathcal{S}$ of mutually incomparable sets; in other words, $S \not\subseteq S'$ whenever S and S' are distinct members of $\mathcal{S}$. Every set $S \subseteq \{0, 1, \ldots, n-1\}$ can be represented as an n-bit integer $s = \sum\{2^e \mid e \in S\}$; so every family of such sets corresponds to a binary vector $x_0 x_1 \ldots x_{2^n - 1}$, with $x_s = 1$ if and only if s represents a set of the family.

Show that the BDD for the function '$[x_0 x_1 \ldots x_{2^n-1}$ corresponds to a clutter$]$' has a simple relation to the BDD for the monotone-function function $\mu_n(x_1, \ldots, x_{2^n})$.

▶ **77.** [*M30*] Show that there's an infinite sequence $(b_0, b_1, b_2, \ldots) = (1, 2, 3, 5, 6, \ldots)$ such that the profile of the BDD for μ_n is $(b_0, b_1, \ldots, b_{2^{n-1}-1}, b_{2^{n-1}-1}, \ldots, b_1, b_0, 2)$. (See Fig. 25.) How many branch nodes of that BDD have LO $= \boxed{\bot}$?

consecutive 1s forbidden
two-in-a-row function
2^m-way multiplexer
median
replacement of variables by functions
substitution of functions for variables
composition of Boolean functions
ternary operators
hashing
virtual addresses
page
slot
self-dual
truth table
regular
monotone Boolean functions
clutter
antichain of subsets, see clutter
mutually incomparable sets
family of sets
monotone-function function
profile

▶ **78.** [*25*] Use BDDs to determine the number of graphs on 12 labeled vertices for which the maximum vertex degree is d, for $0 \le d \le 11$.

79. [*20*] For $0 \le d \le 11$, compute the probability that a graph on vertices $\{1,\ldots,12\}$ has maximum degree d, if each edge is present with probability 1/3.

80. [*23*] The recursive algorithm (55) computes $f \wedge g$ in a depth-first manner, while Algorithm S does its computation breadth-first. Do both algorithms encounter the same subproblems $f' \wedge g'$ as they proceed (but in a different order), or does one algorithm consider fewer cases than the other?

▶ **81.** [*20*] By modifying (55), explain how to compute $f \oplus g$ in a BDD base.

▶ **82.** [*25*] When the nodes of a BDD base have been endowed with REF fields, explain how those fields should be adjusted within (55) and within Algorithm U.

83. [*M20*] Prove that if f and g both have reference count 1, we needn't consult the memo cache when computing $\mathrm{AND}(f,g)$ by (55).

84. [*24*] Suggest strategies for choosing the size of the memo cache and the sizes of the unique tables, when implementing algorithms for BDD bases. What is a good way to schedule periodic garbage collections?

85. [*16*] Compare the size of a BDD base for the 32 functions of 16×16-bit binary multiplication with the alternative of just storing a complete table of all possible products.

▶ **86.** [*21*] The routine MUX in (62) refers to "obvious" values. What are they?

87. [*20*] If the median operator $\langle fgh\rangle$ is implemented with a recursive subroutine analogous to (62), what are its "obvious" values?

▶ **88.** [*M25*] Find functions f, g, and h for which the recursive ternary computation of $f \wedge g \wedge h$ outperforms any of the binary computations $(f \wedge g) \wedge h$, $(g \wedge h) \wedge f$, $(h \wedge f) \wedge g$.

89. [*15*] Are the following quantified formulas true or false? (a) $\exists x_1 \exists x_2 f = \exists x_2 \exists x_1 f$. (b) $\forall x_1 \forall x_2 f = \forall x_2 \forall x_1 f$. (c) $\forall x_1 \exists x_2 f \le \exists x_2 \forall x_1 f$. (d) $\forall x_1 \exists x_2 f \ge \exists x_2 \forall x_1 f$.

90. [*M20*] When $l = m = n = 3$, Eq. (64) corresponds to the `MOR` operation of `MMIX`. Is there an analogous formula that corresponds to `MXOR` (matrix multiplication mod 2)?

▶ **91.** [*26*] In practice we often want to simplify a Boolean function f with respect to a "care set" g, by finding a function $\hat f$ with small $B(\hat f)$ such that

$$f(x) \wedge g(x) \;\le\; \hat f(x) \;\le\; f(x) \vee \bar g(x) \qquad \text{for all } x.$$

In other words, $\hat f(x)$ must agree with $f(x)$ whenever x satisfies $g(x) = 1$, but we don't care what value $\hat f(x)$ assumes when $g(x) = 0$. An appealing candidate for such an $\hat f$ is provided by the function $f \downarrow g$, "f constrained by g," defined as follows: If $g(x)$ is identically 0, $f \downarrow g = 0$. Otherwise $(f \downarrow g)(x) = f(y)$, where y is the first element of the sequence x, $x \oplus 1$, $x \oplus 2$, …, such that $g(y) = 1$. (Here we think of x and y as n-bit numbers $(x_1 \ldots x_n)_2$ and $(y_1 \ldots y_n)_2$. Thus $x \oplus 1 = x \oplus 0\ldots01 = x_1 \ldots x_{n-1}\bar x_n$; $x \oplus 2 = x \oplus 0\ldots010 = x_1 \ldots x_{n-2}\bar x_{n-1}x_n$; etc.)

a) What are $f \downarrow 1$, $f \downarrow x_j$, and $f \downarrow \bar x_j$?
b) Prove that $(f \wedge f') \downarrow g = (f \downarrow g) \wedge (f' \downarrow g)$.
c) True or false: $\bar f \downarrow g = \overline{f \downarrow g}$.
d) Simplify the formula $f(x_1,\ldots,x_n) \downarrow (x_2 \wedge \bar x_3 \wedge \bar x_5 \wedge x_6)$.
e) Simplify the formula $f(x_1,\ldots,x_n) \downarrow (x_1 \oplus x_2 \oplus \cdots \oplus x_n)$.
f) Simplify the formula $f(x_1,\ldots,x_n) \downarrow ((x_1 \wedge \cdots \wedge x_n) \vee (\bar x_1 \wedge \cdots \wedge \bar x_n))$.
g) Simplify the formula $f(x_1,\ldots,x_n) \downarrow (x_1 \wedge g(x_2,\ldots,x_n))$.

graphs
vertex degree
recursive algorithm
depth-first
breadth-first
reference counters
memo cache
unique tables
garbage collections
binary multiplication
median operator
ternary
quantified formulas
MOR
MXOR
matrix multiplication mod 2
care set
don't care

h) Find functions $f(x_1, x_2)$ and $g(x_1, x_2)$ such that $B(f \downarrow g) > B(f)$.
i) Devise a recursive way to compute $f \downarrow g$, analogous to (55).

92. [*M27*] The operation $f{\downarrow}g$ in exercise 91 sometimes depends on the ordering of the variables. Given $g = g(x_1, \ldots, x_n)$, prove that $(f^\pi \downarrow g^\pi) = (f \downarrow g)^\pi$ for all permutations π of $\{1, \ldots, n\}$ and for all functions $f = f(x_1, \ldots, x_n)$ if and only if $g = 0$ or g is a subcube (a conjunction of literals).

93. [*36*] Given a graph G on the vertices $\{1, \ldots, n\}$, construct Boolean functions f and g with the property that an approximating function $\hat{f}$ exists as in exercise 91 with small $B(\hat{f})$ if and only if G can be 3-colored. (Hence the task of minimizing $B(\hat{f})$ is NP-complete.)

94. [*21*] Explain why (65) performs existential quantification correctly.

▸ **95.** [*20*] Improve on (65) by testing if $r_l = 1$ before computing r_h.

96. [*20*] Show how to achieve (a) universal quantification $\forall x_{j_1} \ldots \forall x_{j_m} f = f \mathbin{\mathrm{A}} g$, and (b) differential quantification $Ↄx_{j_1} \ldots Ↄx_{j_m} f = f \mathbin{\mathrm{D}} g$, by modifying (65).

97. [*M20*] Prove that it's possible to compute arbitrary bottom-of-the-BDD quantifications such as $\exists x_{n-5} \forall x_{n-4} Ↄx_{n-3} \exists x_{n-2} \curlywedge x_{n-1} \forall x_n f(x_1, \ldots, x_n)$ in $O(B(f))$ steps.

▸ **98.** [*22*] In addition to (70), explain how to define the vertices $\mathrm{ENDPT}(x)$ of $\mathcal{G}$ that have degree ≤ 1. Also characterize $\mathrm{PAIR}(x, y)$, the components of size 2.

99. [*20*] (R. E. Bryant, 1984.) Every 4-coloring of the US map considered in the text corresponds to 24 solutions of the COLOR function (73), under permutation of colors. What's a good way to remove this redundancy?

▸ **100.** [*24*] In how many ways is it possible to 4-color the contiguous USA with exactly 12 states of each color? (Eliminate `DC` from the graph.)

101. [*20*] Continuing exercise 100, with colors $\{1, 2, 3, 4\}$, find such a coloring that maximizes $\sum$ (state weight) $\times$ (state color), where states are weighted as in (18).

102. [*23*] Design a method to cache the results of functional composition using the following conventions: The system maintains at all times an array of functions $[g_1, \ldots, g_n]$, one for each variable x_j. Initially g_j is simply the projection function x_j, for $1 \leq j \leq n$. This array can be changed only by the subroutine $\mathrm{NEWG}(j, g)$, which replaces g_j by g. The subroutine $\mathrm{COMPOSE}(f)$ always performs functional composition with respect to the current array of replacement functions.

▸ **103.** [*20*] Mr. B. C. Dull wanted to evaluate the formula

$$\exists y_1 \ldots \exists y_m \big((y_1 = f_1(x_1, \ldots, x_n)) \wedge \cdots \wedge (y_m = f_m(x_1, \ldots, x_n)) \wedge g(y_1, \ldots, y_m)\big),$$

for certain functions $f_1, \ldots, f_m$, and g. But his fellow student, J. H. Quick, found a much simpler formula for the same problem. What was Quick's idea?

▸ **104.** [*21*] Devise an efficient way to decide whether $f \leq g$ or $f \geq g$ or $f \parallel g$, where $f \parallel g$ means that f and g are incomparable, given the BDDs for f and g.

105. [*25*] A Boolean function $f(x_1, \ldots, x_n)$ is called *unate* with polarities $(y_1, \ldots, y_n)$ if the function $h(x_1, \ldots, x_n) = f(x_1 \oplus y_1, \ldots, x_n \oplus y_n)$ is monotone.
a) Show that f can be tested for unateness by using the $\curlywedge$ and N quantifiers.
b) Design a recursive algorithm to test unateness in at most $O(B(f)^2)$ steps, given the BDD for f. If f is unate, your algorithm should also find appropriate polarities.

subcube
approximating function
3-colored
NP-complete
existential quantification
universal quantification
differential quantification
components
Bryant
symmetry breaking
4-color
contiguous USA
cache
functional composition
projection function
replacement functions
Dull
Quick
notation: $f \parallel g$
unate
monotone
yes/no quantifiers
recursive algorithm

106. [*25*] Let $f\$g\$h$ denote the relation "$f(x) = g(y) = 1$ implies $h(x \wedge y) = 1$, for all x and y." Show that this relation can be evaluated in at most $O(B(f)B(g)B(h))$ steps. [*Motivation:* Theorem 7.1.1H states that f is a Horn function if and only if $f\$f\$f$; thus we can test Horn-ness in $O(B(f)^3)$ steps.]

107. [*26*] Continuing exercise 106, show that it's possible to determine whether or not f is a Krom function in $O(B(f)^4)$ steps. [*Hint:* See Theorem 7.1.1S.]

108. [*HM24*] Let $b(n,s)$ be the number of n-variable Boolean functions with $B(f) \le s$. Prove that $(s-3)!\,b(n,s) \le (n(s-1)^2)^{s-2}$ when $s \ge 3$, and explore the ramifications of this inequality when $s = \lfloor 2^n/(n + 1/\ln 2)\rfloor$. *Hint:* See the proof of Theorem 7.1.2S.

▸ **109.** [*HM17*] Continuing exercise 108, show that almost all Boolean functions of n variables have $B(f^\pi) > 2^n/(n+1/\ln 2)$, for all permutations π of $\{1,\ldots,n\}$, as $n \to \infty$.

110. [*25*] Construct explicit worst-case functions f_n for which $f_n = U_n$ in Theorem U.

111. [*M21*] Verify the summation formula (79) in Theorem U.

112. [*HM23*] Prove that $\min(2^k, 2^{2^{n-k}} - 2^{2^{n-k-1}}) - \hat{b}_k$ is very small, where $\hat{b}_k$ is the number defined in (80), except when $n - \lg n - 1 < k < n - \lg n + 1$.

113. [*20*] Instead of having sink nodes, one for each Boolean constant, we could have 2^{16} sinks, one for each Boolean function of four variables. Then a BDD could stop four levels earlier, after branching on x_{n-4}. Would this be a good idea?

114. [*20*] Is there a function with profile $(1,1,1,1,1,2)$ and quasi-profile $(1,2,3,4,3,2)$?

▸ **115.** [*M22*] Prove the quasi-profile inequalities (84) and (124).

116. [*M21*] What is the (a) worst case (b) average case of a random quasi-profile?

117. [*M20*] Compare $Q(f)$ to $B(f)$ when $f = M_m(x_1,\ldots,x_m; x_{m+1},\ldots,x_{m+2^m})$.

118. [*M23*] Show that, from the perspective of Section 7.1.2, the hidden weighted bit function has cost $C(h_n) = O(n)$. What is the exact value of $C(h_4)$?

119. [*20*] True or false: Every symmetric Boolean function of n variables is a special case of h_{2n+1}. (For example, $x_1 \oplus x_2 = h_5(0,1,0,x_1,x_2)$.)

120. [*18*] Explain the hidden-permuted-weighted-bit formula (94).

▸ **121.** [*M22*] If $f(x_1,\ldots,x_n)$ is any Boolean function, its *dual* f^D is $\bar{f}(\bar{x}_1,\ldots,\bar{x}_n)$, and its *reflection* f^R is $f(x_n \ldots, x_1)$. Notice that $f^{DD} = f^{RR} = f$ and $f^{DR} = f^{RD}$.

a) Show that $h_n^{DR}(x_1,\ldots,x_n) = h_n(x_2,\ldots,x_n,x_1)$.

b) Furthermore, the hidden weighted bit function satisfies the recurrence

$$h_1(x_1) = x_1, \quad h_{n+1}(x_1,\ldots,x_{n+1}) = (x_{n+1}?\ h_n(x_2,\ldots,x_n,x_1) \colon h_n(x_1,\ldots,x_n)).$$

c) Define $x\psi$, a permutation on the set of all binary strings x, by the recursive rules

$$\epsilon\psi = \epsilon, \quad (x_1\ldots x_n 0)\psi = (x_1\ldots x_n\psi)0, \quad (x_1\ldots x_n 1)\psi = (x_2\ldots x_n x_1)\psi 1.$$

For example, $1101\psi = (101\psi)1 = (01\psi)11 = (0\psi)111 = (\psi)0111 = 0111$; and we also have $0111\psi = 1101$. Is ψ an involution?

d) Show that $h_n(x) = \hat{h}_n(x\psi)$, where the function $\hat{h}_n$ has a very small BDD.

122. [*27*] Construct an FBDD for h_n that has fewer than n^2 nodes, when $n > 1$.

123. [*M20*] Prove formula (97), which enumerates all slates of offset s.

▸ **124.** [*27*] Design an efficient algorithm to compute the profile and quasi-profile of h_n^π, given a permutation π. *Hint:* When does the slate $[r_0,\ldots,r_{n-k}]$ correspond to a bead?

Horn function
sinks, more than two
2^m-way multiplexer
hidden weighted bit function
cost
$C(f)$, see cost of a Boolean function
symmetric Boolean function
dual
reflection
recurrence
recursive
involution
FBDD
profile
quasi-profile
slate
bead

▶ **125.** [*HM34*] Prove that $B(h_n)$ can be expressed exactly in terms of the sequences

$$A_n = \sum_{k=0}^{n} \binom{n-k}{2k}, \qquad B_n = \sum_{k=0}^{n} \binom{n-k}{2k+1}.$$

126. [*HM42*] Analyze $B(h_n^\pi)$ for the organ-pipe permutation $\pi = (2, 4, \ldots, n, \ldots, 3, 1)$.

127. [*46*] Find a permutation π that minimizes $B(h_{100}^\pi)$.

▶ **128.** [*25*] Given a permutation π of $\{1, \ldots, m + 2^m\}$, explain how to compute the profile and quasi-profile of the permuted 2^m-way multiplexer

$$M_m^\pi(x_1, \ldots, x_m; x_{m+1}, \ldots, x_{m+2^m}) = M_m(x_{1\pi}, \ldots, x_{m\pi}; x_{(m+1)\pi}, \ldots, x_{(m+2^m)\pi}).$$

129. [*M25*] Define $Q_m(x_1, \ldots, x_{m^2})$ to be 1 if and only if the 0–1 matrix $(x_{(i-1)m+j})$ has no all-zero row and no all-zero column. Prove that $B(Q_m^\pi) = \Omega(2^m/m^2)$ for all π.

130. [*HM31*] The adjacency matrix of an undirected graph G on vertices $\{1, \ldots, m\}$ consists of $\binom{m}{2}$ variable entries $x_{uv} = [u \longrightarrow v \text{ in } G]$, for $1 \le u < v \le m$. Let $C_{m,k}$ be the Boolean function [G has a k-clique], for some ordering of those $\binom{m}{2}$ variables.

a) If $1 < k \le \sqrt{m}$, prove that $B(C_{m,k}) \ge \binom{s+t}{s}$, where $s = \binom{k}{2} - 1$ and $t = m+2-k^2$.

b) Consequently $B(C_{m,\lceil m/2 \rceil}) = \Omega(2^{m/3}/\sqrt{m})$, regardless of the variable ordering.

131. [*M28*] (*The covering function.*) The Boolean function

$$\begin{aligned}&C(x_1, x_2, \ldots, x_p; y_{11}, y_{12}, \ldots, y_{1q}, y_{21}, \ldots, y_{2q}, \ldots, y_{p1}, y_{p2}, \ldots, y_{pq})\\ &\quad = ((x_1 \wedge y_{11}) \vee (x_2 \wedge y_{21}) \vee \cdots \vee (x_p \wedge y_{p1})) \wedge \cdots \wedge ((x_1 \wedge y_{1q}) \vee (x_2 \wedge y_{2q}) \vee \cdots \vee (x_p \wedge y_{pq}))\end{aligned}$$

is true if and only if all columns of the matrix product

$$x \cdot Y \;=\; (x_1 x_2 \ldots x_p) \begin{pmatrix} y_{11} & y_{12} & \ldots & y_{1q} \\ y_{21} & y_{22} & \ldots & y_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ y_{p1} & y_{p2} & \ldots & y_{pq} \end{pmatrix}$$

are positive, i.e., when the rows of Y selected by x "cover" every column of that matrix. The reliability polynomial of C is important in the analysis of fault-tolerant systems.

a) When a BDD for C tests the variables in the order

$$x_1, y_{11}, y_{12}, \ldots, y_{1q}, x_2, y_{21}, y_{22}, \ldots, y_{2q}, \ldots, x_p, y_{p1}, y_{p2}, \ldots, y_{pq},$$

show that the number of nodes is asymptotically $pq2^{q-1}$ for fixed q as $p \to \infty$.

b) Find an ordering for which the size is asymptotically $pq2^{p-1}$ for fixed p as $q \to \infty$.

c) Prove, however, that $B_{\min}(C) = \Omega(2^{\min(p,q)/2})$ in general.

132. [*32*] What Boolean functions $f(x_1, x_2, x_3, x_4, x_5)$ have the largest $B_{\min}(f)$?

133. [*20*] Explain how to compute $B_{\min}(f)$ and $B_{\max}(f)$ from f's master profile chart.

134. [*24*] Construct the master profile chart, analogous to (102), for the Boolean function $x_1 \oplus \big((x_2 \oplus (x_1 \vee (\bar{x}_2 \wedge x_3))) \wedge (x_3 \oplus x_4)\big)$. What are $B_{\min}(f)$ and $B_{\max}(f)$? *Hint:* The identity $f(x_1, x_2, x_3, x_4) = f(x_1, x_2, \bar{x}_4, \bar{x}_3)$ saves about half the work.

135. [*M27*] For all $n \ge 4$, find a Boolean function $\theta_n(x_1, \ldots, x_n)$ that is *uniquely thin*, in the sense that $B(\theta_n^\pi) = n + 2$ for exactly one permutation π. (See (93) and (102).)

Analyze
organ-pipe permutation
permuted 2^m-way multiplexer
0–1 matrix
all-zero row
adjacency matrix
clique
covering function
2-level redundancies function, see covering function
reliability polynomial
fault-tolerant systems
five-variable functions
master profile chart
uniquely thin

▶ **136.** [*M34*] What is the master profile chart of the median-of-medians function

$$\langle\langle x_{11}x_{12}\ldots x_{1n}\rangle\langle x_{21}x_{22}\ldots x_{2n}\rangle\ldots\langle x_{m1}x_{m2}\ldots x_{mn}\rangle\rangle,$$

when m and n are odd integers? What is the best ordering? (There are mn variables.)

137. [*M38*] Given a graph, the *optimum linear arrangement problem* asks for a permutation π of the vertices that minimizes $\sum_{u-v}|u\pi-v\pi|$. Construct a Boolean function f for which this minimum value is characterized by the optimum BDD size $B_{\min}(f)$.

▶ **138.** [*M36*] The purpose of this exercise is to develop an attractive algorithm that computes the master profile chart for a function f, given f's QDD (not its BDD).

a) Explain how to find $\binom{n+1}{2}$ weights of the master profile chart from a single QDD.

b) Show that the jump-up operation can be performed easily in a QDD, without garbage collection or hashing. *Hint:* See the "bucket sort" in Algorithm R.

c) Consider the 2^{n-1} orderings of variables in which the $(i+1)$st is obtained from the ith by a jump-up from depth $\rho i+\nu i$ to depth $\nu i-1$. For example, we get

12345 21345 32145 31245 43125 41325 42135 42315 54231 52431 53241 53421 51342 51432 51243 51234

when $n=5$. Show that every k-element subset of $\{1,\ldots,n\}$ occurs at the top k levels of one of these orderings.

d) Combine these ideas to design the desired chart-construction algorithm.

e) Analyze the space and time requirements of your algorithm.

139. [*22*] Generalize the algorithm of exercise 138 so that (i) it computes a common profile chart for all functions of a BDD base, instead of a single function; and (ii) it restricts the chart to variables $\{x_a,x_{a+1},\ldots,x_b\}$, preserving $\{x_1,\ldots,x_{a-1}\}$ at the top and $\{x_{b+1},\ldots,x_n\}$ at the bottom.

140. [*27*] Explain how to find $B_{\min}(f)$ without knowing all of f's master profile chart.

141. [*30*] True or false: If X_1, X_2, …, X_m are disjoint sets of variables, then an optimum BDD ordering for the variables of $g(h_1(X_1),h_2(X_2),\ldots,h_m(X_m))$ can be found by restricting consideration to cases where the variables of each X_j are consecutive.

▶ **142.** [*HM32*] The representation of threshold functions by BDDs is surprisingly mysterious. Consider the self-dual function $f(x)=\langle x_1^{w_1}\ldots x_n^{w_n}\rangle$, where each w_j is a positive integer and $w_1+\cdots+w_n$ is odd. We observed in (28) that $B(f)=O(w_1+\cdots+w_n)^2$; and $B(f)$ is often $O(n)$ even when the weights grow exponentially, as in (29) or exercise 41.

a) Prove that when $w_1=1$, $w_k=2^{k-2}$ for $1<k\le m$, and $w_k=2^m-2^{n-k}$ for $m<k\le 2m=n$, $B(f)$ grows exponentially as $n\to\infty$, but $B_{\min}(f)=O(n^2)$.

b) Find weights $\{w_1,\ldots,w_n\}$ for which $B_{\min}(f)=\Omega(2^{\sqrt{n}/2})$.

143. [*24*] Continuing exercise 142(a), find an optimum ordering of variables for the function $\langle x_1x_2x_3^2x_4^4x_5^8x_6^{16}x_7^{32}x_8^{64}x_9^{128}x_{10}^{256}x_{11}^{512}x_{12}^{768}x_{13}^{896}x_{14}^{960}x_{15}^{992}x_{16}^{1008}x_{17}^{1016}x_{18}^{1020}x_{19}^{1022}x_{20}^{1023}\rangle$.

144. [*16*] What is the quasi-profile of the addition functions $\{f_1,f_2,f_3,f_4,f_5\}$ in (36)?

145. [*24*] Find $B_{\min}(f_1,f_2,f_3,f_4,f_5)$ and $B_{\max}(f_1,f_2,f_3,f_4,f_5)$ of those functions.

▶ **146.** [*M22*] Let $(b_0,\ldots,b_n)$ and $(q_0,\ldots,q_n)$ be a BDD base profile and quasi-profile.

a) Prove that $b_0\le\min(q_0,(b_1+q_2)(b_1+q_2-1))$, $b_1\le\min(b_0+q_0,q_2(q_2-1))$, and $b_0+b_1\ge q_0-q_2$.

b) Conversely, if b_0, b_1, q_0, and q_2 are nonnegative integers that satisfy those inequalities, there is a BDD base with such a profile and quasi-profile.

▶ **147.** [*27*] Flesh out the details of Rudell's swap-in-place algorithm, using the conventions of Algorithm U and the reference counters of exercise 82.

median-of-medians function
optimum linear arrangement problem
QDD
ruler function ρ
ν function
Analyze
disjoint decomposition
decomposition of functions
threshold functions
self-dual
exponentially
quasi-profile
Rudell
swap-in-place
reference counters

148. [*M21*] True or false: $B(f_1^\pi,\ldots,f_m^\pi)\le 2B(f_1,\ldots,f_m)$, after swapping ① ↔ ②.

149. [*M20*] (Bollig, Löbbing, and Wegener.) Show that, in addition to Theorem J$^-$, we also have $B(f_1^\pi,\ldots,f_m^\pi)\le(2^k-2)b_0+B(f_1,\ldots,f_m)$ after a jump-down operation of $k-1$ levels, when $(b_0,\ldots,b_n)$ is the profile of $\{f_1,\ldots,f_m\}$.

150. [*30*] When repeated swaps are used to implement jump-up or jump-down, the intermediate results might be much larger than the initial or final BDD. Show that variable jumps can actually be done more directly, with a method whose worst-case running time is $O\bigl(B(f_1,\ldots,f_m)+B(f_1^\pi,\ldots,f_m^\pi)\bigr)$.

151. [*20*] Suggest a way to invoke Algorithm J so that each variable is sifted just once.

152. [*25*] The hidden weighted bit function h_{100} has more than 17.5 trillion nodes in its BDD. By how much does sifting reduce this number? *Hint:* Use exercise 124, instead of actually constructing the diagrams.

153. [*30*] Put the tic-tac-toe functions $\{y_1,\ldots,y_9\}$ of exercise 7.1.2–65 into a BDD base. How many nodes are present when variables are tested in the order $x_1, x_2, \ldots, x_9$, $o_1, o_2, \ldots, o_9$, from top to bottom? What is $B_{\min}(y_1,\ldots,y_9)$?

154. [*20*] By comparing (104) to (106), can you tell how far each state was moved when it was sifted?

▶ **155.** [*25*] Let f_1 be the independent-set function (105) of the contiguous USA, and let f_2 be the corresponding kernel function (see (67)). Find orderings π of the states so that (a) $B(f_2^\pi)$ and (b) $B(f_1^\pi,f_2^\pi)$ are as small as you can make them. (Note that the ordering (110) gives $B(f_1^\pi)=339$, $B(f_2^\pi)=795$, and $B(f_1^\pi,f_2^\pi)=1129$.)

156. [*30*] Theorems J$^+$ and J$^-$ suggest that we could save reordering time by only jumping up when sifting, not bothering to jump down. Then we could eliminate steps J3, J5, J6, and J7 of Algorithm J. Would that be wise?

157. [*M24*] Show that if the $m+2^m$ variables of the 2^m-way multiplexer M_m are arranged in any order such that $B(M_m^\pi)>2^{m+1}+1$, then sifting will reduce the BDD size.

158. [*M24*] When a Boolean function $f(x_1,\ldots,x_n)$ is symmetrical in the variables $\{x_1,\ldots,x_p\}$, it's natural to expect that those variables will appear consecutively in at least one of the reorderings $f^\pi(x_1,\ldots,x_n)$ that minimize $B(f^\pi)$. Show, however, that if

$$f(x_1,\ldots,x_n)=[x_1+\cdots+x_p=\lfloor p/3\rfloor]+[x_1+\cdots+x_p=\lceil 2p/3\rceil]\,g(x_{p+1},\ldots,x_{p+m}),$$

where $p=n-m$ and $g(y_1,\ldots,y_m)$ is any nonconstant Boolean function, then $B(f^\pi)=\frac13 n^2+O(n)$ as $n\to\infty$ when $\{x_1,\ldots,x_p\}$ are consecutive in π, but $B(f^\pi)=\frac14 n^2+O(n)$ when π places about half of those variables at the beginning and half at the end.

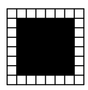
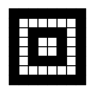
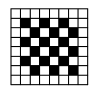
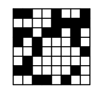
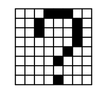
159. [*20*] John Conway's basic rule for Life, exercise 7.1.3–167, is a Boolean function $L(x_{\rm NW},x_{\rm N},x_{\rm NE},x_{\rm W},x,x_{\rm E},x_{\rm SW},x_{\rm S},x_{\rm SE})$. What ordering of those nine variables will make the BDD as small as possible?

▶ **160.** [*24*] (*Chess Life.*) Consider an 8×8 matrix $X=(x_{ij})$ of 0s and 1s, bordered by infinitely many 0s on all sides. Let $L_{ij}(X)=L(x_{(i-1)(j-1)},\ldots,x_{ij},\ldots,x_{(i+1)(j+1)})$ be Conway's basic rule at position (i,j). Call X "tame" if $L_{ij}(X)=0$ whenever $i\notin[1\,.\,.\,8]$ or $j\notin[1\,.\,.\,8]$; otherwise X is "wild," because it activates cells outside the matrix.

a) How many tame configurations X vanish in one Life step, making all $L_{ij}=0$?
b) What is the maximum weight $\sum_{i=1}^8\sum_{j=1}^8 x_{ij}$ among all such solutions?
c) How many wild configurations vanish *within* the matrix after one Life step?
d) What are the minimum and maximum weight, among all such solutions?
e) How many configurations X make $L_{ij}(X)=1$ for $1\le i,j\le 8$?

Bollig
Löbbing
Wegener
jump-up
jump-down
sifted
hidden weighted bit function
tic-tac-toe
sifted
independent-set function
contiguous USA
kernel function
jumping up
jump down
2^m-way multiplexer
partially symmetric funcs
symmetric functions, partial
reorderings
Conway
Life
0–1 matrices
tame
wild

f) Investigate the tame 8×8 predecessors of the following patterns:

(1)  (2)  (3)  (4)  (5) 

(Here, as in Section 7.1.3, black cells denote 1s in the matrix.)

161. [*28*] Continuing exercise 160, write $L(X) = Y = (y_{ij})$ if X is a tame matrix such that $L_{ij}(X) = y_{ij}$ for $1 \le i, j \le 8$.

a) How many X's satisfy $L(X) = X$ ("still Life")?
b) Find an 8×8 still Life with weight 35.
c) A "flip-flop" is a pair of distinct matrices with $L(X)=Y$, $L(Y)=X$. Count them.
d) Find a flip-flop for which X and Y both have weight 28.

▸ **162.** [*30*] (*Caged Life.*) If X and $L(X)$ are tame but $L(L(X))$ is wild, we say that X "escapes" its cage after three steps. How many 6×6 matrices escape their 6×6 cage after exactly k steps, for $k = 1, 2, \ldots$?

163. [*23*] Prove formulas (112) and (113) for the BDD sizes of read-once functions.

▸ **164.** [*M27*] What is the maximum of $B(f)$, over all read-once functions $f(x_1, \ldots, x_n)$?

165. [*M21*] Verify the Fibonacci-based formulas (115) for $B(u_m)$ and $B(v_m)$.

166. [*M29*] Complete the proof of Theorem W.

167. [*21*] Design an efficient algorithm that computes a permutation π for which both $B(f^\pi)$ and $B(f^\pi, \bar{f}^\pi)$ are minimized, given any read-once function $f(x_1, \ldots, x_n)$.

▸ **168.** [*HM40*] Consider the following binary operations on ordered pairs $z = (x, y)$:

$$z \circ z' = (x, y) \circ (x', y') = (x + x', \min(x + y', x' + y));$$
$$z \bullet z' = (x, y) \bullet (x', y') = (x + x' + \min(y, y'), \max(y, y')).$$

(These operations are associative and commutative.) Let $S_1 = \{(1, 0)\}$, and

$$S_n = \bigcup_{k=1}^{n-1} \{z \circ z' \mid z \in S_k,\ z' \in S_{n-k}\} \cup \bigcup_{k=1}^{n-1} \{z \bullet z' \mid z \in S_k,\ z' \in S_{n-k}\} \ \text{ for } n > 1.$$

Thus $S_2 = \{(2, 0), (2, 1)\}$; $S_3 = \{(3, 0), (3, 1), (3, 2)\}$; $S_4 = \{(4, 0), \ldots, (4, 3), (5, 1)\}$; etc.

a) Prove that there exists a read-once function $f(x_1, \ldots, x_n)$ for which we have $\min_\pi B(f^\pi) = c$ and $\min_\pi B(f^\pi, \bar{f}^\pi) = c'$ if and only if $(\frac{1}{2}c' - 1, c - \frac{1}{2}c' - 1) \in S_n$.
b) True or false: $0 \le y < x$ for all $(x, y) \in S_n$.
c) If $z^T = (x + y, x - y)/\sqrt{2}$, show that $z^T \circ z'^T = (z \bullet z')^T$ and $z^T \bullet z'^T = (z \circ z')^T$.
d) Prove that $x^2 + y^2 \le n^{2\beta}$ for all $(x, y) \in S_n$, if β is the constant in (116). *Hints:* Let $|z|^2 = x^2 + y^2$; it suffices to prove that $|z \bullet z'| \le 2^\beta = \sqrt{2}\phi$ whenever $0 \le y \le x$, $0 \le y' \le x'$, $|z| = r = (1-\delta)^\beta$, $|z'| = r' = (1+\delta)^\beta$, and $0 \le \delta \le 1$. If also $y = y'$, $z \bullet z'$ lies inside the ellipse $(a\cos\theta + b\sin\theta, b\sin\theta)$, where $a = r + r'$ and $b = \sqrt{rr'}$.

169. [*M46*] Is $\min_\pi B(f^\pi) \le B(v_{2m+1})$ for every read-once function f of 2^{2m+1} variables?

▸ **170.** [*M25*] Let's say that a Boolean function is "skinny" if its BDD involves all the variables in the simplest possible way: A skinny BDD has exactly one branch node (j) for each variable x_j, and either LO or HI is a sink node at every branch.

a) How many Boolean functions $f(x_1, \ldots, x_n)$ are skinny in this sense?
b) How many of them are monotone?
c) Show that $f_t(x_1, \ldots, x_n) = [(x_1 \ldots x_n)_2 \ge t]$ is skinny when $0 < t < 2^n$ and t is odd.

still Life
flip-flop
Caged Life
escapes
read-once functions
associative
commutative
read-once function
skinny
positive Boolean function, see monotone

d) What is the *dual* of the function f_t in part (c)?

e) Explain how to find the shortest CNF and DNF formulas for f_t, given t.

171. [*M26*] Continuing exercise 170, show that a function is *read-once* and *regular* if and only if it is skinny and monotone.

172. [*M27*] How many skinny functions $f(x_1,\ldots,x_n)$ are also Horn functions? How many of them have the property that f and $\bar{f}$ *both* satisfy Horn's condition?

▸ **173.** [*HM28*] Exactly how many Boolean functions $f(x_1,\ldots,x_n)$ are skinny after some reordering of the variables, $f(x_{1\pi},\ldots,x_{n\pi})$?

▸ **174.** [*M39*] Let S_n be the number of Boolean functions $f(x_1,\ldots,x_n)$ whose BDD is "thin" in the sense that it has exactly one node labeled (j) for $1 \le j \le n$. Show that S_n is also the number of combinatorial objects of the following types:

a) *Dellac permutations of order* $2n$ (namely, permutations $p_1p_2\ldots p_{2n}$ such that $\lceil k/2\rceil \le p_k \le n+\lceil k/2\rceil$ for $1\le k\le 2n$).

b) *Genocchi derangements of order* $2n+2$ (namely, permutations $q_1q_2\ldots q_{2n+2}$ such that $q_k > k$ if and only if k is odd, for $1 \le k \le 2n{+}2$; also $q_k \ne k$ in a derangement).

c) *Irreducible Dumont pistols of order* $2n+2$ (namely, sequences $r_1r_2\ldots r_{2n+2}$ such that $k \le r_k \le 2n+2$ for $1 \le k \le 2n{+}2$ and $\{r_1,r_2,\ldots,r_{2n+2}\} = \{2,4,6,\ldots,2n,2n+2\}$, with the special property that $2k \in \{r_1,\ldots,r_{2k-1}\}$ for $1\le k\le n$).

d) Paths from $(1,0)$ to $(2n+2,0)$ in the directed graph

$$\begin{tikzcd}
 & & & & & & (7,3) \arrow[r] & (8,3) \arrow[r] \arrow[d] & \cdots \\
 & & & & (5,2) \arrow[r] & (6,2) \arrow[r] \arrow[d] & (7,2) \arrow[r] \arrow[u] & (8,2) \arrow[r] \arrow[d] & \cdots \\
 & & (3,1) \arrow[r] & (4,1) \arrow[r] \arrow[d] & (5,1) \arrow[r] \arrow[u] & (6,1) \arrow[r] \arrow[d] & (7,1) \arrow[r] \arrow[u] & (8,1) \arrow[r] \arrow[d] & \cdots \\
(1,0) \arrow[r] & (2,0) \arrow[r] & (3,0) \arrow[r] \arrow[u] & (4,0) \arrow[r] & (5,0) \arrow[r] \arrow[u] & (6,0) \arrow[r] & (7,0) \arrow[r] \arrow[u] & (8,0) \arrow[r] & \cdots
\end{tikzcd} .$$

(Notice that objects of type (d) are very easy to count.)

175. [*M30*] Continuing exercise 174, find a way to enumerate the Boolean functions whose BDD contains exactly b_{j-1} nodes labeled (j), given a profile $(b_0,\ldots,b_{n-1},b_n)$.

176. [*M35*] To complete the proof of Theorem X, we will use exercise 6.4–78, which states that $\{h_{a,b} \mid a\in A \text{ and } b \in B\}$ is a universal family of hash functions from n bits to l bits, when $h_{a,b}(x) = ((ax+b)\gg(n-l)) \bmod 2^l$, $A=\{a \mid 0<a<2^n,\ a \text{ odd}\}$, $B = \{b \mid 0 \le b < 2^{n-l}\}$, and $0\le l\le n$. Let $I = \{h_{a,b}(p) \mid p\in P\}$ and $J=\{h_{a,b}(q)\mid q\in Q\}$.

a) Show that if $2^l - 1 \le 2^{t-1}\epsilon/(1-\epsilon)$, there are constants $a\in A$ and $b \in B$ for which $|I| \ge (1-\epsilon)2^l$ and $|J|\ge(1-\epsilon)2^l$.

b) Given such an a, let $J=\{j_1,\ldots,j_{|J|}\}$ where $0=j_1<\cdots<j_{|J|}$, and choose $Q' = \{q_1,\ldots,q_{|J|}\}\subseteq Q$ so that $h_{a,b}(q_k)=j_k$ for $1\le k\le |J|$. Let $g(q)$ denote the middle $l-1$ bits of aq, namely $(aq\gg(n-l+1)) \bmod 2^{l-1}$. Prove that $g(q)\ne g(q')$ whenever q and q' are distinct elements of the set $Q''=\{q_1,q_3,\ldots,q_{2\lceil |J|/2\rceil-1}\}$.

c) Prove that the following set Q^* satisfies condition (120), when $l\ge 3$ and $y=a$:

$$Q^* = \{q \mid q\in Q'',\ g(q) \text{ is even, and } g(p)+g(q)=2^{l-1} \text{ for some } p\in P\}.$$

d) Finally, show that $|Q^*|$ is large enough to prove Theorem X.

177. [*M22*] Complete the proof of Theorem A by bounding the entire quasi-profile.

178. [*M24*] (Amano and Maruoka.) Improve the constant in (121) by using a better variable ordering: $Z_n(x_{2n-1},x_1,x_3,\ldots,x_{2n-3};x_{2n},x_2,x_4,\ldots,x_{2n-2})$.

dual
CNF
DNF
read-once
regular
Horn functions
reordering
thin
Dellac permutations
permutations
Genocchi derangements
Dumont pistols
profile
universal hashing
multiplication
quasi-profile
Amano
Maruoka

179. [*M47*] Does the middle bit of multiplication satisfy $B_{\min}(Z_n)=\Theta(2^{6n/5})$?

180. [*M27*] Prove Theorem Y, using the hint given in the text.

181. [*M21*] Let $L_{m,n}$ be the *leading bit function* $Z_{m,n}^{(m+n)}(x_1,\ldots,x_m;y_1,\ldots,y_n)$. Prove that $B_{\min}(L_{m,n})=O(2^m n)$ when $m\le n$.

182. [*M38*] (I. Wegener.) Does $B_{\min}(L_{n,n})$ grow exponentially as $n\to\infty$?

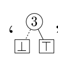
▶ **183.** [*M25*] Draw the first few levels of the BDD for the "limiting leading bit function"

$$[(.x_1x_3x_5\ldots)_2\cdot(.x_2x_4x_6\ldots)_2\ge\tfrac12],$$

which has infinitely many Boolean variables. How many nodes b_k are there on level k? (We don't allow $(.x_1x_3x_5\ldots)_2$ or $(.x_2x_4x_6\ldots)_2$ to end with infinitely many 1s.)

184. [*M23*] What are the BDD and ZDD profiles of the permutation function P_m?

185. [*M25*] How large can $Z(f)$ be, when f is a symmetric Boolean function of n variables? (See exercise 44.)

186. [*10*] What Boolean function of $\{x_1,x_2,x_3,x_4,x_5,x_6\}$ has the ZDD ''?

▶ **187.** [*20*] Draw the ZDDs for all 16 Boolean functions $f(x_1,x_2)$ of two variables.

188. [*16*] Express the 16 Boolean functions $f(x_1,x_2)$ as families of subsets of $\{1,2\}$.

189. [*18*] What functions $f(x_1,\ldots,x_n)$ have a ZDD equal to their BDD?

190. [*20*] Describe all functions f for which (a) $Q(f)=B(f)$; (b) $Q(f)=Z(f)$.

▶ **191.** [*HM25*] How many functions $f(x_1,\ldots,x_n)$ have no $\boxed{\perp}$ in their ZDD?

192. [*M20*] Define the Z-transform of binary strings as follows: $\epsilon^Z=\epsilon$, $0^Z=0$, $1^Z=1$, and

$$(\alpha\beta)^Z=\begin{cases}\alpha^Z\alpha^Z, & \text{if } |\alpha|=n \text{ and } \beta=0^n;\\ \alpha^Z0^n, & \text{if } |\alpha|=n \text{ and } \beta=\alpha;\\ \alpha^Z\beta^Z, & \text{if } |\alpha|=|\beta|-1, \text{ or if } |\alpha|=|\beta|=n \text{ and } \alpha\ne\beta\ne0^n.\end{cases}$$

a) What is 11001001000011111^Z?
b) True or false: $(\tau^Z)^Z=\tau$ for all binary strings τ.
c) If $f(x_1,\ldots,x_n)$ is a Boolean function with truth table τ, let $f^Z(x_1,\ldots,x_n)$ be the Boolean function whose truth table is τ^Z. Show that the profile of f is almost identical to the z-profile of f^Z, and vice versa. (Therefore Theorem U holds for ZDDs as well as for BDDs, and statistics such as (80) are valid also for z-profiles.)

193. [*M21*] Continuing exercise 192, what is $S_k^Z(x_1,\ldots,x_n)$ when $0\le k\le n$?

194. [*M25*] How many $f(x_1,\ldots,x_n)$ have the z-profile $(1,\ldots,1)$? (See exercise 174.)

195. [*24*] Find $Z(M_2)$, $Z_{\min}(M_2)$, and $Z_{\max}(M_2)$, where M_2 is the 4-way multiplexer.

196. [*M21*] Find a function $f(x_1,\ldots,x_n)$ for which $Z(f)=O(n)$ and $Z(\bar f)=\Omega(n^2)$.

197. [*25*] Modify the algorithm of exercise 138 so that it computes the "master z-profile chart" of f. (Then $Z_{\min}(f)$ and $Z_{\max}(f)$ can be found as in exercise 133.)

▶ **198.** [*23*] Explain how to compute AND(f,g) with ZDDs instead of BDDs (see (55)).

199. [*21*] Similarly, implement (a) OR(f,g), (b) XOR(f,g), (c) BUTNOT(f,g).

200. [*21*] And similarly, implement MUX(f,g,h) for ZDDs (see (62)).

201. [*22*] The projection functions x_j each have a simple 3-node BDD, but their ZDD representations are more complicated. What's a good way to implement these functions in a general-purpose ZDD toolkit?

leading bit function
Wegener
ZDD profile
permutation function P_m
symmetric Boolean function
ZDD
Binary Boolean operations
Two-variable functions
families of subsets
$\boxed{\perp}$
Z-transform
pi, as source
$f^Z(x_1,\ldots,x_n)$
z-profile
average nodes on level k
multiplexer
complementation
master z-profile chart
ternary operation
MUX
projection functions
ZDD versus BDD
ZDD toolkit

202. [*24*] What changes are needed to the swap-in-place algorithm of exercise 147, when levels $(u) \leftrightarrow (v)$ are being interchanged in a ZDD base instead of a BDD base?

▸ **203.** [*M24*] (*Family algebra.*) The following algebraic conventions are useful for dealing with finite families of finite subsets of positive integers, and with their representation as ZDDs. The simplest such families are the *empty family*, denoted by $\emptyset$ and represented by $\boxed{\bot}$; the *unit family* $\{\emptyset\}$, denoted by ϵ and represented by $\boxed{\top}$; and the *elementary families* $\{\{j\}\}$ for $j \geq 1$, denoted by e_j and represented by a branch node (j) with LO $= \boxed{\bot}$ and HI $= \boxed{\top}$. (Exercise 186 illustrates the ZDD for e_3.)

Two families f and g can be combined with the usual set operations:

- The *union* $f \cup g = \{\alpha \mid \alpha \in f \text{ or } \alpha \in g\}$ is implemented by OR(f, g);
- The *intersection* $f \cap g = \{\alpha \mid \alpha \in f \text{ and } \alpha \in g\}$ is implemented by AND(f, g);
- The *difference* $f \setminus g = \{\alpha \mid \alpha \in f \text{ and } \alpha \notin g\}$ is implemented by BUTNOT(f, g);
- The *symmetric difference* $f \oplus g = (f \setminus g) \cup (g \setminus f)$ is implemented by XOR(f, g).

And we also define three new ways to construct families of subsets:

- The *join* $f \sqcup g = \{\alpha \cup \beta \mid \alpha \in f \text{ and } \beta \in g\}$, sometimes written just fg;
- The *meet* $f \sqcap g = \{\alpha \cap \beta \mid \alpha \in f \text{ and } \beta \in g\}$;
- The *delta* $f \boxplus g = \{\alpha \oplus \beta \mid \alpha \in f \text{ and } \beta \in g\}$.

All three are commutative and associative: $f \sqcup g = g \sqcup f$, $f \sqcup (g \sqcup h) = (f \sqcup g) \sqcup h$, etc.

a) Suppose $f = \{\emptyset, \{1,2\}, \{1,3\}\} = \epsilon \cup (e_1 \sqcup (e_2 \cup e_3))$ and $g = \{\{1,2\}, \{3\}\} = (e_1 \sqcup e_2) \cup e_3$. What are $f \sqcup g$ and $(f \sqcap g) \setminus (f \boxplus e_1)$?

b) Any family f can also be regarded as a Boolean function $f(x_1, x_2, \ldots)$, where $\alpha \in f \iff f([1 \in \alpha], [2 \in \alpha], \ldots) = 1$. Describe the operations $\sqcup$, $\sqcap$, and $\boxplus$ in terms of Boolean logical formulas.

c) Which of the following formulas hold for all families f, g, and h? (i) $f \sqcup (g \cup h) = (f \sqcup g) \cup (f \sqcup h)$; (ii) $f \sqcap (g \cup h) = (f \sqcap g) \cup (f \sqcap h)$; (iii) $f \sqcup (g \sqcap h) = (f \sqcup g) \sqcap (f \sqcup h)$; (iv) $f \cup (g \sqcup h) = (f \cup g) \sqcup (f \cup h)$; (v) $f \boxplus \emptyset = \emptyset \sqcap g = h \sqcup \emptyset$; (vi) $f \sqcap \epsilon = \epsilon$.

d) We say that f and g are *orthogonal*, written $f \perp g$, if $\alpha \cap \beta = \emptyset$ for all $\alpha \in f$ and all $\beta \in g$. Which of the following statements is true for all families f and g? (i) $f \perp g \iff f \sqcap g = \epsilon$; (ii) $f \perp g \implies |f \sqcup g| = |f||g|$; (iii) $|f \sqcup g| = |f||g| \implies f \perp g$; (iv) $f \perp g \iff f \sqcup g = f \boxplus g$.

e) Describe all families f for which the following statements hold: (i) $f \cup g = g$ for all g; (ii) $f \sqcup g = g$ for all g; (iii) $f \sqcap g = g$ for all g; (iv) $f \sqcup (e_1 \sqcup e_2) = f$; (v) $f \sqcup (e_1 \cup e_2) = f$; (vi) $f \boxplus ((e_1 \sqcup e_2) \cup e_3) = f$; (vii) $f \boxplus f = \epsilon$; (viii) $f \sqcap f = f$.

▸ **204.** [*M25*] Continuing exercise 203, two further operations are also important:

- the *quotient* $f/g = \{\alpha \mid \alpha \cup \beta \in f \text{ and } \alpha \cap \beta = \emptyset, \text{ for all } \beta \in g\}$.
- the *remainder* $f \bmod g = f \setminus (g \sqcup (f/g))$.

The quotient is sometimes also called the "cofactor" of f with respect to g.

a) Prove that $f/(g \cup h) = (f/g) \cap (f/h)$.

b) Suppose $f = \{\{1,2\}, \{1,3\}, \{2\}, \{3\}, \{4\}\}$. What are f/e_2 and $f/(f/e_2)$?

c) Simplify the expressions $f/\emptyset$, f/ϵ, f/f, and $(f \bmod g)/g$, for arbitrary f and g.

d) Show that $f/g = f/(f/(f/g))$. *Hint:* Start with the relation $g \subseteq f/(f/g)$.

e) Prove that f/g can also be defined as $\bigcup\{h \mid g \sqcup h \subseteq f \text{ and } g \perp h\}$.

f) Given f and j, show that f has a unique representation $(e_j \sqcup g) \cup h$ with $e_j \perp (g \cup h)$.

g) True or false: $(f \sqcup g) \bmod e_j = (f \bmod e_j) \sqcup (g \bmod e_j)$; $(f \sqcap g)/e_j = (f/e_j) \sqcap (g/e_j)$.

205. [*M25*] Implement the five basic operations of family algebra, namely (a) $f \sqcup g$, (b) $f \sqcap g$, (c) $f \boxplus g$, (d) f/g, and (e) $f \bmod g$, using the conventions of exercise 198.

swap-in-place
Family algebra
unate cube set algebra, see family algebra
empty family
unit family
ϵ
elementary families
e_j
union
intersection
difference
symmetric difference
notation $f \sqcup g$
join
notation $f \sqcap g$
meet
notation $f \boxplus g$
delta
commutative
associative
Boolean functions versus families
distributive law
orthogonal
quotient
notation f/g
remainder
notation $f \bmod g$
cofactor

206. [*M46*] What are the worst-case running times of the algorithms in exercise 205?

▸ **207.** [*M25*] When one or more projection functions x_j are needed in applications, as in exercise 201, the following "symmetrizing" operation turns out to be very handy:

$$(e_{i_1} \cup e_{i_2} \cup \cdots \cup e_{i_l}) \S k \;=\; S_k(x_{i_1}, x_{i_2}, \ldots, x_{i_l}), \qquad \text{integer } k \geq 0.$$

For example, $e_j \S 1 = x_j$; $e_j \S 0 = \bar{x}_j$; $(e_i \cup e_j) \S 1 = x_i \oplus x_j$; $(e_2 \cup e_3 \cup e_5) \S 2 = (x_2 \wedge x_3 \wedge \bar{x}_5) \vee (x_2 \wedge \bar{x}_3 \wedge x_5) \vee (\bar{x}_2 \wedge x_3 \wedge x_5)$. Show that it's easy to implement this operation. (Notice that $e_{i_1} \cup \cdots \cup e_{i_l}$ has a very simple ZDD of size $l+2$, when $l > 0$.)

▸ **208.** [*16*] By modifying Algorithm C, show that all solutions of a Boolean function can readily be counted when its ZDD is given instead of its BDD.

209. [*M21*] Explain how to compute the fully elaborated truth table of a Boolean function from its ZDD representation. (See exercise 31.)

▸ **210.** [*23*] Given the ZDD for f, show how to construct the ZDD for the function

$$g(x) = [f(x) = 1 \text{ and } \nu x = \max\{\nu y \mid f(y) = 1\}].$$

211. [*M20*] When f describes the solutions to an exact cover problem, is $Z(f) \leq B(f)$?

▸ **212.** [*25*] What's a good way to compute the ZDD for an exact cover problem?

213. [*16*] Why can't the mutilated chessboard be perfectly covered with dominoes?

▸ **214.** [*21*] When some shape is covered by dominoes, we say that the covering is *faultfree* if every straight line that passes through the interior of the shape also passes through the interior of some domino. For example, the right-hand covering in (127) is faultfree, but the middle one isn't; and the left-hand one has faults galore.

How many domino coverings of a chessboard are faultfree?

215. [*21*] Japanese tatami mats are 1×2 rectangles that are traditionally used to cover rectangular floors in such a way that no four mats meet at any corner. For example, Fig. 29(a) shows a 6×5 pattern from the 1641 edition of Mitsuyoshi Yoshida's *Jinkōki*, a book first published in 1627.

Find all domino coverings of a chessboard that are also tatami tilings.

(a) (b)

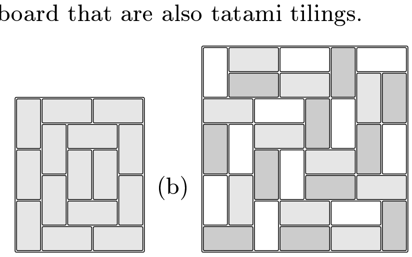


Fig. 29. Two nice examples:
(a) A 17th-century tatami tiling;
(b) a tricolored domino covering.

▸ **216.** [*30*] Figure 29(b) shows a chessboard covered with red, white, and blue dominoes, in such a way that no two dominoes of the same color are next to each other.

a) In how many ways can this be done?
b) How many of the 12,988,816 domino coverings are 3-colorable?

217. [*29*] The monomino/domino/tromino covering illustrated in (130) happens to satisfy an additional constraint: *No two congruent pieces are adjacent.* How many of the 92 sextillion coverings mentioned in the text are "separated," in this sense?

▸ **218.** [*24*] Apply BDD and ZDD techniques to the problem of Langford pairs, discussed at the beginning of this chapter.

analysis of algs
projection functions
symmetrizing
symmetric functions
fully elaborated truth table
ZDD
exact cover problem
mutilated chessboard
dominoes
faultfree
Yoshida
tatami tilings
3-colorable
monomino
tromino
polyominoes
separated
Langford pairs

219. [*20*] What is $Z(F)$ when F is the family (a) WORDS(1000); ...; (e) WORDS(5000)?

▶ **220.** [*21*] The z-profile of the 5757 SGB words, represented with 130 variables $a_1 \mathinner{..} z_5$ as discussed in (131), is (1, 1, 1, ..., 1, 1, 1, 23, 3, ..., 6, 2, 0, 3, 2, 1, 1, 2).

a) Explain the entries 23 and 3, which correspond to the variables a_2 and b_2.

b) Explain the final entries 0, 3, 2, 1, 1, 2, which correspond to v_5, w_5, x_5, etc.

▶ **221.** [*M27*] Only 5020 nodes are needed to represent the 5757 most common five-letter words of English, using the 130-variable representation, because of special linguistic properties. But there are $26^5 = 11{,}881{,}376$ possible five-letter words. Suppose we choose 5757 of them at random; how big will the ZDD be then, on average?

▶ **222.** [*27*] When family algebra is applied to five-letter words as in (131), the 130 variables are called a_1, b_1, ..., z_5 instead of x_1, x_2, ..., x_{130}; and the corresponding elementary families are denoted by the symbols $\mathtt{a}_1, \mathtt{b}_1, \ldots, \mathtt{z}_5$ instead of $e_1, e_2, \ldots, e_{130}$. Thus the family $F =$ WORDS(5757) can be constructed by synthesizing the formula

$$F = (\mathtt{w}_1 \sqcup \mathtt{h}_2 \sqcup \mathtt{i}_3 \sqcup \mathtt{c}_4 \sqcup \mathtt{h}_5) \cup \cdots \cup (\mathtt{f}_1 \sqcup \mathtt{u}_2 \sqcup \mathtt{n}_3 \sqcup \mathtt{n}_4 \sqcup \mathtt{y}_5) \cup \cdots \cup (\mathtt{p}_1 \sqcup \mathtt{u}_2 \sqcup \mathtt{p}_3 \sqcup \mathtt{a}_4 \sqcup \mathtt{l}_5).$$

a) Let $\wp$ denote the *universal family* of all subsets of $\{a_1, \ldots, z_5\}$, also called the "power set." What does the formula $F \sqcap \wp$ signify?

b) Let $X = X_1 \sqcup \cdots \sqcup X_5$, where $X_j = \{\mathtt{a}_j, \mathtt{b}_j, \ldots, \mathtt{z}_j\}$. Interpret the formula $F \sqcap X$.

c) Find a simple formula for all words of F that match the pattern t*u*h.

d) Find a formula for all SGB words that contain exactly k vowels, for $0 \le k \le 5$ (considering only a, e, i, o, and u to be vowels). Let $V_j = \mathtt{a}_j \cup \mathtt{e}_j \cup \mathtt{i}_j \cup \mathtt{o}_j \cup \mathtt{u}_j$.

e) How many patterns in which exactly three letters are specified are matched by at least one SGB word? (For example, m*tc* is such a pattern.) Give a formula.

f) How many of those patterns are matched at least twice (e.g., *atc*)?

g) Express all words that remain words when a 'b' is changed to 'o'.

h) What's the significance of the formula F/V_2?

i) Contrast $(X_1 \sqcup V_2 \sqcup V_3 \sqcup V_4 \sqcup X_5) \cap F$ with $(X_1 \sqcup X_5) \setminus ((\wp \setminus F)/(V_2 \sqcup V_3 \sqcup V_4))$.

223. [*28*] A "median word" is a five-letter word $\mu = \mu_1 \ldots \mu_5$ that can be obtained from three words $\alpha = \alpha_1 \ldots \alpha_5$, $\beta = \beta_1 \ldots \beta_5$, $\gamma = \gamma_1 \ldots \gamma_5$ by the rule $[\alpha_i = \mu_i] + [\beta_i = \mu_i] + [\gamma_i = \mu_i] = 2$ for $1 \le i \le 5$. For example, **mixed** is a median of the words {**fixed**, **mixer**, **mound**}, and also of {**mated**, **mixup**, **nixed**}. But **noted** is not a median of {**notes**, **voted**, **naked**}, because each of those words has e in position 4.

a) Show that $\{d(\alpha,\mu), d(\beta,\mu), d(\gamma,\mu)\}$ is either $\{1,1,3\}$ or $\{1,2,2\}$ whenever μ is a median of $\{\alpha,\beta,\gamma\}$. (Here d denotes Hamming distance.)

b) How many medians can be obtained from WORDS(n), when $n = 100$? 1000? 5757?

c) How many of those medians belong to WORDS(m), when $m = 100$? 1000? 5757?

▶ **224.** [*20*] Suppose we form the ZDD for all source-to-sink paths in a dag, as in Fig. 28, when the dag happens to be a forest; that is, assume that every non-source vertex of the dag has in-degree 1. Show that the corresponding ZDD is essentially the same as the binary tree that represents the forest under the "natural correspondence between forests and binary trees," Eqs. 2.3.2–(1) through 2.3.2–(3).

▶ **225.** [*30*] Design an algorithm that will produce a ZDD for all sets of edges that form a simple path from s to t, given a graph and two distinct vertices $\{s,t\}$ of the graph.

▶ **226.** [*20*] Modify the algorithm of exercise 225 so that it yields a ZDD for all of the simple *cycles* in a given graph.

227. [*20*] Similarly, modify it so that it considers only *Hamiltonian paths* from s to t.

z-profile
SGB words
family algebra
five-letter words
$\wp$
universal family
power set
vowels
SGB word: A word in WORDS(5757)
median word
Hamming distance
forest
binary tree
natural correspondence between forests and binary trees
cycles
Hamiltonian paths

228. [*21*] And mutate it once more, for Hamiltonian paths from s to *any* other vertex.

229. [*15*] There are 587,218,421,488 paths from `CA` to `ME` in the graphs (18), but only 437,525,772,584 such paths in (133). Explain the discrepancy.

230. [*25*] Find the Hamiltonian paths of (133) that have minimum and maximum total length. What is the *average* length, if all Hamiltonian paths are equally likely?

231. [*23*] In how many ways can a king travel from one corner of a chessboard to the opposite corner, never occupying the same cell twice? (These are the simple paths from corner to corner of the graph $P_8 \boxtimes P_8$.)

▸ **232.** [*23*] Continuing exercise 231, a *king's tour* of the chessboard is an oriented Hamiltonian cycle of $P_8 \boxtimes P_8$. Determine the exact number of king's tours. What is the longest possible king's tour, in terms of Euclidean distance traveled?

▸ **233.** [*25*] Design an algorithm that builds a ZDD for the family of all *oriented cycles* of a given digraph. (See exercise 226.)

234. [*22*] Apply the algorithm of exercise 233 to the directed graph on the 49 postal codes `AL`, `AR`, ..., `WY` of (18), with `XY` $\longrightarrow$ `YZ` as in exercise 7–54(b). For example, one such oriented cycle is `NC` $\longrightarrow$ `CT` $\longrightarrow$ `TN` $\longrightarrow$ `NC`. How many oriented cycles are possible? What are the minimum and maximum cycle lengths?

235. [*22*] Form a digraph on the five-letter words of English by saying that $x \longrightarrow y$ when the last three letters of x match the first three letters of y (e.g., `crown` $\longrightarrow$ `owner`). How many oriented cycles does this digraph have? What are the longest and shortest?

▸ **236.** [*M25*] Many extensions to the family algebra of exercise 203 suggest themselves when ZDDs are applied to combinatorial problems, including the following five operations on families of sets:

- The *maximal elements* $f^{\uparrow} = \{\alpha \in f \mid \beta \in f \text{ and } \alpha \subseteq \beta \text{ implies } \alpha = \beta\}$;
- The *minimal elements* $f^{\downarrow} = \{\alpha \in f \mid \beta \in f \text{ and } \alpha \supseteq \beta \text{ implies } \alpha = \beta\}$;
- The *nonsubsets* $f \nearrow g = \{\alpha \in f \mid \beta \in g \text{ implies } \alpha \not\subseteq \beta\}$;
- The *nonsupersets* $f \searrow g = \{\alpha \in f \mid \beta \in g \text{ implies } \alpha \not\supseteq \beta\}$;
- The *cross elements* $f^{\sharp} = \{\alpha \mid \beta \in f \text{ implies } \alpha \cap \beta \neq \emptyset\}^{\downarrow}$.

For example, when f and g are the families of exercise 203(a) we have $f^{\uparrow} = e_1 \sqcup (e_2 \cup e_3)$, $f^{\downarrow} = \epsilon$, $f^{\sharp} = \emptyset$, $g^{\uparrow} = g^{\downarrow} = g$, $g^{\sharp} = (e_1 \cup e_2) \sqcup e_3$, $f \nearrow g = e_1 \sqcup e_3$, $f \searrow g = \epsilon$, $g \nearrow f = g \searrow f = \emptyset$.

a) Prove that $f \nearrow g = f \setminus (f \sqcap g)$, and give a similar formula for $f \searrow g$.

b) Let $f^C = \{\overline{\alpha} \mid \alpha \in f\} = f \boxplus U$, where $U = e_1 \sqcup e_2 \sqcup \cdots$ is the "universal set." Clearly $f^{CC} = f$, $(f \cup g)^C = f^C \cup g^C$, $(f \cap g)^C = f^C \cap g^C$, $(f \setminus g)^C = f^C \setminus g^C$. Show that we also have the duality laws $f^{\uparrow C} = f^{C\downarrow}$, $f^{\downarrow C} = f^{C\uparrow}$; $(f \sqcup g)^C = f^C \sqcap g^C$, $(f \sqcap g)^C = f^C \sqcup g^C$; $(f \nearrow g)^C = f^C \searrow g^C$, $(f \searrow g)^C = f^C \nearrow g^C$; $f^{\sharp} = (\wp \nearrow f^C)^{\downarrow}$.

c) True or false? (i) $x_1^{\downarrow} = e_1$; (ii) $x_1^{\uparrow} = e_1$; (iii) $x_1^{\sharp} = e_1$; (iv) $(x_1 \vee x_2)^{\downarrow} = e_1 \cup e_2$; (v) $(x_1 \wedge x_2)^{\downarrow} = e_1 \sqcup e_2$.

d) Which of the following formulas hold for all families f, g, and h? (i) $f^{\uparrow\uparrow} = f^{\uparrow}$; (ii) $f^{\uparrow\downarrow} = f^{\downarrow}$; (iii) $f^{\uparrow\downarrow} = f^{\uparrow}$; (iv) $f^{\downarrow\uparrow} = f^{\downarrow}$; (v) $f^{\sharp\downarrow} = f^{\sharp}$; (vi) $f^{\sharp\uparrow} = f^{\sharp}$; (vii) $f^{\downarrow\sharp} = f^{\sharp}$; (viii) $f^{\uparrow\sharp} = f^{\sharp}$; (ix) $f^{\sharp\sharp} = f^{\sharp}$; (x) $f \nearrow (g \cup h) = (f \nearrow g) \cap (f \nearrow h)$; (xi) $f \searrow (g \cup h) = (f \searrow g) \cap (f \searrow h)$; (xii) $f \searrow (g \cup h) = (f \searrow g) \searrow h$; (xiii) $f \nearrow g^{\uparrow} = f \nearrow g$; (xiv) $f \searrow g^{\uparrow} = f \searrow g$; (xv) $(f \sqcup g)^{\sharp} = (f^{\sharp} \cup g^{\sharp})^{\downarrow}$; (xvi) $(f \cup g)^{\sharp} = (f^{\sharp} \sqcup g^{\sharp})^{\downarrow}$.

e) Suppose $g = \bigcup_{u - v}(e_u \sqcup e_v)$ is the family of all edges in a graph, and let f be the family of all the independent sets. Using the operations of extended family algebra, find simple formulas that express (i) f in terms of g; (ii) g in terms of f.

average solution
king
chessboard
strong product of graphs
king's tour
Hamiltonian cycle
oriented cycles
postal codes
five-letter words of English
family algebra
notation $f^{\uparrow}$
maximal elements
notation $f^{\downarrow}$
minimal elements
notation $f \nearrow g$
nonsubsets
notation $f \searrow g$
nonsupersets
notation $f^{\sharp}$
cross elements
U
universal set
duality laws

237. [*25*] Implement the five operations of exercise 236, in the style of exercise 205.

▶ **238.** [*22*] Use ZDDs to compute the *maximal induced bipartite subgraphs* of the contiguous-USA graph G in (18), namely the maximal subsets U such that $G \mid U$ has no cycles of odd length. How many such sets U exist? Give examples of the smallest and largest. Consider also the maximal induced *tripartite* (3-colorable) subgraphs.

▶ **239.** [*21*] Explain how to compute the *maximal cliques* of a graph G using family algebra, when G is specified by its edges g as in exercise 236(e). Find the maximal sets of vertices that can be covered by k cliques, for $k = 1, 2, \ldots$, when G is the graph (18).

▶ **240.** [*22*] A set of vertices U is called a *dominating set* of a graph if every vertex is at most one step away from U.

a) Prove that every kernel of a graph is a minimal dominating set.
b) How many minimal dominating sets does the USA graph (18) have?
c) Find seven vertices of (18) that dominate 36 of the others.

▶ **241.** [*28*] The *queen graph* Q_8 consists of the 64 squares of a chessboard, with $u \mathrel{-\!\!-} v$ when squares u and v lie in the same row, column, or diagonal. How large are the ZDDs for its (a) kernels? (b) maximal cliques? (c) minimal dominating sets? (d) minimal dominating sets that are also cliques? (e) maximal induced bipartite subgraphs?

Illustrate each of these five categories by exhibiting smallest and largest examples.

242. [*24*] Find all of the maximal ways to choose points on an 8×8 grid so that no three points lie on a straight line of any slope.

243. [*M23*] The *closure* $f^\cap$ of a family f of sets is the family of all sets that can be obtained by intersecting one or more members of f.

a) Prove that $f^\cap = \{\alpha \mid \alpha = \bigcap\{\beta \mid \beta \in f \text{ and } \beta \supseteq \alpha\}\}$.
b) What's a good way to compute the ZDD for $f^\cap$, given the ZDD for f?
c) Find the generating function for $F^\cap$ when $F = \texttt{WORDS(5757)}$ as in exercise 222.

244. [*25*] What is the ZDD for the connectedness function of $P_3 \square P_3$ (Fig. 22)? What is the BDD for the spanning tree function of the same graph? (See Corollary S.)

▶ **245.** [*M22*] Show that the *prime clauses* of a monotone function f are $\mathrm{PI}(f)^\sharp$.

246. [*M21*] Prove Theorem S, assuming that (137) is true.

▶ **247.** [*M27*] Determine the number of sweet Boolean functions of n variables for $n \le 7$.

248. [*M22*] True or false: If f and g are sweet, so is $f(x_1, \ldots, x_n) \wedge g(x_1, \ldots, x_n)$.

249. [*HM31*] The connectedness function of a graph is "ultrasweet," in the sense that it is sweet under all permutations of its variables. Is there a nice way to characterize ultrasweet Boolean functions?

250. [*28*] There are 7581 monotone Boolean functions $f(x_1, x_2, x_3, x_4, x_5)$. What are the average values of $B(f)$ and $Z(\mathrm{PI}(f))$ when one of them is chosen at random? What is the probability that $Z(\mathrm{PI}(f)) > B(f)$? What is the maximum of $Z(\mathrm{PI}(f))/B(f)$?

251. [*M46*] Is $Z(\mathrm{PI}(f)) = O(B(f))$ for all monotone Boolean functions f?

252. [*M30*] When a Boolean function isn't monotone, its prime implicants involve negative literals; for example, the prime implicants of $(x_1?\ x_2\colon x_3)$ are $x_1 \wedge x_2$, $\bar{x}_1 \wedge x_3$, and $x_2 \wedge x_3$. In such cases we can conveniently represent them with ZDDs if we consider them to be words in the $2n$-letter alphabet $\{e_1, e_1', \ldots, e_n, e_n'\}$. A "subcube" such as 01∗0∗ is then $e_1' \sqcup e_2 \sqcup e_4'$ in family algebra (see 7.1.1–29); and $\mathrm{PI}(x_1?\ x_2\colon x_3) = (e_1 \sqcup e_2) \cup (e_1' \sqcup e_3) \cup (e_2 \sqcup e_3)$.

maximal induced bipartite subgraphs
bipartite subgraphs
tripartite
3-colorable
maximal cliques
clique covering
dominating set
absorbent sets, see dominating sets
minimal dominating set
USA graph
queen graph
chessboard
kernels
maximal cliques
cliques
no-three-on-a-line problem
collinear points
closure
notation $f^\cap$
closed item sets, see $f^\cap$
five-letter words
connectedness function
spanning tree function
prime clauses
$f^\sharp$
CNF
sweet Boolean functions
connectedness function
ultrasweet
monotone Boolean functions
MUX
prime implicants in general+
negative literals+
subcube
family algebra

Exercise 7.1.1–116 shows that symmetric functions of n variables might have $\Omega(3^n/n)$ prime implicants. How large can $Z(\mathrm{PI}(f))$ be when f is symmetric?

▸ **253.** [*M26*] Continuing exercise 252, prove that if $f = (\bar{x}_1 \wedge f_0) \vee (x_1 \wedge f_1)$ we have $\mathrm{PI}(f) = A \cup (e_1' \sqcup B) \cup (e_1 \sqcup C)$, where $A = \mathrm{PI}(f_0 \wedge f_1)$, $B = \mathrm{PI}(f_0) \setminus A$, and $C = \mathrm{PI}(f_2) \setminus A$. (Equation (137) is the special case when f is monotone.)

▸ **254.** [*M23*] Let the functions f and g of (52) be monotone, with $f \subseteq g$. Prove that

$$\mathrm{PI}(g) \setminus \mathrm{PI}(f) \;=\; \bigl(\mathrm{PI}(g_l) \setminus \mathrm{PI}(f_l)\bigr) \cup \bigl(\mathrm{PI}(g_h) \setminus \mathrm{PI}(f_h \cup g_l)\bigr).$$

▸ **255.** [*25*] A *multifamily* of sets, in which members of f are allowed to occur more than once, can be represented as a sequence of ZDDs $(f_0, f_1, f_2, \ldots)$ in which f_k is the family of sets that occur $(\ldots a_2 a_1 a_0)_2$ times in f where $a_k = 1$. For example, if α appears exactly $9 = (1001)_2$ times in the multifamily, α would be in f_3 and f_0.

a) Explain how to insert and delete items from this representation of a multifamily.
b) Implement the multiset union $h = f \uplus g$ for multifamilies.

256. [*M32*] Any nonnegative integer x can be represented as family of subsets of the binary powers $U = \{2^{2^k} \mid k \geq 0\} = \{2^1, 2^2, 2^4, 2^8, \ldots\}$, in the following way: If $x = 2^{e_1} + \cdots + 2^{e_t}$, where $e_1 > \cdots > e_t \geq 0$ and $t \geq 0$, the corresponding family has t sets $E_j \subseteq U$, where $2^{e_j} = \prod\{u \mid u \in E_j\}$. Conversely, every finite family of finite subsets of U corresponds in this way to a nonnegative integer x. For example, the number $41 = 2^5 + 2^3 + 1$ corresponds to the family $\{\{2^1, 2^4\}, \{2^1, 2^2\}, \emptyset\}$.

a) Find a simple connection between the binary representation of x and the truth table of the Boolean function that corresponds to the family for x.
b) Let $Z(x)$ be the size of the ZDD for the family that represents x, when the elements of U are tested in reverse order $\ldots, 2^4, 2^2, 2^1$ (with highest exponents nearest to the root); for example, $Z(41) = 5$. Show that $Z(x) = O(\log x / \log\log x)$.
c) The integer x is called "sparse" if $Z(x)$ is substantially smaller than the upper bound in (b). Prove that the sum of sparse integers is sparse, in the sense that $Z(x + y) = O(Z(x)Z(y))$.
d) Is the saturating difference of sparse integers, $x \dot{-} y$, always sparse?
e) Is the product of sparse integers always sparse?

257. [*40*] (S. Minato.) Explore the use of ZDDs to represent polynomials with nonnegative integer coefficients. *Hint:* Any such polynomial in x, y, and z can be regarded as a family of subsets of $\{2, 2^2, 2^4, \ldots, x, x^2, x^4, \ldots, y, y^2, y^4, \ldots, z, z^2, z^4, \ldots\}$; for example, $x^3 + 3xy + 2z$ corresponds naturally to the family $\{\{x, x^2\}, \{x, y\}, \{2, x, y\}, \{2, z\}\}$.

▸ **258.** [*25*] Given a positive integer n, what is the minimum size of a BDD that has exactly n solutions? Answer this question also for a ZDD of minimum size.

▸ **259.** [*25*] A sequence of *parentheses* can be can be encoded as a binary string by letting 0 represent '(' and 1 represent ')'. For example, `())(()` is encoded as 011001.

Every forest of n nodes corresponds to a sequence of $2n$ parentheses that are properly *nested*, in the sense that left and right parentheses match in the normal way. (See, for example, 2.3.3–(1) or 7.2.1.6–(1).) Let

$$N_n(x_1, \ldots, x_{2n}) = [x_1 \ldots x_{2n} \text{ represents properly nested parentheses}].$$

For example, $N_3(0,1,1,0,0,1) = 0$ and $N_3(0,0,1,0,1,1) = 1$; in general, N_n has $C_n \approx 4^n/(\sqrt{\pi}\, n^{3/2})$ solutions, where C_n is a Catalan number. What are $B(N_n)$ and $Z(N_n)$?

symmetric functions
monotone
multifamily
transaction database, see multifamily of sets
multiset union
Mathews, Edwin Lee (= 41)
truth table
$Z(x)$
sparse
sum
saturating subtraction
monus
Minato
polynomials
family of subsets
solutions
parentheses
forest
nested
Catalan number

▸ **260.** [*M27*] We will see in Section 7.2.1.5 that every partition of $\{1,\ldots,n\}$ into disjoint subsets corresponds to a "restricted growth sequence" $a_1\ldots a_n$, which is a sequence of nonnegative integers with

$$a_1 = 0 \quad\text{and}\quad a_{j+1} \le 1+\max(a_1,\ldots,a_j) \text{ for } 1\le j<n.$$

Elements j and k belong to the same subset of the partition if and only if $a_j = a_k$.

a) Let $x_{j,k} = [a_j = k]$ for $0 \le k < j \le n$, and let R_n be the function of these $\binom{n+1}{2}$ variables that is true if and only if $a_1\ldots a_n$ is a restricted growth sequence. (By studying this Boolean function we can study the family of all set partitions, and by placing further restrictions on R_n we can study set partitions with special properties. There are $\varpi_{100} \approx 5\times 10^{115}$ set partitions when $n = 100$.) Calculate $B(R_{100})$ and $Z(R_{100})$. Approximately how large are $B(R_n)$ and $Z(R_n)$ as $n\to\infty$?

b) Show that, with a proper ordering of the variables $x_{j,k}$, the BDD base for $\{R_1,\ldots,R_n\}$ has the same number of nodes as the BDD for R_n alone.

c) We can also use fewer variables, approximately $n\lg n$ instead of $\binom{n+1}{2}$, if we represent each a_k as a binary integer with $\lceil \lg k\rceil$ bits. How large are the BDD and ZDD bases in *this* representation of set partitions?

261. [*HM21*] *"The deterministic finite-state automaton with fewest states that accepts any given regular language is unique."* What is the connection between this famous theorem of automata theory and the theory of binary decision diagrams?

262. [*M26*] The determination of optimum Boolean chains in Section 7.1.2 was greatly accelerated by restricting consideration to Boolean functions that are *normal*, in the sense that $f(0,\ldots,0) = 0$. (See Eq. 7.1.2–(10).) Similarly, we could restrict BDDs so that each of their nodes denotes a normal function.

a) Explain how to do this by introducing "complement links," which point to the complement of a subfunction instead of to the subfunction itself.

b) Show that every Boolean function has a unique normalized BDD.

c) Draw the normalized BDDs for the 16 functions in exercise 1.

d) Let $B^0(f)$ be the size of the normalized BDD for f. Find the average and worst case of $B^0(f)$, and compare $B^0(f)$ to $B(f)$. (See (80) and Theorem U.)

e) The BDD base for 3×3 multiplication in (58) has $B(F_1,\ldots,F_6) = 52$ nodes. What is $B^0(F_1,\ldots,F_6)$?

f) How do (54) and (55) change, when AND is implemented with complement links?

263. [*HM25*] A *linear block code* is the set of binary column vectors $x = (x_1,\ldots,x_n)^T$ such that $Hx = 0$, where H is a given $m\times n$ "parity check matrix."

a) The linear block code with $n = 2^m - 1$, whose columns are the nonzero binary m-tuples from $(0,\ldots,0,1)^T$ to $(1,\ldots,1,1)^T$, is called the *Hamming code*. Prove that the Hamming code is 1-error correcting in the sense of exercise 7–23.

b) Let $f(x) = [Hx = 0]$, where H is an $m\times n$ matrix with no all-zero columns. Show that the BDD profile of f has a simple relation to the ranks of submatrices of H mod 2, and compute $B(f)$ for the Hamming code.

c) In general we can let $f(x) = [x \text{ is a codeword}]$ define *any* block code. Suppose some codeword $x = x_1\ldots x_n$ has been transmitted through a possibly noisy channel, and that we've received the bits $y = y_1\ldots y_n$, where the channel delivers $y_k = x_k$ with probability p_k for each k independently. Explain how to determine the most likely codeword x, given y, p_1, …, p_n, and the BDD for f.

set partitions
restricted growth sequence
ordering
BDD base
finite-state automaton
regular language
automata theory
Boolean chains
normal
complement links
normalized BDDs
AND
attributed edges, see complement links
linear block code
parity check matrix
Hamming code
error-correcting codes
profile
ranks
block code
maximum likelihood

264. [*M46*] The text's "sweeping generalization" of Algorithms B and C, based on (22), embraces many important applications; but it does not appear to include quantities such as

$$\max_{f(x)=1}\Bigl(\sum_{k=1}^{n} w_k x_k + \sum_{k=1}^{n-1} w'_k x_k x_{k+1}\Bigr) \qquad \text{or} \qquad \max_{f(x)=1}\sum_{j=0}^{n-1}\Bigl(w_j \sum_{k=1}^{n-j} x_k \ldots x_{k+j}\Bigr),$$

which also can be computed efficiently from the BDD or ZDD for f.

Develop a generalization that is even more sweeping.

generalization, sweeping
optimization
Boolean programming, generalized
ÆLFRIC
THOREAU

We dare not lengthen this book much more,
lest it be out of due proportion,
and repel men by its size.

— ÆLFRIC, *Catholic Homilies II* (c. 1000)

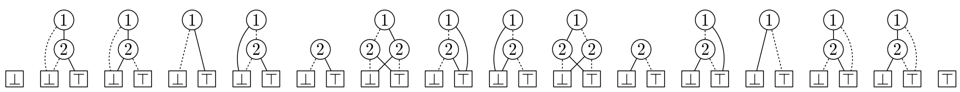
There are a thousand hacking at the branches of evil
to one who is striking at the root.

— HENRY D. THOREAU, *Walden; or, Life in the Woods* (1854)

dual
Dubrova
Macchiarulo

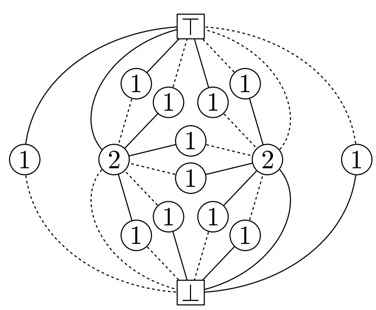
SECTION 7.1.4

1. Here are the BDDs for truth tables 0000, 0001, ..., 1111, showing the sizes below:



1 4 4 3 4 3 5 4 4 5 3 4 3 4 4 1

2. (The ordering property determines the direction of each arc.)



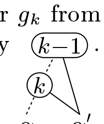
3. There are two with size 1 (namely the two constant functions); none with size 2 (because two sinks cannot both be reachable unless there's also a branch node); and $2n$ with size 3 (namely x_j and $\bar{x}_j$ for $1 \le j \le n$).

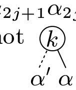
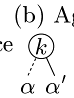
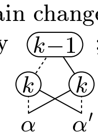
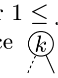
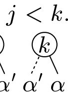
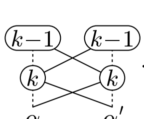
4. Set $y \leftarrow {}^{\#}\texttt{0ffffffeffffffe} \,\&\, \bar{x} + {}^{\#}\texttt{20000002}$, $y \leftarrow (y \gg 28) \,\&\, {}^{\#}\texttt{10000001}$, $x' \leftarrow x \oplus y$. (See 7.1.3–(93).)

5. You get $\overline{f(\bar{x}_1, \ldots, \bar{x}_n)} = f^D(x_1, \ldots, x_n)$, the *dual* of f (see exercise 7.1.1–2).

6. The largest subtables of 1011000110010011, namely 10110001, 10010011, 1011, 0001, 1001, 0011, are all distinct beads; squares and duplicates don't appear until we look at the subtables $\{10, 11, 00, 01\}$ of length 2. So g has size 11.

7. (a) If the truth table of f is $\alpha_0\alpha_1 \ldots \alpha_{2^k-1}$, where each α_j is a binary string of length 2^{n-k}, the truth table of g_k is $\beta_0\beta_2 \ldots \beta_{2^k-2}$, where $\beta_{2j} = \alpha_{2j}\alpha_{2j+1}\alpha_{2j+1}\alpha_{2j+1}$.

(b) Thus the beads of f and g_k are closely related. We get the BDD for g_k from the BDD for f by changing j to $j{-}1$ for $1 \le j < k$, and replacing  by .

8. (a) Now $\beta_{2j} = \alpha_{2j}\alpha_{2j+1}\alpha_{2j+1}\alpha_{2j}$. (b) Again change j to $j{-}1$ for $1 \le j < k$. If  is present in f but not , replace  by ; otherwise replace   by . [E. Dubrova and L. Macchiarulo, *IEEE Trans.* **C-49** (2000), 1290–1292.]

9. There is no solution if $s = 1$. Otherwise set $k \leftarrow s-1$, $j \leftarrow 1$, and do the following steps repeatedly: (i) While $j < v_k$, set $x_j \leftarrow 1$ and $j \leftarrow j+1$; (ii) stop if $k = 0$; (iii) if $h_k \ne 1$, set $x_j \leftarrow 1$ and $k \leftarrow h_k$, otherwise set $x_j \leftarrow 0$ and $k \leftarrow l_k$; (iv) set $j \leftarrow j+1$.

nim-like
games
von Neumann
Morgenstern
Berge
Lucas numbers
Fibonacci numbers
Perrin numbers
recurrences
Perrin

10. Let $I_k = (\bar{v}_k?\, l_k\!: h_k)$ for $0 \le k < s$ and $I'_k = (\bar{v}'_k?\, l'_k\!: h'_k)$ for $0 \le k < s'$. We may assume that $s = s'$; otherwise $f \ne f'$. The following algorithm either finds indices $(t_0, \ldots, t_{s-1})$ such that I_k corresponds to I'_{t_k}, or concludes that $f \ne f'$:

I1. [Initialize and loop.] Set $t_{s-1} \leftarrow s-1$, $t_1 \leftarrow 1$, $t_0 \leftarrow 0$, and $t_k \leftarrow -1$ for $2 \le k \le s-2$. Do steps I2–I4 for $k = s-1, s-2, \ldots, 2$ (in this order). If those steps "quit" at any point, we have $f \ne f'$; otherwise $f = f'$.

I2. [Test v_k.] Set $t \leftarrow t_k$. (Now $t \ge 0$; otherwise I_k would have no predecessor.) Quit if $v'_t \ne v_k$.

I3. [Test l_k.] Set $l \leftarrow l_k$. If $t_l < 0$, set $t_l \leftarrow l'_t$; otherwise quit if $l'_t \ne t_l$.

I4. [Test h_k.] Set $h \leftarrow h_k$. If $t_h < 0$, set $t_h \leftarrow h'_t$; otherwise quit if $h'_t \ne t_h$. ▮

11. (a) Yes, since c_k correctly counts all paths from node k to node 1. (In fact, many BDD algorithms will run correctly — but more slowly — in the presence of equivalent nodes or redundant branches. But reduction is important when, say, we want to test quickly if $f = f'$ as in exercise 10.)

(b) No. For example, suppose $I_3 = (\bar{1}?\,2\!:1)$, $I_2 = (\bar{1}?\,0\!:1)$, $I_1 = (\bar{2}?\,1\!:1)$, $I_0 = (\bar{2}?\,0\!:0)$; then the algorithm sets $c_2 \leftarrow 1$, $c_3 \leftarrow \frac{3}{2}$. (But see exercise 35(b).)

12. (a) The first condition makes K independent; the second makes it maximally so.

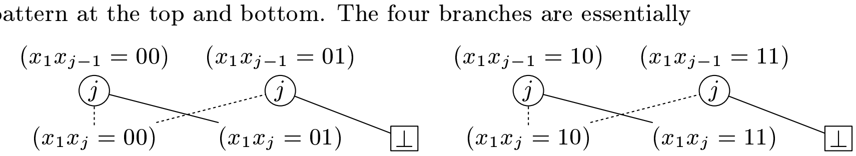
(b) None when n is odd; otherwise there are two sets of alternate vertices.

(c) A vertex is in the kernel if and only if it is a sink vertex or in the kernel of the graph obtained by deleting all sink vertices and their immediate predecessors.

[Kernels represent winning positions in nim-like games, and they also arise in n-person games. See J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior* (1944), §30.1; C. Berge, *Graphs and Hypergraphs* (1973), Chapter 14.]

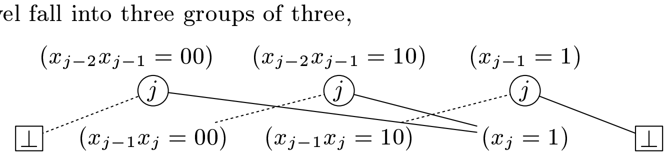
13. (a) A maximal clique of G is a kernel of $\overline{G}$, and vice versa. (b) A minimal vertex cover U is the complement $V \setminus W$ of a kernel W, and vice versa (see 7–(61)).

14. (a) The size is $4(n-2) + 2[n=3]$. When $n \ge 6$ these BDDs form a pattern in which there are four branch nodes for variables $4, 5, \ldots, n-2$, together with a fixed pattern at the top and bottom. The four branches are essentially



.

(b) Here the numbers for $3 \le n \le 10$ are $(7, 9, 14, 17, 22, 30, 37, 45)$; then a fixed pattern at the top and bottom develops as in (a), with nine branch nodes for each variable in the middle, and the total size comes to $9(n-5)$. The nine nodes on each middle level fall into three groups of three,



,

with one group for $x_1x_2 = 00$, one for $x_1x_2 = 01$, and one for $x_1 = 1$.

15. Both cases lead by induction to well known sequences of numbers: (a) The Lucas numbers $L_n = F_{n+1} + F_{n-1}$ [see E. Lucas, *Théorie des Nombres* (1891), Chapter 18]. (b) The Perrin numbers, defined by $P_3 = 3$, $P_4 = 2$, $P_5 = 5$, $P_n = P_{n-2} + P_{n-3}$. [See R. Perrin, L'*Intermédiaire des Mathématiciens* **6** (1899), 76–77.]

16. When the BDD isn't $\boxed{\bot}$, all solutions are generated by calling $List(1,\text{root})$, where $List(j,p)$ is the following recursive procedure: If $v(p)>j$, set $x_j\leftarrow 0$, call $List(j+1,p)$, set $x_j\leftarrow 1$, and call $List(j+1,p)$. Otherwise if p is the sink node $\boxed{\top}$, visit the solution $x_1\ldots x_n$. (The idea of "visiting" a combinatorial object while generating them all is discussed at the beginning of Section 7.2.1.) Otherwise set $x_j\leftarrow 0$; call $List(j+1,\text{LO}(p))$ if $\text{LO}(p)\neq\boxed{\bot}$; set $x_j\leftarrow 1$; and call $List(j+1,\text{HI}(p))$ if $\text{HI}(p)\neq\boxed{\bot}$.

recursive
visiting
lexicographic order
Hadamard matrix
Dahlheimer
Thue
Morse
Hamming distance

The solutions are generated in lexicographic order. Suppose there are N of them. If the kth solution agrees with the $(k-1)$st solution in positions $x_1\ldots x_{j-1}$ but not in x_j, let $c(k)=n-j$; and let $c(1)=n$. Then the running time is proportional to $\sum_{k=1}^{N}c(k)$, which is $O(nN)$ in general. (This bound holds because every branch node of a BDD leads to at least one solution. In fact, the running time is usually $O(N)$ in practice.)

17. That mission is impossible, because there's a function with $N=2^{2k}$ and $B(f)=O(2^{2k})$ for which every two solutions differ in more than 2^{k-1} bit positions. The running time for any algorithm that generates all solutions for such a function must be $\Omega(2^{3k})$, because $\Omega(2^k)$ operations are needed between solutions. To construct f, first let

$$g(x_1,\ldots,x_k,y_0,\ldots,y_{2^k-1})=\bigl[y_{(t_1\ldots t_k)_2}=x_1t_1\oplus\cdots\oplus x_kt_k \text{ for } 0\le t_1,\ldots,t_k\le 1\bigr].$$

(In other words, g asserts that $y_0\ldots y_{2^k-1}$ is row $(x_1\ldots x_k)_2$ of an Hadamard matrix; see Eq. 4.6.4–(38).) Now we let $f(x_1,\ldots,x_k,y_0,\ldots,y_{2^k-1},x'_1,\ldots,x'_k,y'_0,\ldots,y'_{2^k-1})=g(x_1,\ldots,x_k,y_0,\ldots,y_{2^k-1})\wedge g(x'_1,\ldots,x'_k,y'_0,\ldots,y'_{2^k-1})$. Clearly $B(f)=O(2^{2k})$ when the variables are ordered in this way. Indeed, T. Dahlheimer observes that $B(f)=2B(g)-2$, where $B(g)=2^k+1+\sum_{j=1}^{2^k}2^{\min(k,1+\lceil\lg j\rceil)}=\frac{5}{3}2^{2k-1}+2^k+\frac{5}{3}$.

18. First, $(W_1,\ldots,W_5)=(5,4,4,4,0)$. Then $m_2=w_4=4$ and $t_2=1$; $m_3=t_3=0$; $m_4=\max(m_3,m_2+w_3)=1$, $t_4=1$; $m_5=W_4-W_5=4$, $t_5=0$; $m_6=w_2+W_3-W_5=2$, $t_6=1$; $m_7=\max(m_5,m_4+w_2)=4$, $t_7=0$; $m_8=\max(m_7,m_6+w_1)=4$, $t_8=0$. Solution $x_1x_2x_3x_4=0001$.

19. $\sum_{j=1}^{n}\min(w_j,0)\le\sum_{j=v_k}^{n}\min(w_j,0)\le m_k\le\sum_{j=v_k}^{n}\max(w_j,0)=W_{v_k}\le W_1$.

20. Set $w_1\leftarrow -1$, then $w_{2j}\leftarrow w_j$ and $w_{2j+1}\leftarrow -w_j$ for $1\le j\le n/2$. [This method may also compute w_{n+1}. The sequence is named for works of A. Thue, *Skrifter udgivne af Videnskabs-Selskabet i Christiania*, Mathematisk-Naturvidenskabelig Klasse (1912), No. 1, §7, and H. M. Morse, *Trans. Amer. Math. Soc.* **22** (1921), 84–100, §14.]

21. Yes; we just have to change the sign of each weight w_j. (Or we could reverse the roles of LO and HI at each vertex.)

22. If $f(x)=f(x')=1$ when f represents a graph kernel, the Hamming distance $\nu(x\oplus x')$ cannot be 1. In such cases $v_l=v+1$ when $l\neq 0$ and $v_h=v+1$ when $h\neq 0$.

23. The BDD for the connectedness function of any connected graph will have exactly $n-1$ solid arcs on every root-to-$\boxed{\top}$ path, because that many edges are needed to connect n vertices, and because a BDD has no redundant branches. (See also Theorem S.)

24. Apply Algorithm B with weights $(w'_{12},\ldots,w'_{89})=(-w_{12}-x,\ldots,-w_{89}-x)$, where x is large enough to make all of these new weights w'_{uv} negative. The maximum of $\sum w'_{uv}x_{uv}$ will then occur with $\sum x_{uv}=8$, and those edges will form a spanning tree with minimum $\sum w_{uv}x_{uv}$. (We've seen a better algorithm for minimum spanning trees in exercise 2.3.4.1–11, and other methods will be studied in Section 7.5.4. However, this exercise indicates that a BDD can compactly represent the set of *all* spanning trees.)

25. The answer in step C1 becomes $(1+z)^{v_{s-1}-1}c_{s-1}$; the value of c_k in step C2 becomes $(1+z)^{v_l-v_k-1}c_l+(1+z)^{v_h-v_k-1}zc_h$.

Blum Chandra Wegman Coudert Madre Coudert

26. In this case the answer in step C1 is simply c_{s-1}; and the value of c_k in step C2 is simply $(1-p_{v_k})c_l + p_{v_k}c_h$.

27. The multilinear polynomial $H(x_1,\ldots,x_n) = F(x_1,\ldots,x_n) - G(x_1,\ldots,x_n)$ is nonzero modulo q, because it is ± 1 for some choice of integers with each $x_k \in \{0,1\}$. If it has degree d (modulo q), we can prove that there are at least $(q-1)^d q^{n-d}$ sets of values $(q_1,\ldots,q_n)$ with $0 \le q_k < q$ such that $H(q_1,\ldots,q_n) \bmod q \ne 0$. This statement is clear when $d=0$. And if x_k is a variable that appears in a term of degree $d>0$, the coefficient of x_k is a polynomial of degree $d-1$, which by induction on d is nonzero for at least $(q-1)^{d-1}q^{n-d}$ choices of $(q_1,\ldots,q_{k-1},q_{k+1},\ldots,q_n)$; for each of those choices there are $q-1$ values of q_k such that $H(q_1,\ldots,q_n) \bmod q \ne 0$.

Hence the stated probability is $\ge (1-1/q)^d \ge (1-1/q)^n$. [See M. Blum, A. K. Chandra, and M. N. Wegman, *Information Processing Letters* **10** (1980), 80–82.]

28. $F(p) = (1-p)^n G(p/(1-p))$. Similarly, $G(z) = (1+z)^n F(z/(1+z))$.

29. In step C1, also set $c'_0 \leftarrow 0$, $c'_1 \leftarrow 0$; return c_{s-1} and c'_{s-1}. In step C2, set $c_k \leftarrow (1-p)c_l + pc_h$ and $c'_k \leftarrow (1-p)c'_l - c_l + pc'_h + c_h$.

30. The following analog of Algorithm B does the job (assuming exact arithmetic):

A1. [Initialize.] Set $P_{n+1} \leftarrow 1$ and $P_j \leftarrow P_{j+1}\max(1-p_j,p_j)$ for $n \ge j \ge 1$.

A2. [Loop on k.] Set $m_1 \leftarrow 1$ and do step A3 for $2 \le k < s$. Then do step A4.

A3. [Process I_k.] Set $v \leftarrow v_k$, $l \leftarrow l_k$, $h \leftarrow h_k$, $t_k \leftarrow 0$. If $l \ne 0$, set $m_k \leftarrow m_l(1-p_v)P_{v+1}/P_{v_l}$. Then if $h \ne 0$, compute $m \leftarrow m_h p_v P_{v+1}/P_{v_h}$; and if $l=0$ or $m > m_k$, set $m_k \leftarrow m$ and $t_k \leftarrow 1$.

A4. [Compute the x's.] Set $j \leftarrow 0$, $k \leftarrow s-1$, and do the following operations until $j=n$: While $j < v_k - 1$, set $j \leftarrow j+1$ and $x_j \leftarrow [p_j > \frac{1}{2}]$; if $k>1$, set $j \leftarrow j+1$ and $x_j \leftarrow t_k$ and $k \leftarrow (t_k{=}0?\ l_k\colon h_k)$. ▌

31. **C1′.** [Loop over k.] Set $\alpha_0 \leftarrow \bot$, $\alpha_1 \leftarrow \top$, and do step C2′ for $k = 2, 3, \ldots, s-1$. Then go to C3′.

C2′. [Compute α_k.] Set $v \leftarrow v_k$, $l \leftarrow l_k$, and $h \leftarrow h_k$. Set $\beta \leftarrow \alpha_l$ and $j \leftarrow v_l - 1$; then while $j > v$ set $\beta \leftarrow (\bar{x}_j \circ x_j) \bullet \beta$ and $j \leftarrow j-1$. Set $\gamma \leftarrow \alpha_h$ and $j \leftarrow v_h - 1$; then while $j > v$ set $\gamma \leftarrow (\bar{x}_j \circ x_j) \bullet \gamma$ and $j \leftarrow j-1$. Finally set $\alpha_k \leftarrow (\bar{x}_v \bullet \beta) \circ (x_v \bullet \gamma)$.

C3′. [Finish.] Set $\alpha \leftarrow \alpha_{s-1}$ and $j \leftarrow v_{s-1}-1$; then while $j>0$ set $\alpha \leftarrow (\bar{x}_j \circ x_j) \bullet \alpha$ and $j \leftarrow j-1$. Return the answer α. ▌

This algorithm performs $\circ$ and $\bullet$ operations at most $O(nB(f))$ times. The upper bound can often be lowered to $O(n) + O(B(f))$; but shortcuts like the calculation of W_k in step B1 aren't always available. [See O. Coudert and J. C. Madre, *Proc. Reliability and Maint. Conf.* (IEEE, 1993), 240–245, §4; O. Coudert, *Integration* **17** (1994), 126–127.]

32. For exercise 25, '$\circ$' is addition, '$\bullet$' is multiplication, '$\bot$' is 0, '$\top$' is 1, '$\bar{x}_j$' is 1, 'x_j' is z. Exercise 26 is similar, but '$\bar{x}_j$' is $1-p_j$ and 'x_j' is p_j.

In exercise 29 the objects of the algebra are pairs (c,c'), and we have $(a,a') \circ (b,b') = (a+b, a'+b')$, $(a,a') \bullet (b,b') = (ab, ab'+a'b)$. Also '$\bot$' is $(0,0)$, '$\top$' is $(1,0)$, '$\bar{x}_j$' is $(1{-}p,-1)$, and 'x_j' is $(p,1)$.

In exercise 30, '$\circ$' is max, '$\bullet$' is multiplication, '$\bot$' is $-\infty$, '$\top$' is 1, '$\bar{x}_j$' is $1-p_j$, 'x_j' is p_j. Multiplication distributes over max in this case because the quantities are either nonnegative or $-\infty$; we must define $0 \bullet (-\infty) = -\infty$ in order to satisfy (22).

(Additional possibilities abound, because associative and distributive operators are ubiquitous in mathematics. The algebraic objects need not be numbers or polynomials or pairs; they can be strings, matrices, functions, sets of numbers, sets of strings, sets or multisets of matrices of pairs of functions of strings, etc., etc. We will see many further examples in Section 7.3. The min-plus algebra, with $\circ = \min$ and $\bullet = +$, is particularly important, and we could have used it in exercise 21 or 24. It is often called *tropical*, implicitly honoring the Brazilian mathematician Imre Simon.)

min-plus algebra
tropical
Simon
Jeong
Somenzi

33. Operate on triples (c, c', c''), with $(a, a', a'') \circ (b, b', b'') = (a + b, a' + b', a'' + b'')$ and $(a, a', a'') \bullet (b, b', b'') = (ab, a'b + b'a, a''b + 2a'b' + ab'')$. Interpret '$\bot$' as $(0, 0, 0)$, '$\top$' as $(1, 0, 0)$, '$\bar{x}_j$' as $(1, 0, 0)$, and 'x_j' as $(1, w_j, w_j^2)$.

34. Let $x \vee y = \max(x, y)$. Operate on pairs (c, c'), with $(a, a') \circ (b, b') = (a \vee b, a' \vee b')$ and $(a, a') \bullet (b, b') = (a + b, (a' + b) \vee (a + b'))$. Interpret '$\bot$' as $(-\infty, -\infty)$, '$\top$' as $(0, -\infty)$, '$\bar{x}_j$' as $(0, w_j'')$, and 'x_j' as $(w_j, w_j' + w_j'')$. The first component of the result will agree with Algorithm B; the second component is the desired maximum.

35. (a) The supposed FBDD can be represented by instructions $I_{s-1}, \ldots, I_0$ as in Algorithm C. Start with $R_0 \leftarrow R_1 \leftarrow \emptyset$, then do the following for $k = 2, \ldots, s - 1$: Report failure if $v_k \in R_{l_k} \cup R_{h_k}$; otherwise set $R_k \leftarrow \{v_k\} \cup R_{l_k} \cup R_{h_k}$. (The set R_k identifies all variables that are reachable from I_k.)

(b) The reliability polynomial can be calculated just as in answer 26. To count solutions, we essentially set $p_1 = \cdots = p_n = \frac{1}{2}$ and multiply by 2^n: Start with $c_0 \leftarrow 0$ and $c_1 \leftarrow 2^n$, then set $c_k \leftarrow (c_{l_k} + c_{h_k})/2$ for $1 < k < s$. The answer is c_{s-1}.

36. Compute the sets R_k as in answer 35(a). Instead of looping on j as stated in step C2$'$ of answer 31, set $\beta \leftarrow \alpha_l$ and then $\beta \leftarrow (\bar{x}_j \circ x_j) \bullet \beta$ for all $j \in R_k \setminus R_l \setminus \{v\}$; treat γ in the same manner. Similarly, in step C3$'$ set $\alpha \leftarrow (\bar{x}_j \circ x_j) \bullet \alpha$ for all $j \notin R_{s-1}$.

37. Given any FBDD for f, the function $G(z)$ is the sum of $(1+z)^{n-\text{length}\,P} z^{\text{solid arcs in}\,P}$ over all paths P from the root to $\boxed{\top}$. [See *Theoretical Comp. Sci.* **3** (1976), 371–384.]

38. The key fact is that $x_j = 1$ forces $f = 1$ if and only if we have (i) $h_k = 1$ whenever $v_k = j$; (ii) $v_k = j$ in at least one step k; (iii) there are no steps with ($v_k < j < v_{l_k}$ and $l_k \neq 1$) or ($v_k < j < v_{h_k}$ and $h_k \neq 1$).

K1. [Initialize.] Set $t_j \leftarrow 2$ and $p_j \leftarrow 0$ for $1 \le j \le n$.

K2. [Examine all branches.] Do the following operations for $2 \le k < s$: Set $j \leftarrow v_k$ and $q \leftarrow 0$. If $l_k = 1$, set $q \leftarrow -1$; otherwise set $p_j \leftarrow \max(p_j, v_{l_k})$. If $h_k = 1$, set $q \leftarrow +1$; otherwise set $p_j \leftarrow \max(p_j, v_{h_k})$. If $t_j = 2$, set $t_j \leftarrow q$; otherwise if $t_j \neq q$ set $t_j \leftarrow 0$.

K3. [Finish up.] Set $m \leftarrow v_{s-1}$, and do the following for $j = 1, 2, \ldots, n$: If $j < m$, set $t_j \leftarrow 0$; then if $p_j > m$, set $m \leftarrow p_j$. ▌

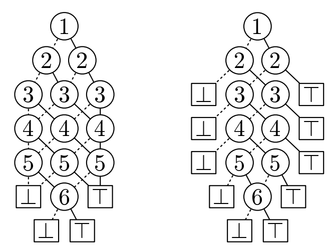
[See S.-W. Jeong and F. Somenzi, in *Logic Synthesis and Optimization* (1993), 154–156.]

39. $k(n + 1 - k) + 2$, for $1 \le k \le n$. (See (26).)

40. (a) Suppose the BDDs for f and g have respectively a_j and b_j branch nodes (j), for $1 \le j \le n$. Each subtable of f of order $n + 1 - k$ has the form $\alpha\beta\gamma\delta$, where α, β, γ, and δ are subtables of order $n - 1 - k$. The corresponding subtables of g are $\alpha\alpha\delta\delta$; hence they are beads if and only if $\alpha \neq \delta$, in which case either $\alpha\beta\gamma\delta$ is a bead or $\alpha\beta = \gamma\delta$ is a bead. Consequently $b_k \le a_k + a_{k+1}$, and $b_{k+1} = 0$. We also have $b_j \le a_j$ for $1 \le j < k$, because every bead of g of order $> n+1-k$ is "condensed" from at least one such bead of f. And $b_j \le a_j$ for $j > k+1$, because the subtables on $(x_{k+2}, \ldots, x_n)$ are identical although they might not appear in g.

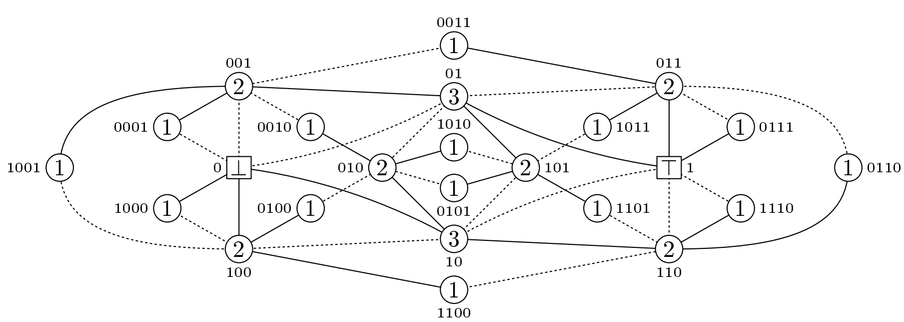
(b) Not always. The simplest counterexample is $f(x_1, x_2, x_3, x_4) = x_2 \wedge (x_3 \vee x_4)$, $h(x_1, x_2, x_1, x_4) = x_2 \wedge (x_1 \vee x_4)$, when $B(f) = 5$ and $B(h) = 6$. (We do, however, always have $B(h) < 2B(f)$.)

41. (a) $3n - 3$; (b) $2n$. (The general patterns are illustrated here for $n = 6$. One can also show that the "organ-pipe ordering" $\langle x_n^{F_1} x_1^{F_2} x_{n-1}^{F_3} x_2^{F_4} \ldots x_{\lfloor n/2 \rfloor + [n\,\mathrm{even}]}^{F_{n-1}} x_{\lceil n/2 \rceil}^{F_{n-2}} \rangle$ produces the profile 1, 2, 4, ..., $2\lceil n/2 \rceil - 2$, $2\lfloor n/2 \rfloor - 1$, ..., 5, 3, 1, 2, giving the total BDD size $\binom{n}{2} + 3$; this ordering appears to be the worst for the Fibonacci weights.)



The functions $[F_n x_1 + \cdots + F_1 x_n \ge t]$ have been studied by J. T. Butler and T. Sasao, *Fibonacci Quart.* **34** (1996), 413–422.

42. (Compare with exercise 2.) The sixteen roots are the (1) nodes and the two sinks:



43. (a) Since $f(x_1, \ldots, x_{2n})$ is the symmetric function $S_n(x_1, \ldots, x_n, \bar{x}_{n+1}, \ldots, \bar{x}_{2n})$, we have $B(f) = 1 + 2 + \cdots + (n{+}1) + \cdots + 3 + 2 + 2 = n^2 + 2n + 2$.

(b) By symmetry, the size is the same for $[\sum\{x_i \mid i \in I\} = \sum\{x_i \mid i \notin I\}]$, $|I| = n$.

44. There are at most $\min(k, 2^{n+2-k} - 2)$ nodes labeled (k), for $1 \le k \le n$, because there are $2^{n+2-k} - 2$ symmetric functions of $(x_k, \ldots, x_n)$ that aren't constant. Thus S_n is at most $2 + \sum_{k=1}^{n} \min(k, 2^{n+2-k} - 2)$, which can be expressed in closed form as $(n{+}2{-}b_n)(n{+}1{-}b_n)/2 + 2(2^{b_n} - b_n)$, where $b_n = \lambda(n{+}4 - \lambda(n{+}4))$ and $\lambda n = \lfloor \lg n \rfloor$.

A symmetric function that attains this worst-case bound can be constructed in the following way (related to the de Bruijn cycles constructed in exercise 3.2.2–7): Let $p(x) = x^d + a_1 x^{d-1} + \cdots + a_d$ be a primitive polynomial modulo 2. Set $t_k \leftarrow 1$ for $0 \le k < d$; $t_k \leftarrow (a_1 t_{k-1} + \cdots + a_d t_{k-d}) \bmod 2$ for $d \le k < 2^d + d - 2$; $t_k \leftarrow (1 + a_1 t_{k-1} + \cdots + a_d t_{k-d}) \bmod 2$ for $2^d + d - 2 \le k < 2^{d+1} + d - 3$; and $t_{2^{d+1}+d-3} \leftarrow 1$. For example, when $p(x) = x^3 + x + 1$ we get $t_0 \ldots t_{16} = 11100101101000111$.

Then (i) the sequence $t_1 \ldots t_{2^d+d-3}$ contains all d-tuples except 0^d and 1^d as substrings; (ii) the sequence $t_{2^d+d-2} \ldots t_{2^{d+1}+d-4}$ is a cyclic shift of $\bar{t}_0 \ldots \bar{t}_{2^d-2}$; and (iii) $t_k = 1$ for $2^d - 1 \le k \le 2^d + d - 3$ and $2^{d+1} - 2 \le k \le 2^{d+1} + d - 3$. Consequently the sequence $t_0 \ldots t_{2^{d+1}+d-3}$ contains all $(d{+}1)$-tuples except 0^{d+1} and 1^{d+1} as substrings. Set $f(x) = t_{\nu x}$ to maximize $B(f)$ when $2^d + d - 4 < n \le 2^{d+1} + d - 3$.

Asymptotically, $S_n = \frac{1}{2} n^2 - n \lg n + O(n)$. [See I. Wegener, *Information and Control* **62** (1984), 129–143; M. Heap, *J. Electronic Testing* **4** (1993), 191–195.]

45. Module M_1 has only three inputs (x_1, y_1, z_1), and only three outputs $u_2 = x_1$, $v_2 = y_1 x_1$, $w_2 = z_1 x_1$. Module M_{n-1} is almost normal, but it has no input port for z_{n-1},

organ-pipe ordering
reordering
Butler
Sasao
symmetric function
λx
binary logarithm
de Bruijn cycles
primitive polynomial modulo 2
Wegener
Heap

and it doesn't output u_n; it sets $z_{n-2} = x_{n-1}y_{n-1}$. Module M_n has only three inputs (v_n, w_n, x_n), and one output $y_{n-1} = x_n$ together with the main output, $w_n \vee v_n x_n$. With these definitions the dependencies between ports form an acyclic digraph.

(Modules could be constructed with all $b_k = 0$ and $a_k \le 5$, or even with $a_k \le 4$ as we'll see in exercise 47. But (33) and (34) are intended to illustrate backward signals in a simple example, not to demonstrate the tightest possible construction.)

46. For $6 \le k \le n-3$ there are nine branches on (k), corresponding to three cases $(\bar{x}_1, x_1\bar{x}_2, x_1x_2)$ times three cases $(\bar{x}_{k-1}, \bar{x}_{k-2}x_{k-1}, \bar{x}_{k-3}x_{k-2}x_{k-1})$. The total BDD size turns out to be exactly $9n-38$, if $n \ge 6$.

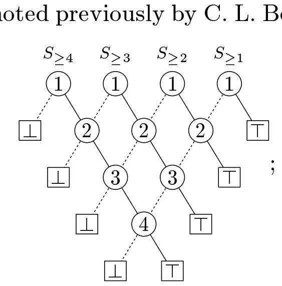
47. Suppose f has q_k subtables of order $n-k$, so that its QDD has q_k nodes that branch on x_{k+1}. We can encode them in $a_k = \lceil \lg q_k \rceil$ bits, and construct a module M_{k+1} with $b_k = b_{k+1} = 0$ that mimics the behavior of those q_k branch nodes. Thus by (86),

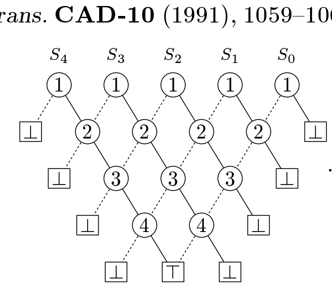
$$\sum_{k=0}^{n} 2^{a_k 2^{b_k}} = \sum_{k=0}^{n} 2^{\lceil \lg q_k \rceil} \;\le\; \sum_{k=0}^{n} (2q_k - 1) \;=\; 2Q(f) - (n+1) \;\le\; (n+1)B(f).$$

(The 2^m-way multiplexer shows that the additional factor of $(n+1)$ is necessary; indeed, Theorem M actually gives an upper bound on $Q(f)$.)

48. The sums $u_k = x_1 + \cdots + x_k$ and $v_k = x_{k+1} + \cdots + x_n$ can be represented on $1 + \lambda k$ and $1 + \lambda(n-k)$ wires, respectively. Let $t_k = x_k \wedge [u_k + v_k = k]$ and $w_k = t_1 \vee \cdots \vee t_k$. We can construct modules M_k having inputs u_{k-1} and w_{k-1} from M_{k-1} together with inputs v_k from M_{k+1}; module M_k outputs $u_k = u_{k-1} + x_k$ and $w_k = w_{k-1} \vee t_k$ to M_{k+1} as well as $v_{k-1} = v_k + x_k$ to M_{k-1}.

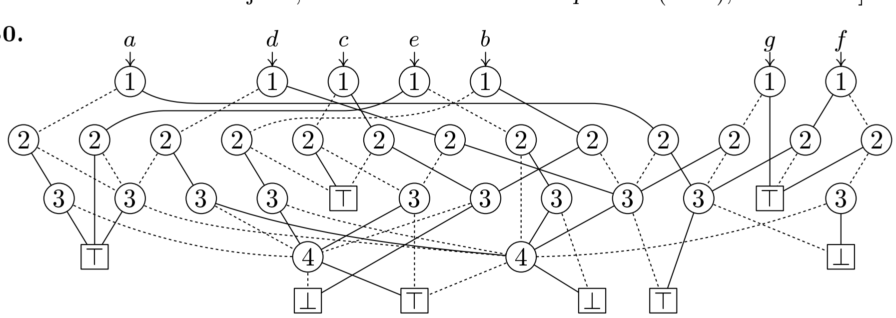
If p is a polynomial, $\sum_{k=0}^{n} 2^{p(a_k, b_k)} = 2^{(\log n)^{O(1)}}$ is asymptotically less than $2^{\Omega(n)}$. [See K. L. McMillan, *Symbolic Model Checking* (1993), §3.5, where Theorem M was introduced, with extensions to nonlinear layouts. The special case $b_1 = \cdots = b_n = 0$ had been noted previously by C. L. Berman, *IEEE Trans.* **CAD-10** (1991), 1059–1066.]

49. (a) 

; (b) 

.

[See I. Semba and S. Yajima, *Trans. Inf. Proc. Soc. Japan* **35** (1994), 1663–1665.]

50. 

subtables
QDD
2^m-way multiplexer
$Q(f)$
McMillan
Berman
Semba
Yajima

51. In this case $B(f_j) = 3j+2$ for $1 \le j \le n$, and $B(f_{n+1}) = 3n+1$; so the individual BDDs are only about 1/3 as big as they are within (36). But almost no nodes are shared — only the sinks and one branch. So the total BDD size comes to $(3n^2+9n)/2$.

52. If the BDD base for $\{f_1, \ldots, f_m\}$ has s nodes, then $B(f) = s+m+1+[s=1]$.

53. Call the branch nodes a, b, c, d, e, f, g, with `ROOT` $= a$. After step R1 we have `HEAD[1]` $= \sim a$, `AUX`$(a) = \sim 0$; `HEAD[2]` $= \sim b$, `AUX`$(b) = \sim c$, `AUX`$(c) = \sim 0$; `HEAD[3]` $= \sim d$, `AUX`$(d) = \sim e$, `AUX`$(e) = \sim f$, `AUX`$(f) = \sim g$, `AUX`$(g) = \sim 0$.

After R3 with $v = 3$ we have $s = \sim 0$, `AUX(0)` $= \sim e$, `AUX`$(e) = f$, `AUX`$(f) = 0$; also `AVAIL` $= g$, `LO`$(g) = \sim 1$, `HI`$(g) = d$, `LO`$(d) = \sim 0$, and `HI`$(d) = \alpha$, where α was the initial value of `AVAIL`. (Nodes g and d have been recycled in favor of 1 and 0.) Then R4 sets $s \leftarrow e$ and `AUX(0)` $\leftarrow 0$. (The remaining nodes with `V` $= v$ start at s, linked via `AUX`.)

Now R7, starting with $p = q = e$ and $s = 0$, sets `AUX(1)` $\leftarrow \sim e$, `LO`$(f) \leftarrow \sim e$, `HI`$(f) \leftarrow g$, `AVAIL` $\leftarrow f$; and R8 resets `AUX(1)` $\leftarrow 0$.

Then step R3 with $v = 2$ sets `LO`$(b) \leftarrow 0$, `LO`$(c) \leftarrow e$, and `HI`$(c) \leftarrow 1$. No further changes of importance take place, although some `AUX` fields temporarily become negative. We end up with Fig. 21.

54. Create nodes j for $1 < j \le 2^{n-1}$ by setting `V`$(j) \leftarrow \lceil \lg j \rceil$, `LO`$(j) \leftarrow 2j-1$, and `HI`$(j) \leftarrow 2j$; also for $2^{n-1} < j \le 2^n$ by setting `V`$(j) \leftarrow n$, `LO`$(j) \leftarrow f(x_1, \ldots, x_{n-1}, 0)$, and `HI`$(j) \leftarrow f(x_1, \ldots, x_{n-1}, 1)$ when $j = (1x_1 \ldots x_{n-1})_2 + 1$. Then apply Algorithm R with `ROOT` $= 2$. (We can bypass step R1 by first setting `AUX`$(j) \leftarrow -j$ for $4 \le j \le 2^n$, then `HEAD`$[k] \leftarrow \sim(2^k)$ and `AUX`$(2^{k-1}+1) \leftarrow -1$ for $1 \le k \le n$.)

55. It suffices to construct an unreduced diagram, since Algorithm R will then finish the job. Number the vertices $1, \ldots, n$ in such a way that no vertex except 1 appears before all of its neighbors. Represent the edges by arcs $a_1, \ldots, a_e$, where a_k is $u_k \longrightarrow v_k$ for some $u_k < v_k$, and where the arcs having $u_k = j$ are consecutive, with $s_j \le k < s_{j+1}$ and $1 = s_1 \le \cdots \le s_n = s_{n+1} = e+1$. Define the "frontier" $V_k = \{1, v_1, \ldots, v_k\} \cap \{u_k, \ldots, n\}$ for $1 \le k \le e$, and let $V_0 = \{1\}$. The unreduced decision diagram will have branches on arc a_k for all partitions of V_{k-1} that correspond to connectedness relations that have arisen because of previous branches.

For example, consider $P_3 \mathbin{\square} P_3$, where $(s_1, \ldots, s_{10}) = (1,3,5,7,8,10,11,12,13,13)$ and $V_0 = \{1\}$, $V_1 = \{1,2\}$, $V_2 = \{1,2,3\}$, $V_3 = \{2,3,4\}$, $\ldots$, $V_{12} = \{8,9\}$. The branch on a_1 goes from the trivial partition 1 of V_0 to the partition $1|2$ of V_1 if $1 \not\!\!-\!\!- 2$, or to the partition 12 if $1 \mathrel{-\!\!-} 2$. (The notation '$1|2$' stands for the set partition $\{1\} \cup \{2\}$, as in Section 7.2.1.5.) From $1|2$, the branch on a_2 goes to the partition $1|2|3$ of V_2 if $1 \not\!\!-\!\!- 3$, otherwise to $13|2$; from 12, the branches go respectively to partitions $12|3$ and 123. Then from $1|2|3$, both branches on a_3 go to $\boxed{\bot}$, because vertex 1 can no longer be connected to the others. And so on. Eventually the partitions of $V_e = V_{12}$ are all identified with $\boxed{\bot}$, except for the trivial one-set partition, which corresponds to $\boxed{\top}$.

56. Start with $m \leftarrow 2$ in step R1, and $v_0 \leftarrow v_1 \leftarrow v_{\max}+1$, $l_0 \leftarrow h_0 \leftarrow 0$, $l_1 \leftarrow h_1 \leftarrow 1$ as in (8). Assume that `HI(0)` $= 0$ and `HI(1)` $= 1$. Omit the assignments that involve `AVAIL` in steps R3 and R7. After setting `AUX(HI`(p)`)` $\leftarrow 0$ in step R8, also set $v_m \leftarrow v$, $l_m \leftarrow$ `HI(LO`(p)`)`, $h_m \leftarrow$ `HI(HI`(p)`)`, `HI`$(p) \leftarrow m$, and $m \leftarrow m+1$. At the end of step R9, set $s \leftarrow m - [$`ROOT`$=0]$.

57. Set `LO(ROOT)` $\leftarrow \sim$`LO(ROOT)`. (We briefly complement the `LO` field of nodes that are still accessible after restriction.) Then for $v =$ `V(ROOT)`, $\ldots$, $v_{\max}$, set $p \leftarrow \sim$`HEAD`$[v]$, `HEAD`$[v] \leftarrow \sim 0$, and do the following while $p \ne 0$: (i) Set $p' \leftarrow \sim$`AUX`(p). (ii) If `LO`$(p) \ge 0$, set `HI`$(p) \leftarrow$ `AVAIL`, `AUX`$(p) \leftarrow 0$, and `AVAIL` $\leftarrow p$ (node p can no longer be reached). Otherwise set `LO`$(p) \leftarrow \sim$`LO`(p); if `FIX`$[v] = 0$, set `HI`$(p) \leftarrow$ `LO`(p); if `FIX`$[v] = 1$, set

common subfunctions
frontier
partitions of a set
set partition

$\mathtt{LO}(p) \leftarrow \mathtt{HI}(p)$; if $\mathtt{LO}(\mathtt{LO}(p)) \geq 0$, set $\mathtt{LO}(\mathtt{LO}(p)) \leftarrow {\sim}\mathtt{LO}(\mathtt{LO}(p))$; if $\mathtt{LO}(\mathtt{HI}(p)) \geq 0$, set $\mathtt{LO}(\mathtt{HI}(p)) \leftarrow {\sim}\mathtt{LO}(\mathtt{HI}(p))$; and set $\mathtt{AUX}(p) \leftarrow \mathtt{HEAD}[v]$, $\mathtt{HEAD}[v] \leftarrow {\sim}p$. (iii) Set $p \leftarrow p'$. Finally, after finishing the loop on v, restore $\mathtt{LO}(0) \leftarrow 0$, $\mathtt{LO}(1) \leftarrow 1$.

58. Since $l \neq h$ and $l' \neq h'$, we have $l \diamond l' \neq h \diamond h'$, $l \diamond \alpha' \neq h \diamond \alpha'$, and $\alpha \diamond l' \neq \alpha \diamond h'$.

Suppose $\alpha \diamond \alpha' = \beta \diamond \beta'$, where $\beta = (v'', l'', h'')$ and $\beta' = (v''', l''', h''')$. If $v'' = v'''$ we have $v = v''$, $l \diamond l' = l'' \diamond l'''$, and $h \diamond h' = h'' \diamond h'''$. If $v'' < v'''$ we have $v = v''$, $l \diamond \alpha' = l'' \diamond \beta'$, and $h \diamond \alpha' = h'' \diamond \beta'$. Otherwise we have $v' = v'''$, $\alpha \diamond l' = \beta \diamond l'''$, and $\alpha \diamond h' = \beta \diamond h'''$. By induction, therefore, we have $\alpha = \beta$ and $\alpha' = \beta'$ in all cases.

59. (a) If h isn't constant we have $B(f \diamond g) = 3B(h) - 2$, essentially obtained by taking a copy of the BDD for h and replacing its sink nodes by two other copies.

(b) Suppose the profile and quasi-profile of h are $(b_0, \ldots, b_n)$ and $(q_0, \ldots, q_n)$, where $b_n = q_n = 2$. Then there are $b_k q_k$ branches on x_{2k+1} in $f \diamond g$, and $q_k b_{k-1}$ branches on x_{2k}, corresponding to ordered pairs of beads and subtables of h. When the BDD for h contains a branch from α to β and from α' to β', where $\mathtt{V}(\alpha) = j$, $\mathtt{V}(\beta) = k$, $\mathtt{V}(\alpha') = j'$, and $\mathtt{V}(\beta') = k'$, the BDD for $f \diamond g$ contains a corresponding branch with $\mathtt{V}(\alpha \diamond \alpha') = 2j - 1$ from $\alpha \diamond \alpha'$ to $\beta \diamond \alpha'$ when $j \leq j' < k$, and with $\mathtt{V}(\alpha \diamond \alpha') = 2j'$ from $\alpha \diamond \alpha'$ to $\alpha \diamond \beta'$ when $j' < j \leq k'$.

60. Every bead of order $n - j$ of the ordered pair (f, g) is either one of the $b_j b'_j$ ordered pairs of beads of f and g, or one of the $b_j(q'_j - b'_j) + (q_j - b_j)b'_j$ ordered pairs that have the form (bead, nonbead) or (nonbead, bead). [This upper bound is achieved in the examples of exercises 59(b) and 63.]

61. Assume that $v = V(\alpha) \leq V(\beta)$. Let $\alpha_1, \ldots, \alpha_k$ be the nodes that point to α, and let $\beta_1, \ldots, \beta_l$ be the nodes with $V(\beta_j) < v$ that point to β; an imaginary node is assumed to point to each root. (Thus $k = \text{in-degree}(\alpha)$ and $l \leq \text{in-degree}(\beta)$.) Then the melded nodes that point to $\alpha \diamond \beta$ are of three types: (i) $\alpha_i \diamond \beta_j$, where $V(\alpha_i) = V(\beta_j)$ and either ($\mathrm{LO}(\alpha_i) = \alpha$ and $\mathrm{LO}(\beta_j) = \beta$) or ($\mathrm{HI}(\alpha_i) = \alpha$ and $\mathrm{HI}(\beta_j) = \beta$); (ii) $\alpha \diamond \beta_j$, where $V(\alpha_i) < V(\beta_j)$ for some i; or (iii) $\alpha_i \diamond \beta$, where $V(\alpha_i) > V(\beta_j)$ for some j.

62. The BDD for f has one node on each level, and the BDD for g has two, except at the top and bottom. The BDD for $f \vee g$ has four nodes on nearly every level, by exercise 14(a). The BDD for $f \diamond g$ has seven nodes (j) when $5 \leq j \leq n - 3$, corresponding to ordered pairs of subtables of (f, g) that depend on x_j when $(x_1, \ldots, x_{j-1})$ have fixed values. Thus $B(f) = n + O(1)$, $B(g) = 2n + O(1)$, $B(f \diamond g) = 7n + O(1)$, and $B(f \vee g) = 4n + O(1)$. (Also $B(f \wedge g) = 7n + O(1)$, $B(f \oplus g) = 7n + O(1)$.)

63. The profiles of f and g are respectively $(1, 2, 2, \ldots, 2^{m-1}, 2^{m-1}, 2^m, 1, 1, \ldots, 1, 2)$ and $(0, 1, 2, 2, \ldots, 2^{m-1}, 2^{m-1}, 1, 1, \ldots, 1, 2)$; so $B(f) = 2^{m+2} - 1 \approx 4n$ and $B(g) = 2^{m+1} + 2^m - 1 \approx 3n$. The profile of $f \wedge g$ begins with $(1, 2, 4, \ldots, 2^{2m-2}, 2^{2m-1} - 2^{m-1})$, because there's a unique solution $x_1 \ldots x_{2m}$ to the equations

$$((x_1 \oplus x_2)(x_3 \oplus x_4) \ldots (x_{2m-1} \oplus x_{2m}))_2 = p, ((x_2 \oplus x_3) \ldots (x_{2m-2} \oplus x_{2m-1})x_{2m})_2 = q$$

for $0 \leq p, q < 2^m$, and $p = q$ if and only if $x_1 = x_3 = \cdots = x_{2m-1} = 0$. After that the profile continues $(2^{m+1} - 2, 2^{m+1} - 2, 2^{m+1} - 4, 2^{m+1} - 6, \ldots, 4, 2, 2)$; the subfunctions are $x_{2m+j} \wedge \bar{x}_{2m+k}$ or $\bar{x}_{2m+j} \wedge x_{2m+k}$ for $1 \leq j < k \leq 2^m$, together with x_{2m+j} and $\bar{x}_{2m+j}$ for $2 \leq j \leq 2^m$. All in all, we have $B(f \wedge g) = 2^{2m+1} + 2^{m-1} - 1 \approx 2n^2$.

64. The BDD for *any* Boolean combination of f_1, f_2, and f_3 is contained in the meld $f_1 \diamond f_2 \diamond f_3$, whose size is at most $B(f_1)B(f_2)B(f_3)$.

65. $h = g?\ f_1\colon f_0$, where f_c is the restriction of f obtained by setting $x_j \leftarrow c$. The first upper bound follows as in answer 64, because $B(f_c) \leq B(f)$. The second bound

profile
quasi-profile
beads
subtables
bead
depend on
meld
mux
if-then-else function
restriction

fails when, for example, $n = 2^m + 3m$ and $h = M_m(x;y)? \ M_m(x';y) \colon M_m(x'';y)$, where $x = (x_1, \ldots, x_m)$, $x' = (x'_1, \ldots, x'_m)$, $x'' = (x''_1, \ldots, x''_m)$, and $y = (y_0, \ldots, y_{2^m-1})$; but such failures appear to be rare. [See R. E. Bryant, *IEEE Trans.* **C-35** (1986), 685; J. Jain, K. Mohanram, D. Moundanos, I. Wegener, and Y. Lu, *ACM/IEEE Design Automation Conf.* **37** (2000), 681–686.]

2^m-way multiplexer
notation $M_m(x;y)$
Bryant
Jain
Mohanram
Moundanos
Wegener
Lu

66. Set `NTOP` $\leftarrow f_0 + 1 - l$ and terminate the algorithm.

67. Let t_k denote template location `POOLSIZE` $- 2k$. Step S1 sets `LEFT(`t_1`)` $\leftarrow 5$, `RIGHT(`t_1`)` $\leftarrow 7$, $l \leftarrow 1$. Step S2 for $l = 1$ puts t_1 into both `LLIST[2]` and `HLIST[2]`. Step S5 for $l = 2$ sets `LEFT(`t_2`)` $\leftarrow 4$, `RIGHT(`t_2`)` $\leftarrow 5$, `L(`t_1`)` $\leftarrow t_2$; `LEFT(`t_3`)` $\leftarrow 3$, `RIGHT(`t_3`)` $\leftarrow 6$, `H(`t_1`)` $\leftarrow t_3$. Step S2 for $l = 2$ sets `L(`t_2`)` $\leftarrow 0$ and puts t_2 in `HLIST[3]`; then it puts t_3 into `LLIST[3]` and `HLIST[3]`. And so on. Phase 1 ends with (`LSTART[0]`$, \ldots,$ `LSTART[4]`) $= (t_0, t_1, t_3, t_5, t_8)$ and

k	`LEFT(`t_k`)`	`RIGHT(`t_k`)`	`L(`t_k`)`	`H(`t_k`)`	k	`LEFT(`t_k`)`	`RIGHT(`t_k`)`	`L(`t_k`)`	`H(`t_k`)`
1	5 $[\alpha]$	7 $[\omega]$	t_2	t_3	5	3 $[\gamma]$	4 $[\varphi]$	t_6	t_8
2	4 $[\beta]$	5 $[\chi]$	0	t_4	6	2 $[\delta]$	2 $[\tau]$	0	1
3	3 $[\gamma]$	6 $[\psi]$	t_4	t_5	7	2 $[\delta]$	1 $[\top]$	0	1
4	3 $[\gamma]$	1 $[\top]$	t_7	1	8	1 $[\top]$	3 $[\upsilon]$	1	0

representing the meld $\alpha \diamond \omega$ in Fig. 24 but with $\bot \diamond x = x \diamond \bot = \bot$ and $\top \diamond \top = \top$.

Let $f_k = f_0 + k$. In phase 2, step S7 for $l = 4$ sets `LEFT(`t_6`)` $\leftarrow {\sim}0$, `LEFT(`t_7`)` $\leftarrow t_6$, `LEFT(`t_8`)` $\leftarrow {\sim}1$, and `RIGHT(`t_6`)` $\leftarrow$ `RIGHT(`t_7`)` $\leftarrow$ `RIGHT(`t_8`)` $\leftarrow -1$. Step S8 undoes the changes made to `LEFT(0)` and `LEFT(1)`. Step S11 with $s = t_8$ sets `LEFT(`t_8`)` $\leftarrow {\sim}2$, `RIGHT(`t_8`)` $\leftarrow t_8$, `V(`f_2`)` $\leftarrow 4$, `LO(`f_2`)` $\leftarrow 1$, `HI(`f_2`)` $\leftarrow 0$. With $s = t_7$ that step sets `LEFT(`t_7`)` $\leftarrow {\sim}3$, `RIGHT(`t_7`)` $\leftarrow t_7$, `V(`f_3`)` $\leftarrow 4$, `LO(`f_3`)` $\leftarrow 0$, `HI(`f_3`)` $\leftarrow 1$; meanwhile step S10 has set `RIGHT(`t_6`)` $\leftarrow t_7$. Eventually the templates will be transformed to

k	`LEFT(`t_k`)`	`RIGHT(`t_k`)`	`L(`t_k`)`	`H(`t_k`)`	k	`LEFT(`t_k`)`	`RIGHT(`t_k`)`	`L(`t_k`)`	`H(`t_k`)`
1	${\sim}8$	t_1	t_2	t_3	5	${\sim}4$	t_5	t_7	t_8
2	${\sim}7$	t_2	0	t_4	6	${\sim}0$	t_7	0	1
3	${\sim}6$	t_3	t_4	t_5	7	${\sim}3$	t_7	0	1
4	${\sim}5$	t_4	t_7	1	8	${\sim}2$	t_8	1	0

(but they can then be discarded). The resulting BDD for $f \wedge g$ is

k	`V(`f_k`)`	`LO(`f_k`)`	`HI(`f_k`)`	k	`V(`f_k`)`	`LO(`f_k`)`	`HI(`f_k`)`
2	4	1	0	6	2	5	4
3	4	0	1	7	2	0	5
4	3	3	2	8	1	7	6.
5	3	3	1				

68. If `LEFT(`t`)` < 0 at the beginning of step S10, set `RIGHT(`t`)` $\leftarrow t$, $q \leftarrow$ `NTOP`, `NTOP` $\leftarrow q + 1$, `LEFT(`t`)` $\leftarrow {\sim}(q - f_0)$, `LO(`$q$`)` $\leftarrow$ ${\sim}$`LEFT(L(`t`))`, `HI(`q`)` $\leftarrow$ ${\sim}$`LEFT(H(`t`))`, `V(`q`)` $\leftarrow l$, and return to S9.

69. Make sure that `NTOP` $\le$ `TBOT` at the end of step S1 and when going from S11 to S9. (It's *not* necessary to make this test inside the loop of S11.) Also make sure that `NTOP` $\le$ `HBASE` just after setting `HBASE` in step S4.

70. This choice would make the hash table a bit smaller; memory overflow would therefore be slightly less likely, at the expense of slightly more collisions. But it also would slow down the action, because *make_template* would have to check that `NTOP` $\le$ `TBOT` whenever `TBOT` decreases.

71. Add a new field, `EXTRA(`t`)` $= \alpha''$, to each template t (see (43)).

72. In place of steps S4 and S5, use the approach of Algorithm R to bucket-sort the elements of the linked lists that begin at `LLIST[`l`]` and `HLIST[`l`]`. This is possible if an extra one-bit hint is used within the pointers to distinguish links in the `L` fields from links in the `H` fields, because we can then determine the LO and HI parameters of t's descendants as a function of t and its "parity."

73. If the BDD profile is $(b_0, \ldots, b_n)$, we can assign $p_j = \lceil b_{j-1}/2^e \rceil$ pages to branches on x_j. Auxiliary tables of $p_1 + \cdots + p_{n+1} \le \lceil B(f)/2^e \rceil + n$ short integers allow us to compute $V(p) = T[\pi(p)]$, $\mathrm{LO}(p) = \mathtt{LO}(M[\pi(p)] + \sigma(p))$, $\mathrm{HI}(p) = \mathtt{HI}(M[\pi(p)] + \sigma(p))$.

For example, if $e = 12$ and $n < 2^{16}$, we can represent arbitrary BDDs of up to $2^{32} - 2^{28} + 2^{16} + 2^{12}$ nodes with 32-bit virtual `LO` and `HI` pointers. Each BDD requires appropriate auxiliary T and M tables of size $\le 2^{20}$, constructible from its profile.

[This method can significantly improve caching behavior. It was inspired by the paper of P. Ashar and M. Cheong, *Proc. International Conf. Computer-Aided Design* (IEEE, 1994), 622–627, which also introduced algorithms similar to Algorithm S.]

74. The required condition is now $\mu_n(x_1, \ldots, x_{2^n}) \wedge [\bar{x}_1 = x_{2^n}] \wedge \cdots \wedge [\bar{x}_{2^{n-1}} = x_{2^{n-1}+1}]$. If we set $y_1 = x_1$, $y_2 = x_3$, $\ldots$, $y_{2^{n-2}} = x_{2^{n-1}-1}$, $y_{2^{n-2}+1} = \bar{x}_{2^{n-1}}$, $y_{2^{n-2}+2} = \bar{x}_{2^{n-1}-2}$, $\ldots$, $y_{2^{n-1}} = \bar{x}_2$, (49) yields the equivalent condition $\mu_{n-1}(y_1, \ldots, y_{2^{n-1}}) \wedge [y_{2^{n-2}} \le \bar{y}_{2^{n-2}+1}] \wedge [y_{2^{n-2}-1} \le \bar{y}_{2^{n-2}+2}] \wedge \cdots \wedge [y_1 \le \bar{y}_{2^{n-1}}]$, which is eminently suitable for evaluation by Algorithm S. (The evaluation should be from left to right; right-to-left would generate enormous intermediate results.)

With this approach we find that there are respectively 1, 2, 4, 12, 81, 2646, 1422564, 229809982112 monotone self-dual functions of 1, 2, $\ldots$, 8 variables. (See Table 7.1.1–3 and answer 7.1.2–88.) The 8-variable functions are characterized by a BDD of 130,305,082 nodes; Algorithm S needs about 204 gigamems to compute it.

75. Begin with $\rho_1(x_1, x_2) = [x_1 \le x_2]$, and replace $G_{2^n}(x_1, \ldots, x_{2^n})$ in (49) by the function $H_{2^n}(x_1, \ldots, x_{2^n}) = [x_1 \le x_2 \le x_3 \le x_4] \wedge \cdots \wedge [x_{2^n-3} \le x_{2^n-2} \le x_{2^n-1} \le x_{2^n}]$.

(It turns out that $B(\rho_9) = 3{,}683{,}424$; about 170 megamems suffice to compute that BDD, and ρ_{10} is almost within reach. Algorithm C now quickly yields the exact numbers of regular n-variable Boolean functions for $1 \le n \le 9$, namely 3, 5, 10, 27, 119, 1173, 44315, 16175190, 284432730176. Similarly, we can count the self-dual ones, as in exercise 74; those numbers, whose early history is discussed in answer 7.1.1–123, are 1, 1, 2, 3, 7, 21, 135, 2470, 319124, 1214554343, for $1 \le n \le 10$.)

76. Say that $x_0 \ldots x_{j-1}$ *forces* x_j if $x_i = 1$ for some $i \subseteq j$ with $0 \le i < j$. Then $x_0 x_1 \ldots x_{2^n-1}$ corresponds to a clutter if and only if $x_j = 0$ whenever $x_0 \ldots x_{j-1}$ forces x_j, for $0 \le j < 2^n$. And $\mu_n(x_0, \ldots, x_{2^n-1}) = 1$ if and only if $x_j = 1$ whenever $x_0 \ldots x_{j-1}$ forces x_j. So we get the desired BDD from that of $\mu_n(x_1, \ldots, x_{2^n})$ by (i) changing each branch j to $j{-}1$, and (ii) interchanging the LO and HI branches at every branch node that has LO $= \boxed{\bot}$. (Notice that, by Corollary 7.1.1Q, the prime implicants of every monotone Boolean function correspond to clutters.)

77. Continuing the previous answer, say that the bit vector $x_0 \ldots x_{k-1}$ is *consistent* if we have $x_j = 1$ whenever $x_0 \ldots x_{j-1}$ forces x_j, for $0 < j < k$. Let b_k be the number of consistent vectors of length k. For example, $b_4 = 6$ because of the vectors $\{0000, 0001, 0011, 0101, 0111, 1111\}$. Notice that exactly $c_k = b_{k+1} - b_k$ clutters $\mathcal{S}$ have the properly that k represents their "largest" set, $\max\{s \mid s \text{ represents a set of } \mathcal{S}\}$.

The BDD for $\mu_n(x_1, \ldots, x_{2^n})$ has b_{k-1} branch nodes k when $1 \le k \le 2^{n-1}$. Proof: Every subfunction defined by x_1, $\ldots$, x_{k-1} is either identically false or defines a consistent vector $x_1 \ldots x_{k-1}$. In the latter case the subfunction is a bead, because it takes different values under certain settings of x_{k+1}, $\ldots$, x_{2^n}. Indeed, if $x_1 \ldots x_{k-1}$

profile
caching
Ashar
Cheong
regular functions, enumerated
prime implicants
clutters

forces x_k, we set $x_{k+1} \leftarrow \cdots \leftarrow x_{2^n} \leftarrow 1$; otherwise we set $x_j \leftarrow y_j$ for $k < j \le 2^n$, where $y_{j+1} = [x_{i+1}{=}1$ for some $i \subseteq j$ with $i+1 < k]$, noting that $y_{2^n+k} = 0$.

On the other hand there are $b_{k'}$ branches (k) when $k = 2^n - k'$ and $0 \le k' < 2^{n-1}$. In this case the nonconstant subfunctions arising from $x_1, \ldots, x_{k-1}$ lead to values y_j as above, where the vector $\bar{y}_{0'}\bar{y}_{1'} \ldots \bar{y}_{k'}$ is consistent. (Here $0' = 2^n$, $1' = 2^n - 1$, etc.) Conversely, every such consistent vector describes such a subfunction; we can, for example, set $x_j \leftarrow 0$ when $j < k - 2^{n-1}$ or $2^{n-1} \le j < k$, otherwise $x_j \leftarrow y_{2^{n-1}+j}$. This subfunction is a bead if and only if $y_{k'} = 1$ or $\bar{y}_{0'} \ldots \bar{y}_{(k-1)'}$ forces $\bar{y}_{k'}$. Thus the beads correspond to consistent vectors of length k'; and different vectors define different beads.

This argument shows that there are $b_{k-1} - c_{k-1}$ branches (k) with LO $= \boxed{\bot}$ when $1 \le k \le 2^{n-1}$ and c_{2^n-k} such branches when $2^{n-1} < k \le 2^n$. Hence exactly half of the $B(\mu_n) - 2$ branch nodes have LO $= \boxed{\bot}$.

78. To count graphs on n labeled vertices with maximum degree $\le d$, construct the Boolean function of the $\binom{n}{2}$ variables in its adjacency matrix, namely $\bigwedge_{k=1}^{n} S_{\le d}(X_k)$, where X_k is the set of variables in row k of the matrix. For example, when $n = 5$ there are 10 variables, and the function is $S_{\le d}(x_1, x_2, x_3, x_4) \wedge S_{\le d}(x_1, x_5, x_6, x_7) \wedge S_{\le d}(x_2, x_5, x_8, x_9) \wedge S_{\le d}(x_3, x_6, x_8, x_{10}) \wedge S_{\le d}(x_4, x_7, x_9, x_{10})$. When $n = 12$ the BDDs for $d = (1, 2, \ldots, 10)$ have respectively (5960, 137477, 1255813, 5295204, 10159484, 11885884, 9190884, 4117151, 771673, 28666) nodes, so they are readily computed with Algorithm S. To count solutions with maximum degree d, subtract the number of solutions for degree $\le d{-}1$ from the number for degree $\le d$; the answers for $0 \le d \le 11$ are:

1	3038643940889754	29271277569846191555
140151	211677202624318662	17880057008325613629
3568119351	3617003021179405538	4489497643961740521
8616774658305	17884378201906645374	430038382710483623

[In general there are $t_n - 1$ graphs on n labeled vertices with maximum degree 1, where t_n is the number of involutions, Eq. 5.1.4–(40).]

The methods of Section 7.2.3 are superior to BDDs for enumerations such as these, when n is large, because labeled graphs have $n!$ symmetries. But when n has a moderate size, BDDs produce answers quickly, and nicely characterize all the solutions.

79. In the following counts, obtained from the BDDs in the previous answer, each graph with k edges is weighted by 2^{66-k}. Divide by 3^{66} to get probabilities.

73786976294838206464	11646725483430295546484263747584
553156749930805290074112	7767741687870924305547518803968
598535502868315236548476928	2514457534558975918608668688384
68379835220584550117167595520	452733615636089939218193403904
1380358927564577683479233298432	45968637738881805341545676736
7024096376298397076969081536512	2093195580480313818292294985

80. If the original functions f and g have no BDD nodes in common, both algorithms encounter almost exactly the same subproblems: Algorithm S deals with all nodes of $f \diamond g$ that aren't descended from nodes of the forms $\alpha \diamond \boxed{\bot}$ or $\boxed{\bot} \diamond \beta$, while (55) also avoids nodes that descend from the forms $\alpha \diamond \boxed{\top}$ or $\boxed{\top} \diamond \beta$. Furthermore, (55) takes shortcuts when it meets nontrivial subproblems AND(f', g') with $f' = g'$; Algorithm S cannot recognize the fact that such cases are easy. And (55) can also win if it happens to stumble across a relevant memo left over from a previous computation.

81. Just change 'AND' to 'XOR' and '$\wedge$' to '$\oplus$' throughout. The simple cases are now $f \oplus 0 = f$, $0 \oplus g = g$, and $f \oplus g = 0$ if $f = g$. We should also swap $f \leftrightarrow g$ if $f > g \ne 0$.

adjacency matrix
involutions

Notes: The author experimentally inserted further memos '$f \oplus r = g$' and '$g \oplus r = f$' in the bottom line; but these additional cache entries seemed to do more harm than good. Considering other binary operators, there's no need to implement both $\mathrm{BUTNOT}(f,g) = f \wedge \bar{g}$ and $\mathrm{NOTBUT}(f,g) = \bar{f} \wedge g$, since the latter is $\mathrm{BUTNOT}(g,f)$. Also, $\mathrm{XOR}(1, \mathrm{OR}(f,g))$ may be better than an implementation of $\mathrm{NOR}(f,g) = \neg(f \vee g)$.

Knuth BUTNOT NOTBUT dead dereference Somenzi Knuth, Don Somenzi

82. A top-level computation of $F \leftarrow \mathrm{AND}(f,g)$ begins with f and g in computer registers, but `REF`(f) and `REF`(g) do not include "references" such as those. (We do, however, assume that f and g are both alive.)

If (55) discovers that $f \wedge g$ is obviously r, it increases `REF`(r) by 1.

If (55) finds $f \wedge g = r$ in the memo cache, it increases `REF`(r), and recursively increases `REF(LO`(r)`)` and `REF(HI`(r)`)` in the same way if r was dead.

If step U1 finds $p = q$, it *decreases* `REF`(p) by 1 (believe it or not); this won't kill p.

If step U2 finds r, there are two cases: If r was alive, it sets `REF`$(r) \leftarrow$ `REF`$(r) + 1$, `REF`$(p) \leftarrow$ `REF`$(p) - 1$, `REF`$(q) \leftarrow$ `REF`$(q) - 1$. Otherwise it simply sets `REF`$(r) \leftarrow 1$.

When step U3 creates a new node r, it sets `REF`$(r) \leftarrow 1$.

Finally, after the top-level AND returns a value r that we wish to assign to F, we must first *dereference* F, if $F \neq \Lambda$; this means setting `REF`$(F) \leftarrow$ `REF`$(F) - 1$, and recursively dereferencing `LO`(F) and `HI`(F) if `REF`(F) has become 0. Then we set $F \leftarrow r$ (without adjusting `REF`(r)).

[Furthermore, in a quantification routine such as (65) or in the composition routine (72), both r_l and r_h should be dereferenced after the OR or MUX has computed r.]

83. Exercise 61 shows that the subproblem $f \wedge g$ occurs at most once per top-level call, when `REF`(f) = `REF`(g) = 1. [This idea is due to F. Somenzi; see the paper cited in answer 84. Many nodes have reference count 1, because the average count is approximately 2, and because the sinks usually have large counts. However, such cache-avoidance did not improve the overall performance in the author's experiments, possibly because of the examples investigated, or possibly because "accidental" cache hits in other top-level operations can be useful.]

84. Many possibilities exist, and no simple technique appears to be a clear winner. The cache and table sizes should be powers of 2, to facilitate calculating the hash functions. The size of the unique table for x_v should be roughly proportional to the number of nodes that currently branch on x_v (alive or dead). It's necessary to rehash everything when a table is downsized or upsized.

In the author's experiments while writing this section, the cache size was doubled whenever the number of insertions since the beginning of the most recent top-level command exceeded $\ln 2$ times the current cache size. (At that point a random hash function will have filled about half of the slots.) After garbage collection, the cache was downsized, if necessary, so that it either had 256 slots or was at least 1/4 full.

It's easy to keep track of the current number of dead nodes; hence we know at all times how much memory a garbage collection will reclaim. The author obtained satisfactory results by inserting a new step U2$\frac{1}{2}$ between U2 and U3: "Increase C by 1, where C is a global counter. If $C \bmod 1024 = 0$, and if at least 1/8 of all current nodes are dead, collect garbage."

[See F. Somenzi, *Software Tools for Technology Transfer* **3** (2001), 171–181 for numerous further suggestions based on extensive experience.]

85. The complete table would have 2^{32} entries of 32 bits each, for a total of 2^{34} bytes (≈ 17.2 gigabytes). The BDD base discussed after (58), with about 136 million nodes using zip-ordered bits, can be stored in about 1.1 gigabyte; the one discussed in Corollary Y, which ranks all of the multiplier bits first, needs only about 400 megabytes.

86. If $f=0$ or $g=h$, return g. If $f=1$, return h. If $g=0$ or $f=g$, return $\mathrm{AND}(f,h)$. If $h=1$ or $f=h$, return $\mathrm{OR}(f,g)$. If $g=1$, return $\mathrm{IMPLIES}(f,h)$; if $h=0$, return $\mathrm{BUTNOT}(g,f)$. (If binary IMPLIES and/or BUTNOT aren't implemented directly, it's OK to let the corresponding cases propagate in ternary guise.)

87. Sort so that $f\le g\le h$. If $f=0$, return $\mathrm{AND}(g,h)$. If $f=1$, return $\mathrm{OR}(g,h)$. If $f=g$ or $g=h$, return g.

88. The trio of functions $(f,g,h)=(R_0,R_1,R_2)$ makes an amusing example, when

$$R_a(x_1,\ldots,x_n)=\bigl[(x_n\ldots x_1)_2 \bmod 3\ne a\bigr]=R_{(2a+x_1)\bmod 3}(x_2,\ldots,x_n).$$

Thanks to the memos, the ternary recursion finds $f\wedge g\wedge h=0$ by examining only one case at each level; the binary computation of, say, $f\wedge g=\bar h$ definitely takes longer.

More dramatically, let $f=x_1\wedge(x_2?\ F{:}\ G)$, $g=x_2\wedge(x_1?\ G{:}\ F)$, and $h=x_1?\ \bar x_2\wedge F{:}\ x_2\wedge G$, where F and G are functions of $(x_3,\ldots,x_n)$ such that $B(F\wedge G)=\Theta(B(F)B(G))$ as in exercise 63. Then $f\wedge g$, $g\wedge h$, and $h\wedge f$ all have large BDDs, but the ternary recursion immediately discovers that $f\wedge g\wedge h=0$.

89. (a) True; the left side is $(f_{00}\vee f_{01})\vee(f_{10}\vee f_{11})$, the right side is $(f_{00}\vee f_{10})\vee(f_{01}\vee f_{11})$.
(b) Similarly true. (And Ↄ's are commutative too.)
(c) Usually false; see part (d).
(d) $\forall x_1\exists x_2 f=(f_{00}\vee f_{01})\wedge(f_{10}\vee f_{11})=(\exists x_2\forall x_1 f)\vee(f_{00}\wedge f_{11})\vee(f_{01}\wedge f_{10})$.

90. Change $\exists j_1\ldots\exists j_m$ to $\text{Ↄ}j_1\ldots\text{Ↄ}j_m$.

91. (a) $f\downarrow 1=f$, $f\downarrow x_j=f_1$, and $f\downarrow\bar x_j=f_0$, in the notation of (63).
(b) This distributive law is obvious, by the definition of $\downarrow$. (Also true for $\vee$, $\oplus$, etc.)
(c) True if and only if g is not identically zero. (Consequently the value of $f(x_1,\ldots,x_n)\downarrow g$ for $g\ne 0$ is determined solely by the values of $x_j\downarrow g$ for $1\le j\le n$.)
(d) $f(x_1,1,0,x_4,0,1,x_7,\ldots,x_n)$. This is the restriction of f with respect to $x_2=1$, $x_3=0$, $x_5=0$, $x_6=1$ (see exercise 57), also called the *cofactor* of f with respect to the subcube g. (A similar result holds when g is any product of literals.)
(e) $f(x_1,\ldots,x_{n-1},x_1\oplus\cdots\oplus x_{n-1}\oplus 1)$. (Consider the case $f=x_j$, for $1\le j\le n$.)
(f) $x_1?\ f(1,\ldots,1){:}\ f(0,\ldots,0)$.
(g) $f(1,x_2,\ldots,x_n)\downarrow g(x_2,\ldots,x_n)$.
(h) If $f=x_2$ and $g=x_1\vee x_2$ we have $f\downarrow g=\bar x_1\vee x_2$.
(i) $\mathrm{CONSTRAIN}(f,g)$ = "If $f\downarrow g$ has an obvious value, return it. Otherwise, if $f\downarrow g=r$ is in the memo cache, return r. Otherwise represent f and g as in (52); set $r\leftarrow\mathrm{CONSTRAIN}(f_h,g_h)$ if $g_l=0$, $r\leftarrow\mathrm{CONSTRAIN}(f_l,g_l)$ if $g_h=0$, otherwise $r\leftarrow\mathrm{UNIQUE}(v,\mathrm{CONSTRAIN}(f_l,g_l),\mathrm{CONSTRAIN}(f_h,g_h))$; put '$f\downarrow g=r$' into the memo cache, and return r." Here the obvious values are $f\downarrow 0=0\downarrow g=0$; $f\downarrow 1=f$; $1\downarrow g=g\downarrow g=[g\ne 0]$.

[The operator $f\downarrow g$ was introduced in 1989 by O. Coudert, C. Berthet, and J. C. Madre. Examples such as the functions in (h) led them to propose also the modified operator $f\Downarrow g$, "f restricted to g," which has a similar recursion except that it uses $f\Downarrow(\exists x_v g)$ instead of $(\bar x_v?\ f_l\Downarrow g_l{:}\ f_h\Downarrow g_h)$ when $f_l=f_h$. See *Lecture Notes in Computer Science* **407** (1989), 365–373.]

92. See answer 91(d) for the "if" part. Notice also that (i) $x_1\downarrow g=x_1$ if and only if $g_0\ne 0$ and $g_1\ne 0$, where $g_c=g(c,x_2,\ldots,x_n)$; (ii) $x_n\downarrow g=x_n$ if and only if $\text{Ↄ}x_n g=0$ and $g\ne 0$.

zip-ordered
IMPLIES
BUTNOT
remainders mod 3
associative law
commutative
distributive law
restriction
cofactor
literals
Coudert
Berthet
Madre
restricted to

$J(x;f)$ function
junction function
3-colored
Sauerhoff
Wegener
tail recursion
Rudell
yes/no quantifiers
contiguous-USA

Suppose $f^\pi{\downarrow}g^\pi = (f{\downarrow}g)^\pi$ for all f and π. If $g \neq 0$ isn't a subcube, there's an index j such that $g_0 \neq 0$ and $g_1 \neq 0$ and $\cap x_j\, g \neq 0$, where $g_c = g(x_1,\ldots,x_{j-1},c,x_{j+1},\ldots,x_n)$. By the previous paragraph, we have (i) $x_j \downarrow g = x_j$ and (ii) $x_j \downarrow g \neq x_j$, a contradiction.

93. Let $f = J(x_1,\ldots,x_n; f_1,\ldots,f_n)$ and $g = J(x_1,\ldots,x_n; g_1,\ldots,g_n)$, where

$$f_v = x_{n+1} \vee \cdots \vee x_{5n} \vee J(x_{5n+1},\ldots,x_{6n}; [v{-}1],\ldots,[v{-}n]),$$
$$g_v = x_{n+1} \vee \cdots \vee x_{5n} \vee J(x_{5n+1},\ldots,x_{6n}; [v{=}1]{+}[v{-}1],\ldots,[v{=}n]{+}[v{-}n]),$$

and J is the junction function of exercise 52.

If G can be 3-colored, let $\hat f = J(x_1,\ldots,x_n; \hat f_1,\ldots,\hat f_n)$, where

$$\hat f_v = x_{n+1} \vee \cdots \vee x_{5n} \vee J(x_{5n+1},\ldots,x_{6n}; \hat f_{v1},\ldots,\hat f_{vn}),$$

and $\hat f_{vw} = [v$ and w have different colors]. Then $B(\hat f) < n + 3(5n) + 2$.

Conversely, suppose there's an approximating $\hat f$ such that $B(\hat f) < 16n + 2$, and let $\hat f_v$ be the subfunction with $x_1 = [v{=}1]$, $\ldots$, $x_n = [v{=}n]$. At most three of these subfunctions are distinct, because every distinct $\hat f_v$ must branch on each of x_{n+1}, $\ldots$, x_{5n}. Color the vertices so that u and v get the same color if and only if $\hat f_u = \hat f_v$; this can happen only if $u \mathrel{-\!\!\!/\!\!\!-} v$, so the coloring is legitimate.

[M. Sauerhoff and I. Wegener, *IEEE Transactions* **CAD-15** (1996), 1435–1437.]

94. *Case 1:* $v \neq g_v$. Then we aren't quantifying over x_v; hence $g = g_h$, and $f \mathbin{\mathrm{E}} g = \bar x_v?\ f_l \mathbin{\mathrm{E}} g\colon f_h \mathbin{\mathrm{E}} g$.

Case 2: $v = g_v$. Then $g = x_v \wedge g_h$ and $f \mathbin{\mathrm{E}} g = (f_l \mathbin{\mathrm{E}} g_h) \vee (f_h \mathbin{\mathrm{E}} g_h) = r_l \vee r_h$. In the subcase $v \neq f_v$, we have $f_l = f_h = f$; hence $r_l = r_h$, and we can directly reduce $f \mathbin{\mathrm{E}} g$ to $f \mathbin{\mathrm{E}} g_h$ (an instance of "tail recursion").

[Rudell observes that the order of quantification in (65) corresponds to bottom-up order of the variables. That order is convenient, but not always best; sometimes it's better to remove the $\exists$s one by one in another order, based on knowledge of the functions involved.]

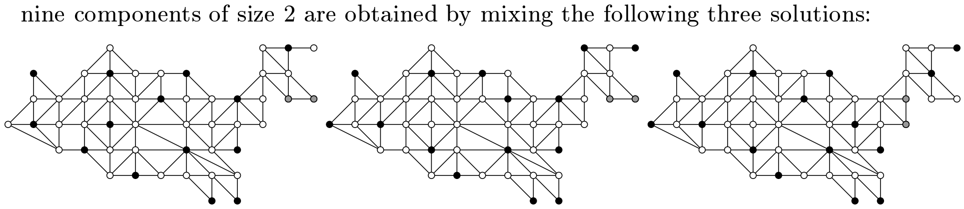
95. If $r_l = 1$ and $v = g_v$, we can set $r \leftarrow 1$ and forget about r_h. (This change led to a 100-fold speedup in some of the author's experiments.)

96. For $\forall$, just change E to A and OR to AND. For $\cap$, change E to D and OR to XOR; also, if $v \neq f_v$, return 0. [Routines for the yes/no quantifiers ⅄ and ℕ are analogous to $\cap$. Yes/no quantifiers should be used only when $m = 1$; otherwise they make little sense.]

97. Proceeding bottom-up, the amount of work on each level is at worst proportional to the number of nodes on that level.

98. The function $\mathrm{NOTEND}(x) = \exists y \exists z(\mathrm{ADJ}(x,y) \wedge \mathrm{ADJ}(x,z) \wedge [y \neq z])$ identifies all vertices of degree ≥ 2. Hence $\mathrm{ENDPT}(x) = \mathrm{KER}(x) \wedge \neg\mathrm{NOTEND}(x)$. And $\mathrm{PAIR}(x,y) = \mathrm{ENDPT}(x) \wedge \mathrm{ENDPT}(y) \wedge \mathrm{ADJ}(x,y)$.

[For example, when G is the contiguous-USA graph, with the states ordered as in (104), we have $B(\mathrm{NOTEND}) = 992$, $B(\mathrm{ENDPT}) = 264$, and $B(\mathrm{PAIR}) = 203$. Before applying $\exists y \exists z$ the BDD size is 50511. There are exactly 49 kernels of degree 1. The nine components of size 2 are obtained by mixing the following three solutions:



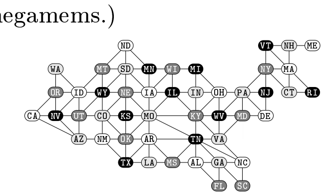
The total cost of this calculation, using the stated algorithms, is about 14 megamems, in 6.3 megabytes of memory — only about 52 memory references per kernel.]

99. Find a triangle of mutually adjacent states, and fix their colors. The BDD size also decreases substantially if we choose states of high degree in the "middle" levels. For example, by setting $a_{\mathtt{MO}} = b_{\mathtt{MO}} = a_{\mathtt{TN}} = \bar{b}_{\mathtt{TN}} = \bar{a}_{\mathtt{AR}} = b_{\mathtt{AR}} = 1$ we reduce the 25,579 nodes to only 4642 (and the total execution time also drops below 2 megamems).

[Bryant's original manuscript about BDDs discussed graph coloring in detail, but he decided to substitute other material when his paper was published in 1986.]

100. Replace $\mathrm{IND}(x_{\mathtt{ME}}, \ldots, x_{\mathtt{CA}})$ by $\mathrm{IND}(x_{\mathtt{ME}}, \ldots, x_{\mathtt{CA}}) \wedge S_{12}(x_{\mathtt{ME}}, \ldots, x_{\mathtt{CA}})$, to get the 12-node independent sets; this BDD has size 1964. Then use (73) as before, and the trick of answer 99, getting a COLOR function with 184,260 nodes and 12,554,677,864 solutions. (The running time is approximately 26 megamems.)

101. If a state's weight is w, assign $2w$ and w as the respective weights of its a and b variables, and use Algorithm B. (For example, variable $a_{\mathtt{WY}}$ gets weight $2(23 + 25) = 96$.) The solution, shown here with color codes ①②③④, is unique.



102. The main idea is that, when g_j changes, all results in the cache for functions with $f_v > j$ remain valid. To exploit this principle we can maintain an array of "time stamps" $G_1 \geq G_2 \geq \cdots \geq G_n \geq 0$, one for each variable. There's a master clock time $G \geq G_1$, representing the number of distinct compositions done or prepared; another variable G' records whether G has changed since COMPOSE was last invoked. Initially $G = G' = G_1 = \cdots = G_n = 0$. The subroutine $\mathrm{NEWG}(j, g)$ is implemented as follows:

N1. [Easy case?] If $g_j = g$, exit the subroutine. Otherwise set $g_j \leftarrow g$.

N2. [Can we reset?] If $g \neq x_j$, or if $j < n$ and $G_{j+1} > 0$, go to N4.

N3. [Reset stamps.] While $j > 0$ and $g_j = x_j$, set $G_j \leftarrow 0$ and $j \leftarrow j - 1$. Then if $j = 0$, set $G \leftarrow G - G'$, $G' \leftarrow 0$, and exit.

N4. [Update G?] If $G' = 0$, set $G \leftarrow G + 1$ and $G' \leftarrow 1$.

N5. [New stamps.] While $j > 0$ and $G_j \neq G$, set $G_j \leftarrow G$ and $j \leftarrow j - 1$. Exit. ▌

(Reference counts also need to be maintained appropriately.) Before launching a top-level call of COMPOSE, set $G' \leftarrow 0$. Change the COMPOSE routine (72) to use $f[G_v]$ in references to the cache, where $v = f_v$; the test '$v > m$' becomes '$G_v = 0$'.

103. The equivalent formula $g(f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$ can be implemented with the COMPOSE operation (72). (However, Dull was vindicated when it turned out that his formula could be evaluated more than a hundred times faster than Quick's, in spite of the fact that it uses twice as many variables! In his application, the computation of $(y_1 = f_1(x_1, \ldots, x_n)) \wedge \cdots \wedge (y_m = f_m(x_1, \ldots, x_n)) \wedge g(y_1, \ldots, y_m)$ turned out to be much easier than COMPOSE's computation of $g_j(f_1, \ldots, f_m)$ for every subfunction g_j of g; see, for example, exercise 162.)

104. The following recursive algorithm $\mathrm{COMPARE}(f, g)$ needs at most $O(B(f)B(g))$ steps when used with a memo cache: If $f = g$, return '='. Otherwise, if $f = 0$ or $g = 1$, return '<'; if $f = 1$ or $g = 0$, return '>'. Otherwise represent f and g as in (52); compute $r_l \leftarrow \mathrm{COMPARE}(f_l, g_l)$. If r_l is '$\|$', return '$\|$'; otherwise compute $r_h \leftarrow \mathrm{COMPARE}(f_h, g_h)$. If r_h is '$\|$', return '$\|$'. Otherwise if r_l is '=', return r_h; if r_h is '=', return r_l; if $r_l = r_h$, return r_l. Otherwise return '$\|$'.

Bryant
symmetric func
time stamps
Reference counts
composition
memo cache

105. (a) A unate function with polarities $(y_1, \ldots, y_n)$ has $\text{⅄}x_j f = 0$ when $y_j = 1$ and $\mathsf{N}x_j f = 0$ when $y_j = 0$, for $1 \le j \le n$. Conversely, f is unate if these conditions hold for all j. (Notice that $\text{⅄}x_j f = \mathsf{N}x_j f = 0$ if and only if $\square x_j f = 0$, if and only if f doesn't depend on x_j. In such cases y_j is irrelevant; otherwise y_j is uniquely determined.)

(b) The following algorithm maintains global variables $(p_1, \ldots, p_n)$, initially zero, with the property that $p_j = +1$ if y_j must be 0 and $p_j = -1$ if y_j must be 1; p_j will remain zero if f doesn't depend on x_j. With this understanding, UNATE(f) is defined as follows: If f is constant, return *true*. Otherwise represent f as in (50). Return *false* if either UNATE(f_l) or UNATE(f_h) is *false*; otherwise set $r \leftarrow$ COMPARE(f_l, f_h) using exercise 104. If r is '$\|$', return *false*. If r is '$<$', return *false* if $p_v < 0$, otherwise set $p_v \leftarrow +1$ and return *true*. If r is '$>$', return *false* if $p_v > 0$, otherwise set $p_v \leftarrow -1$ and return *true*.

This algorithm often terminates quickly. It relies on the fact that $f(x) \le g(x)$ for all x if and only if $f(x \oplus y) \le g(x \oplus y)$ for all x, when y is fixed. If we simply want to test whether or not f is monotone, the p variables should be initialized to $+1$ instead of 0.

106. Define HORN(f, g, h) thus: If $f > g$, interchange $f \leftrightarrow g$. Then if $f = 0$ or $h = 1$, return *true*. Otherwise if $g = 1$ or $h = 0$, return *false*. Otherwise represent f, g, and h as in (59). Return *true* if HORN(f_l, g_l, h_l), HORN(f_l, g_h, h_l), HORN(f_h, g_l, h_l), and HORN(f_h, g_h, h_h) are all *true*; otherwise return *false*. [This algorithm is due to T. Horiyama and T. Ibaraki, *Artificial Intelligence* **136** (2002), 189–213, who also introduced an algorithm similar to that of answer 105(b).]

107. Let $e\$f\$g\$h$ mean that $e(x) = f(y) = g(z) = 1$ implies $h(\langle xyz \rangle) = 1$. Then f is Krom if and only if $f\$f\$f\$f$, and we can use the following recursive algorithm KROM(e, f, g, h): Rearrange $\{e, f, g\}$ so that $e \le f \le g$. Then if $e = 0$ or $h = 1$, return *true*. Otherwise if $f = 1$ or $h = 0$, return *false*. Otherwise represent e, f, g, h with the quaternary analog of (59). Return *true* if KROM(e_l, f_l, g_l, h_l), KROM(e_l, f_l, g_h, h_l), KROM(e_l, f_h, g_l, h_l), KROM(e_l, f_h, g_h, h_h), KROM(e_h, f_l, g_l, h_l), KROM(e_h, f_l, g_h, h_h), KROM(e_h, f_h, g_l, h_h), and KROM(e_h, f_h, g_h, h_h) are all *true*; otherwise return *false*.

108. Label the nodes $\{1, \ldots, s\}$ with root 1 and sinks $\{s-1, s\}$; then $(s-3)!$ permutations of the other labels give different dags for the same function. The stated inequality follows because each instruction $(\bar{v}_k?\ l_k\colon h_k)$ has at most $n(s-1)^2$ possibilities, for $1 \le k \le s-2$. (In fact, it holds also for arbitrary *branching programs*, namely for binary decision diagrams in general, whether or not they are ordered and/or reduced.)

Since $1/(s-3)! < (s-1)^3/s!$ and $s! > (s/e)^s$, we have (generously) $b(n,s) < (nse)^s$. Let $s_n = 2^n/(n+\theta)$, where $\theta = \lg e = 1/\ln 2$; then $\lg b(n, s_n) < s_n \lg(n s_n e) = 2^n(1 - (\lg(1+\theta/n))/(n+\theta)) = 2^n - \Omega(2^n/n^2)$. So the probability that a random n-variable Boolean function has $B(f) \le s_n$ is at most $1/2^{\Omega(2^n/n^2)}$. And that is really tiny.

109. $1/2^{\Omega(2^n/n^2)}$ is really tiny even when multiplied by $n!$.

110. Let $f_n = M_m(x_{n-m+1}, \ldots, x_n; 0, \ldots, 0, x_1, \ldots, x_{n-m}) \vee (\bar{x}_{n-m+1} \wedge \cdots \wedge \bar{x}_n \wedge [0 \ldots 0 x_1 \ldots x_{n-m} \text{ is a square}])$, when $2^{m-1} + m - 1 < n < 2^m + m$. Each term of this formula has $2^m + m - n$ zeros; the second term destroys all of the 2^m-bit squares. [See H.-T. Liaw and C.-S. Lin, *IEEE Transactions* **C-41** (1992), 661–664; Y. Breitbart, H. Hunt III, and D. Rosenkrantz, *Theoretical Comp. Sci.* **145** (1995), 45–69.]

111. Let $\mu n = \lambda(n - \lambda n)$, and notice that $\mu n = m$ if and only if $2^m + m \le n < 2^{m+1} + m + 1$. The sum for $0 \le k < n - \mu n$ is $2^{n-\mu n} - 1$; the other terms sum to $2^{2^{\mu n}}$.

112. Suppose $k = n - \lg n + \lg \alpha$. Then

$$\frac{(2^{2^{n-k}} - 1)^{2^k}}{2^{2^n}} = \exp\Bigl(\frac{2^n \alpha}{n} \ln\Bigl(1 - \frac{1}{2^{n/\alpha}}\Bigr)\Bigr) = \exp\Bigl(-\frac{2^{n-n/\alpha}\alpha}{n}\Bigl(1 + O\Bigl(\frac{1}{2^{n/\alpha}}\Bigr)\Bigr)\Bigr).$$

depend on
boolean difference
global variables
Horiyama
Ibaraki
branching programs
ordered
reduced
2^m-way multiplex
Liaw
Lin
Breitbart
Hunt
Rosenkrantz

If $\alpha \le \frac{1}{2}$ we have $2^{n-n/\alpha}\alpha/n \le 1/(n2^{n+1})$; hence $\hat{b}_k = (2^{n/\alpha} - 2^{n/(2\alpha)})(2^{n-n/\alpha}\alpha/n) \times (1+O(2^{-n/\alpha})) = 2^k(1-O(2^{-n/(2\alpha)}))$. And if $\alpha \ge 2$ we have $2^{n-n/\alpha}\alpha/n \ge 2^{n/2+1}/n$; thus $\hat{b}_k = (2^{2^{n-k}} - 2^{2^{n-k-1}})(1+O(\exp(-2^{n/2}/n)))$.

[For the variance of b_k, see I. Wegener, *IEEE Trans.* **C-43** (1994), 1262–1269.]

113. The idea looks attractive at first glance, but loses its luster when examined closely. Comparatively few nodes of a BDD base appear on the lower levels, by Theorem U; and algorithms like Algorithm S spend comparatively little of their time dealing with those levels. Furthermore, nonconstant sink nodes would make several algorithms more complicated, especially those for reordering.

114. For example, the truth table might be 01010101 00110011 00001111 00001111.

115. Let $N_k = b_0 + \cdots + b_{k-1}$ be the number of nodes (j) of the BDD for which $j \le k$. The sum of the in-degrees of those nodes is at least N_k; the sum of the out-degrees is $2N_k$; and there's an external pointer to the root. Thus at most $N_k + 1$ branches can cross from the upper k levels to lower levels. Every such branch corresponds to some subtable of order $n-k$. Therefore $q_k \le N_k + 1$.

Moreover, we must have $q_k \le b_k + \cdots + b_n$, because every subtable of order $n-k$ corresponds to a unique bead of order $\le n-k$.

For (124), change 'BDD' to 'ZDD', 'b_k' to 'z_k', 'bead' to 'zead' in these arguments.

116. (a) Let $v_k = 2^{2^k} + 2^{2^{k-1}} + \cdots + 2^{2^0}$. Then $Q(f) \le \sum_{k=1}^{n+1} \min(2^{k-1}, 2^{2^{n+1-k}}) = U_n + v_{\lambda(n-\lambda n)-1}$. Examples like (78) show that this upper bound cannot be improved.

(b) $\hat{q}_k/\hat{b}_k = 2^{2^{n-k}}/(2^{2^{n-k}} - 2^{2^{n-k-1}})$ for $0 \le k < n$; $\hat{q}_n = \hat{b}_n$.

117. $q_k = 2^k$ for $0 \le k \le m$, and $q_{m+k} = 2^m + 2 - k$ for $1 \le k \le 2^m$. Hence $Q(f) = 2^{2m-1} + 7\cdot 2^{m-1} - 1 \approx B(f)^2/8$. (Such fs make QDDs unattractive in practice.)

118. If $n = 2^m - 1$ we have $h_n(x_1,\ldots,x_n) = M_m(z_{m-1},\ldots,z_0;0,x_1,\ldots,x_n)$, where $(z_{m-1}\ldots z_0)_2 = x_1 + \cdots + x_n$ is computable in $5n - 5m$ steps by exercise 7.1.2–30, and M_m takes another $2n + O(\sqrt{n})$ by exercise 7.1.2–39. Since $h_n(x_1,\ldots,x_n) = h_{n+k}(x_1,\ldots,x_n,0,\ldots,0)$, we have $C(h_n) \le 14n + O(\sqrt{n})$ for all n. (A little more work will bring this down to $7n + O(\sqrt{n}\log n)$; can the reader do better?)

The cost of h_4 is $6 = L(h_4)$, and $x_2 \oplus ((x_1 \oplus (x_2 \wedge \bar{x}_4)) \wedge (\bar{x}_3 \oplus (\bar{x}_2 \wedge x_4)))$ is a formula of shortest length. (Also $C(h_5) = 10$ and $L(h_5) = 11$.)

119. True. For example, $S_{2,3,5}(x_1,\ldots,x_6) = h_{13}(x_1,x_2,0,0,1,1,0,1,0,x_3,x_4,x_5,x_6)$.

120. We have $h_n^\pi(x_1,\ldots,x_n) = h_n(y_1,\ldots,y_n)$, where $y_j = x_{j\pi}$ for $1 \le j \le n$. And $h_n(y_1,\ldots,y_n) = y_{y_1+\cdots+y_n} = y_{x_1+\cdots+x_n} = x_{(x_1+\cdots+x_n)\pi}$.

121. (a) If $y_k = \bar{x}_{n+1-k}$ we have $h_n(y_1,\ldots,y_n) = y_{\nu y} = y_{n-\nu x} = \bar{x}_{n+1-(n-\nu x)} = \bar{x}_{\nu x+1}$.

(b) If $x = (x_1,\ldots,x_n)$ and $t \in \{0,1\}$ we have $h_{n+1}(x,t) = (t?\ x_{\nu x+1}\colon x_{\nu x})$.

(c) No. For example, ψ sends $0^k11 \mapsto 0^{k-1}101 \mapsto 0^{k-2}10^21 \mapsto \cdots \mapsto 10^k1 \mapsto 0^k11$. (In spite of its simple definition, ψ has remarkable properties, including fixed points such as 10011010000101011000111001011 and 111011110110010111011111101111.)

(d) In fact, $\hat{h}_n(x_1\ldots x_n) = x_1(!)$, by induction using recurrence (b).

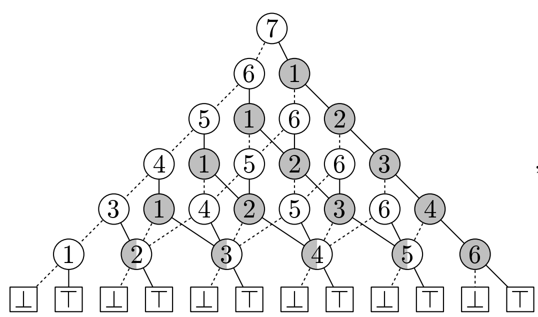
(If $f(x_1,\ldots,x_n)$ is *any* Boolean function and τ is *any* permutation of the binary vectors $x_1\ldots x_n$, we can write $f(x) = \hat{f}(x\tau)$, and the transformed function $\hat{f}$ may well be much easier to work with. Since $f(x) \wedge g(x) = \hat{f}(x\tau) \wedge \hat{g}(x\tau)$, the transform of the AND of two functions is the AND of their transforms, etc. The vector permutations $(x_1\ldots x_n)\pi = x_{1\pi}\ldots x_{n\pi}$ that merely transform the indices, as considered in the text, are a simple special case of this general principle. But the principle is, in a sense, *too* general, because every function f trivially has at least one τ for which $\hat{f}$ is skinny

variance
Wegener
subtable
bead
zead
QDD
2^m-way mux
transformed BDDs

in the sense of exercise 170; all the complexity of f can be transferred to τ. Even simple transformations like ψ have limited utility, because they don't compose well; for example, $\psi\psi$ is not a transformation of the same type. But linear transformations, which take $x \mapsto xT$ for some nonsingular binary matrix T, have proved to be useful ways to simplify BDDs. [See S. Aborhey, *IEEE Trans.* **C-37** (1988), 1461–1465; J. Bern, C. Meinel, and A. Slobodová, *ACM/IEEE Conf. Design Automation* **32** (1995), 408–413; C. Meinel, F. Somenzi, and T. Theobald, *IEEE Trans.* **CAD-19** (2000), 521–533.])

linear transformations
Aborhey
Bern
Meinel
Slobodová
Meinel
Somenzi
Theobald
Sieling
Wegener

122. For example, when $n = 7$ the recurrence in answer 121(b) gives



,

where shaded nodes compute the subfunction h^{DR} on the variables that haven't yet been tested. Simplifications occur at the bottom, because $h_2(x_1,x_2) = x_1$ and $h_2^{DR}(x_1,x_2) = x_2$. [See D. Sieling and I. Wegener, *Theoretical Comp. Sci.* **141** (1995), 283–310.]

123. Let $t = k - s = \bar{x}_1 + \cdots + \bar{x}_k$. There's a slate for every combination of s' 1s and t' 0s such that $s' + t' = w$, $s' \le s$, and $t' \le t$. The sum of $\binom{w}{s'} = \binom{w}{t'}$ over all such (s', t') is (97). (Notice furthermore that it equals 2^w if and only if $w \le \min(s,t)$.)

124. Let $m = n-k$. Each slate $[r_0,\ldots,r_m]$ corresponds to a function of $(x_{k+1},\ldots,x_n)$, whose truth table is a bead except in four cases: (i) $[0,\ldots,0] = 0$; (ii) $[1,\ldots,1] = 1$; (iii) $[0, x_n, 1] = x_n$ (which doesn't depend on x_{n-1}); (iv) $[1,\ldots,1,x_{k+1},0,\ldots,0]$, where there are p 1s so that $x_{k+1} = r_p$, is $S_{<p}(x_{k+2},\ldots,x_n)$.

The following polynomial-time algorithm computes $q_k = q$ and $b_k = q - q'$ by counting all slates. A subtle aspect arises when the entries of $[r_0,\ldots,r_m]$ are all 0 or 1, because such slates can occur for different values of s; we don't want to count them twice. The solution is to maintain four sets

$$C_{ab} = \{r_1 + \cdots + r_{m-1} \mid r_0 = a \text{ and } r_m = b \text{ in some slate}\}.$$

The value of 0π should be artificially set to $n+1$, not 0. Assume that $0 \le k < n$.

H1. [Initialize.] Set $m \leftarrow n-k$, $q \leftarrow q' \leftarrow s \leftarrow 0$, $C_{00} \leftarrow C_{01} \leftarrow C_{10} \leftarrow C_{11} \leftarrow \emptyset$.

H2. [Find v and w.] Set $v = \sum_{j=1}^{m-1}[(s+j)\pi \le k]$ and $w \leftarrow v + [s\pi \le k] + [(s+m)\pi \le k]$. If $v = m-1$, go to step H5.

H3. [Check for nonbeads.] Set $p \leftarrow -1$. If $v \ne m-2$, go to H4. Otherwise, if $m = 2$ and $(s+1)\pi = n$, set $p \leftarrow [(s+2)\pi \le k]$. Otherwise, if $w = m$ and $(s+j)\pi = k+1$ for some $j \in [1\,..\,m-1]$, set $p \leftarrow j$.

H4. [Add binomials.] For all s' and t' such that $s' + t' = w$, $0 \le s' \le s$, and $0 \le t' \le k-s$, set $q \leftarrow q + \binom{w}{s'}$ and $q' \leftarrow q' + [s' = p]$. Then go to H6.

H5. [Remember 0–1 slates.] Do the following for all s' and t' as in step H4: If $(s+m)\pi \le k$, set $C_{00} \leftarrow C_{00} \cup \{s'\}$ and $C_{01} \leftarrow C_{01} \cup \{s'-1\}$; otherwise set

$C_{01} \leftarrow C_{01} \cup \{s'\}$. If $s\pi \le k$ and $(s+m)\pi \le k$, set $C_{10} \leftarrow C_{10} \cup \{s'-1\}$ and $C_{11} \leftarrow C_{11} \cup \{s'-2\}$. If $s\pi \le k$ and $(s+m)\pi > k$, set $C_{11} \leftarrow C_{11} \cup \{s'-1\}$.

H6. [Loop on s.] If $s < k$, set $s \leftarrow s+1$ and return to H2.

H7. [Finish.] For $ab = 00$, 01, 10, and 11, set $q \leftarrow q + \binom{m-1}{r}$ for all $r \in C_{ab}$. Also set $q' \leftarrow q' + [0 \in C_{00}] + [m-1 \in C_{11}]$. ▮

125. Let $S(n,m) = \binom{n}{0} + \cdots + \binom{n}{m}$. There are $S(k+1-s, s) - 1$ nonconstant slates when $0 < s \le k$ and $s \ge 2k - n + 2$. The only other nonconstant slates, one each, arise when $s = 0$ and $k < (n-1)/2$. The constant slates are trickier to count, but there usually are $S(n+1-k, 2k+1-n)$ of them, appearing when $s = 2k-n$ or $s = 2k+1-n$. Taking account of nitpicky boundary conditions and nonbeads, we find

$$b_k \;=\; S(n-k, 2k-n) + \sum_{s=0}^{n-k} S(n-k-s,\; 2k+1-n+s) \\ \qquad - \min(k, n-k) - [n = 2k] - [3k \ge 2n-1] - 1$$

for $0 \le k < n$. Although $S(n,m)$ has no simple form, we can express $\sum_{k=0}^{n-1} b_k$ as $B_{n/2} + \sum_{0 \le m \le n-2k \le n} (n+3-m-2k)\binom{k}{m}$ + (small change) when n is even, and the same expression works when n is odd if we replace $B_{n/2}$ by $A_{(n+1)/2}$. The double sum can be reduced by summing first on k, since $(k+1)\binom{k}{m} = (m+1)\binom{k+1}{m+1}$:

$$\sum_{m=0}^{n}\biggl((n+5-m)\binom{\lfloor (n-m+2)/2\rfloor}{m+1} - (2m+2)\binom{\lfloor (n-m+4)/2\rfloor}{m+2}\biggr).$$

And the remaining sum can be tackled by breaking it into four parts, depending on whether m and/or n is odd. Generating functions are helpful: Let $A(z) = \sum_{k \le n}\binom{n-k}{2k} z^n$ and $B(z) = \sum_{k\le n}\binom{n-k}{2k+1} z^n$. Then $A(z) = 1 + \sum_{k<n}\binom{n-k-1}{2k} z^n + \sum_{k<n}\binom{n-k-1}{2k-1} z^n = 1 + \sum_{k\le n}\binom{n-k}{2k} z^{n+1} + \sum_{k \le n}\binom{n-k}{2k+1} z^{n+2} = 1 + zA(z) + z^2 B(z)$. A similar derivation proves that $B(z) = zB(z) + zA(z)$. Consequently

$$A(z) = \frac{1-z}{1-2z+z^2-z^3} = \frac{1-z^2}{1-z-z^2-z^4}, \qquad B(z) = \frac{z}{1-2z+z^2-z^3} = \frac{z+z^2}{1-z-z^2-z^4}.$$

Thus $A_n = 2A_{n-1} - A_{n-2} + A_{n-3} = A_{n-1} + A_{n-2} + A_{n-4}$ for $n \ge 4$, and B_n satisfies the same recurrences. In fact, we have $A_n = (3P_{2n+1} + 7P_{2n} - 2P_{2n-1})/23$ and $B_n = (3P_{2n+2} + 7P_{2n+1} - 2P_{2n})/23$, using the Perrin numbers of exercise 15.

Furthermore, setting $A^*(z) = \sum_{k\le n} k\binom{n-k}{2k} z^n$ and $B^*(z) = \sum_{k \le n} k\binom{n-k}{2k+1} z^n$, we find $A^*(z) = z^2 A(z)B(z)$ and $B^*(z) = z^2 B(z)^2$. Putting it all together now yields the remarkable exact formula

$$B(h_n) = \frac{56P_{n+2} + 77P_{n+1} + 47P_n}{23} - \left\lfloor \frac{n^2}{4} \right\rfloor - \left\lfloor \frac{7n+1}{3} \right\rfloor + (n \bmod 2) - 10.$$

Historical notes: The sequence $\langle A_n \rangle$ was apparently first studied by R. Austin and R. K. Guy, *Fibonacci Quarterly* **16** (1978), 84–86; it counts binary $x_1 \ldots x_{n-1}$ with each 1 next to another. The plastic constant χ was shown by C. L. Siegel to be the smallest "Pisot number," namely the smallest algebraic integer > 1 whose conjugates all lie inside the unit circle; see *Duke Math. J.* **11** (1944), 597–602.

126. When $n \ge 6$, we have $b_k = F_{\lfloor (k+7)/2 \rfloor} + F_{\lceil (k+7)/2 \rceil} - 4$ for $1 \le k < 2n/3$, and $b_k = 2^{n-k+2} - 6 - [k = n-2]$ for $4n/5 \le k < n$. But the main contributions to $B(h_n^\pi)$ come from the $2n/15$ profile elements between those two regions, and the methods of

binomial coefficient summation techniques
summation of binomial coefs
Generating functions
recurrences
Perrin numbers
Austin
Guy
plastic constant
Siegel
Pisot number

answer 125 can be extended to deal with them. The interesting sequences

$$A_n = \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n-2k}{3k}, \qquad B_n = \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n-2k}{3k+1}, \qquad C_n = \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n-2k}{3k+2}$$

have respective generating functions $(1-z)^2/p(z)$, $(1-z)z/p(z)$, $z^2/p(z)$, where $p(z) = (1-z)^3 - z^5$. These sequences arise in this problem because $\sum_{k=0}^{n} \binom{\lfloor n-2k/3 \rfloor}{k} = A_n + B_{n-1} + C_{n-2}$. They grow as α^n, where $\alpha \approx 1.7016$ is the real root of $(\alpha-1)^3\alpha^2 = 1$.

The BDD size can't be expressed in closed form, but there is a closed form in terms of $A_{\lfloor n/3 \rfloor}$ through $A_{\lfloor n/3 \rfloor+4}$ that is accurate to $O(2^{n/4}/\sqrt{n})$. Thus $B(h_n^\pi) = \Theta(\alpha^{n/3})$.

127. (The permutation $\pi = (3, 5, 7, \ldots, 2n'-1, n, n-1, n-2, \ldots, 2n', 2n'-2, \ldots, 4, 2, 1)$, $n' = \lfloor 2n/5 \rfloor$, turns out to be optimum for h_n when $12 < n \le 24$; but it gives $B(h_{100}^\pi) = 1{,}366{,}282{,}025$. Sifting does much better, as shown in answer 152; but still better permutations almost surely exist.)

128. Consider, for example, $M_3(x_4, x_2, x_7; x_6, x_1, x_8, x_3, x_9, x_{11}, x_5, x_{10})$. The first m variables $\{x_4, x_2, x_7\}$ are called "address bits"; the other 2^m are called "targets." The subfunctions corresponding to $x_1 = c_1, \ldots, x_k = c_k$ can be described by slates of options analogous to (96). For example, when $k = 2$ there are three slates $[x_6, 0, x_9, x_{11}]$, $[x_6, 1, x_9, x_{11}]$, $[x_8, x_3, x_5, x_{10}]$, where the result is obtained by using $(x_4x_7)_2$ to select the appropriate component. Only the third of these depends on x_3; hence $q_2 = 3$ and $b_2 = 1$. When $k = 6$ the slates are $[0,0]$, $[0,1]$, $[1,0]$, $[1,1]$, $[x_8,0]$, $[x_8,1]$, $[x_9,x_{11}]$, $[0,x_{10}]$, and $[1,x_{10}]$, with components selected by x_7; hence $q_6 = 9$ and $b_6 = 7$.

In general, if the variables $\{x_1, \ldots, x_k\}$ include a address bits and t targets, the slates will have $A = 2^{m-a}$ entries. Divide the set of all 2^m targets into 2^a subsets, depending on the known address bits, and suppose s_j of those subsets contain j known targets. (Thus $s_0 + s_1 + \cdots + s_A = 2^a$ and $s_1 + 2s_2 + \cdots + As_A = t$. We have $(s_0, \ldots, s_4) = (1,1,0,0,0)$ when $k = 2$ and $a = t = 1$ in the example above; and $(s_0, s_1, s_2) = (1,2,1)$ when $k = 6$, $a = 2$, $t = 4$.) Then the total number of slates, q_k, is $2^0 s_0 + 2^1 s_1 + \cdots + 2^{A-1}s_{A-1} + 2^A[s_A > 0]$. If x_{k+1} is an address bit, the number b_k of slates that depend on x_{k+1} is $q_k - 2^{A/2}[s_A > 0]$. Otherwise $b_k = 2^c$, where c is the number of constants that appear in the slates containing target x_{k+1}.

129. (Solution by M. Sauerhoff; see I. Wegener, *Branching Programs* (2000), Theorem 6.2.13.) Since $P_m(x_1, \ldots, x_{m^2}) = Q_m(x_1, \ldots, x_{m^2}) \wedge S_m(x_1, \ldots, x_{m^2})$ and $B(S_m) = m^3 + 2$, we have $B(P_m^\pi) \le (m^3+2)B(Q_m^\pi)$. Apply Theorem K.

(A stronger lower bound should be possible, because Q_m seems to have *larger* BDDs than P_m. For example, when $m = 5$ the permutation $(1\pi, \ldots, 25\pi) = (3, 1, 5, 7, 9, 2, 4, 6, 8, 10, 11, 12, 13, 14, 15, 16, 20, 23, 17, 21, 19, 18, 22, 24, 25)$ is optimum for Q_5; but $B(Q_5^\pi) = 535$, while $B(P_5) = 229$.)

130. (a) Each path that starts at the root of the BDD and takes s HI branches and t LO branches defines a subfunction that corresponds to graphs in which s adjacencies are forced and t are forbidden. We shall show that these $\binom{s+t}{s}$ subfunctions are distinct.

If subfunctions g and h correspond to different paths, we can find k vertices W with the following properties: (i) W contains vertices w and w' with $w \mathrel{—} w'$ forced in g and forbidden in h. (ii) No adjacencies between vertices of W are forced in h or forbidden in g. (iii) If $u \in W$ and $v \notin W$ and $u \mathrel{—} v$ is forced in h, then $u = w$ or $u = w'$. (These conditions make at most $2s + t = m - k$ vertices ineligible to be in W.)

We can set the remaining variables so that $u \mathrel{—} v$ if and only if $\{u, v\} \subseteq W$, whenever adjacency is neither forced nor forbidden. This assignment makes $g = 1$, $h = 0$.

(b) Consider the subfunction of $C_{m,\lceil m/2\rceil}$ in which vertices $\{1,\ldots,k\}$ are required to be isolated, but $u \mathrel{—} v$ whenever $k < u \le \lceil m/2\rceil < v \le m$. Then a k-clique on the $\lfloor m/2\rfloor$ vertices $\{\lceil m/2\rceil+1,\ldots,m\}$ is equivalent to an $\lceil m/2\rceil$-clique on $\{1,\ldots,m\}$. In other words, this subfunction of $C_{m,\lceil m/2\rceil}$ is $C_{\lfloor m/2\rfloor,k}$.

Now chose $k \approx \sqrt{m/3}$ and apply (a). [I. Wegener, *JACM* **35** (1988), 461–471.]

131. (a) The profile can be shown to be $(1,\, 1,\, 2,\, 4,\, \ldots,\, 2^{q-1},\, (p-2)\times(2^q-1,\ q\times 2^{q-1}),\, 2^q-1,\, 2^{q-1},\, \ldots,\, 4,\, 2,\, 1,\, 2)$, where $r\times b$ denotes the r-fold repetition of b. Hence the total size is $(pq+2p-2q+2)2^{q-1}-p+2$.

(b) With the ordering $x_1,\, x_2,\, \ldots,\, x_p,\, y_{11},\, y_{21},\, \ldots,\, y_{p1},\, \ldots,\, y_{1q},\, y_{2q},\, \ldots,\, y_{pq}$, the profile comes to $(1,\, 2,\, 4,\, \ldots,\, 2^{p-1},\, (q-1)p\times(2^{p-1}),\, 2^{p-1},\, \ldots,\, 4,\, 2,\, 1,\, 2)$, making the total size $(pq-p+4)2^{p-1}$.

(c) Suppose exactly $m = \lfloor \min(p,q)/2\rfloor$ x's occur among the first k variables in some ordering; we may assume that they are $\{x_1,\ldots,x_m\}$. Consider the 2^m paths in the QDD for C such that $x_j = \bar{x}_{m+j}$ for $1 \le j \le p-m$ and $y_{ij} = [i=j \text{ or } i=j+m \text{ or } j>m]$. These paths must pass through distinct nodes on level k. Hence $q_k \ge 2^m$; use (85). [See M. Nikolskaia and L. Nikolskaia, *Theor. Comp. Sci.* **255** (2001), 615–625.]

Optimum orderings for $(p,q) = (4,4)$, $(4,5)$, and $(5,4)$, via exercise 138, are:

$$x_1y_{11}x_2y_{21}x_3y_{31}y_{41}y_{12}y_{22}y_{32}y_{42}y_{13}y_{23}y_{33}y_{43}y_{14}y_{24}y_{34}y_{44}x_4 \text{ (size 108)};$$
$$x_1y_{11}x_2y_{21}x_2y_{31}y_{41}y_{12}y_{22}y_{32}y_{42}y_{13}y_{23}y_{33}y_{43}y_{14}y_{24}y_{34}y_{44}y_{15}y_{25}y_{35}y_{45}x_4 \text{ (size 140)};$$
$$x_1y_{11}x_2y_{21}y_{12}y_{22}y_{13}y_{23}y_{14}y_{24}x_3y_{31}y_{32}y_{33}y_{34}x_2y_{41}y_{42}y_{51}y_{52}y_{43}y_{53}y_{44}y_{54}x_5 \text{ (size 167)}.$$

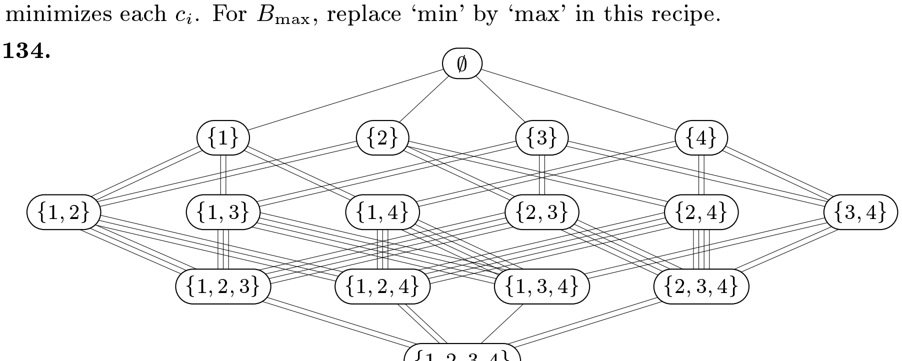
132. There are 616,126 essentially different classes of 5-variable functions, by Table 7.1.1–5. The maximum $B_{\min}(f)$, 17, is attained by 38 of those classes. Three classes have the property that $B(f^\pi) = 17$ for *all* permutations π; one such example, $((x_2 \oplus x_4 \oplus (x_1 \wedge (x_3 \vee \bar{x}_4))) \wedge ((x_2 \oplus x_5) \vee (x_3 \oplus x_4))) \oplus (x_5 \wedge (x_3 \oplus (x_1 \vee \bar{x}_2)))$, has the interesting symmetries $f(x_1,x_2,x_3,x_4,x_5) = f(\bar{x}_2,\bar{x}_3,\bar{x}_4,\bar{x}_1,\bar{x}_5) = f(x_2,\bar{x}_5,x_1,x_3,\bar{x}_4)$.

Incidentally, the maximum difference $B_{\max}(f) - B_{\min}(f) = 10$ occurs only in the "junction function" class $x_1?\ x_2{:}\ x_3?\ x_4{:}\ x_5$, when $B_{\min} = 7$ and $B_{\max} = 17$.

(When $n = 4$ there are 222 classes; and $B_{\min}(f) = 10$ in 25 of them, including S_2 and $S_{2,4}$. The class exemplified by truth table `16ad` is uniquely hardest, in the sense that $B_{\min}(f) = 10$ and most of the 24 permutations give $B(f^\pi) = 11$.)

133. Represent each subset $X \subseteq \{1,\ldots,n\}$ by the n-bit integer $i(X) = \sum_{x\in X} 2^{x-1}$, and let $b_{i(X),x}$ be the weight of the edge between X and $X\cup x$. Set $c_0 \leftarrow 0$, and for $1 \le i < 2^n$ set $c_i \leftarrow \min\{c_{i\oplus j} + b_{i\oplus j,x} \mid j = 2^{x-1} \text{ and } i\,\&\,j \ne 0\}$. Then $B_{\min}(f) = c_{2^n-1} + 2$, and an optimum ordering can be found by remembering which $x = x(i)$ minimizes each c_i. For $B_{\max}$, replace 'min' by 'max' in this recipe.

134.



Wegener
Nikolskaia
Nikolskaia
symmetries
junction function
symmetric functions
four-variable functions

The maximum profile, $(1,2,4,2,2)$, occurs on paths such as $\emptyset \to \{2\} \to \{2,3\} \to \{2,3,4\} \to \{1,2,3,4\}$. The minimum profile, $(1,2,2,1,2)$, occurs only on the paths $\emptyset \to (\{3\}$ or $\{4\}) \to \{3,4\} \to \{1,3,4\} \to \{1,2,3,4\}$. (Five of the 24 possible paths have the profile $(1,2,3,2,2)$ and are unimprovable by sifting on any variable.)

sifting
recurrences
master profile chart
0–1 matrices
symmetric threshold function

135. Let $\theta_0 = 1$, $\theta_1 = x_1$, $\theta_2 = x_1 \wedge x_2$, and $\theta_n = x_n ?\, \theta_{n-1} \colon \theta_{n-3}$ for $n \geq 3$. One can prove that, when $n \geq 4$, $B(\theta_n^\pi) = n+2$ if and only if $(n\pi, \ldots, 1\pi) = (1, \ldots, n)$. The key fact is that if $k < n$ and $n \geq 5$, the subfunctions obtained by setting $x_k \leftarrow 0$ or $x_k \leftarrow 1$ are distinct, and they both depend on the variables $\{x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n\}$, except that the subfunction for $x_{n-1} \leftarrow 0$ does not depend on x_{n-2}. Thus the weights $\{x_k\} \to \{x_k, x_l\}$ in the master profile chart are 2 except when $k = n$ or $(k,l) = (n-1, n-2)$. Below $\{x_{n-1}, x_{n-2}\}$ there are three subfunctions, namely $x_n ?\, \theta_{n-4} \colon \theta_{n-3}$, $x_n ?\, \theta_{n-5} \colon \theta_{n-3}$, and θ_{n-3}; all of them depend on $\{x_1, \ldots, x_{n-3}\}$, and two of them on x_n.

136. Let $n = 2n' - 1$ and $m = 2m' - 1$. The inputs form an $m \times n$ matrix, and we're computing the median of m row-medians. Let V_i be the variables in row i. If X is a subset of the mn variables, let $X_i = X \cap V_i$ and $r_i = |X_i|$. Subfunctions of type $(s_1, \ldots, s_m)$ arise when exactly s_i elements of X_i are set to 1; these subfunctions are

$$\langle S_1 S_2 \ldots S_m \rangle, \qquad \text{where } S_i = S_{\geq n' - s_i}(V_i \backslash X_i) \text{ and } 0 \leq s_i \leq r_i \text{ for } 1 \leq i \leq m.$$

When $x \notin X$, we want to count how many of these subfunctions depend on x. By symmetry we may assume that $x = x_{mn}$. Notice that the symmetric threshold function $S_{\geq t}(x_1, \ldots, x_n)$ equals 0 if $t > n$, or 1 if $t \leq 0$; it depends on all n variables if $1 \leq t \leq n$. In particular, S_m depends on x for exactly $r_m \$ n = \min(r_m + 1, n - r_m)$ choices of s_m.

Let $a_j = \sum_{i=1}^{m-1} [r_i = j]$ for $0 \leq j \leq n$. Then a_n of the functions $\{S_1, \ldots, S_{m-1}\}$ are constant, and $a_{n-1} + \cdots + a_{n'}$ of them might or might not be constant. Choosing c_i to be nonconstant gives us $(r_m \$ n)((a_n + a_{n-1} + \cdots + a_{n'} - c_{n-1} - \cdots - c_{n'}) \$ m)$ times

$$\binom{a_{n-1}}{c_{n-1}} \cdots \binom{a_{n'}}{c_{n'}} 1^{a_0} 2^{a_1} \ldots (n')^{a_{n'-1}} (n'-1)^{c_{n'}} (n'-2)^{c_{n'+1}} \ldots 1^{c_{n-1}}$$

distinct subfunctions that depend on x. Summing over $\{c_{n-1}, \ldots, c_{n'}\}$ gives the answer.

When variables have the natural row-by-row order, these formulas apply with $r_m = k \bmod n$, $a_n = \lfloor k/n \rfloor$, $a_0 = m - 1 - a_n$. The profile element b_k for $0 \leq k < mn$ is therefore $(\lfloor k/n \rfloor \$ m)((k \bmod n) \$ n)$, and we have $\sum_{k=0}^{mn} b_k = (m'n')^2 + 2$. This ordering is optimum, although no easy proof is apparent; for example, some orderings can decrease b_{n+2} or b_{2n-2} from 4 to 3 while increasing b_k for other k.

Every path from top to bottom of the master chart can be represented as $\alpha_0 \to \alpha_1 \to \cdots \to \alpha_{mn}$, where each α_j is a string $r_{j1} \ldots r_{jm}$ with $0 \leq r_{j1} \leq \cdots \leq r_{jm} \leq n$, $r_{j1} + \cdots + r_{jm} = j$, one coordinate increasing at each step. For example, one path when $m = 5$ and $n = 3$ is $00000 \to 00001 \to 00011 \to 00111 \to 00112 \to 00122 \to 00123 \to 01123 \to 11123 \to 11223 \to 12223 \to 12233 \to 12333 \to 22333 \to 23333 \to 33333$. We can convert this path to the "natural" path by a series of steps that don't increase the total edge weight, as follows: In the initial segment up to the first time $r_{jm} = n$, do all transitions on the rightmost coordinate first. (Thus the first steps of the example path would become $00000 \to 00001 \to 00002 \to 00003 \to 00013 \to 00113 \to 00123$.) Then in the final segment after the last time $r_{j1} = 0$, do all transitions on the leftmost coordinate last. (The final steps would thereby become $01123 \to 01223 \to 02223 \to 02233 \to 02333 \to 03333 \to 13333 \to 23333 \to 33333$.) Then, after the first n steps, normalize the second-last coordinates in a similar fashion ($00003 \to 00013 \to 00023 \to 00033 \to 00133 \to 01133 \to 01233 \to 02233$); and before the last n steps, normalize the second coordinates ($00133 \to 00233 \to 00333 \to 01333 \to 02333 \to 03333$). Et cetera.

[This back-and-forth proof technique was inspired by the paper of Bollig and Wegener cited below. Can every nonoptimal ordering be improved by merely sifting?]

Bollig
Wegener
sifting
clique
Bollig
Wegener
Sieling
bucket sorting

137. If we add a clique of c new vertices and $\binom{c}{2}$ new edges, the cost of the optimum arrangement increases by $\binom{c+1}{3}$. So we may assume that the given graph has m edges and n vertices $\{1,\ldots,n\}$, where m and n are odd and sufficiently large. The corresponding function f, which depends on $mn+m+1$ variables x_{ij} and s_k for $1 \le i \le m$, $1 \le j \le n$, and $0 \le k \le m$, is $J(s_0, s_1, \ldots, s_m; h, g_1, \ldots, g_m)$, where $g_i = (x_{iu_i} \oplus x_{iv_i}) \wedge \bigwedge\{x_{iw} \mid w \notin \{u_i, v_i\}\}$ when the ith edge is u_i — v_i, and where $h = \langle\langle x_{11} \ldots x_{m1}\rangle \ldots \langle x_{1n} \ldots x_{mn}\rangle\rangle$ is the transpose of the function in exercise 136.

One can show that $B_{\min}(f) = \min_\pi \sum_{u - v} |u\pi - v\pi| + (\frac{m+1}{2})^2(\frac{n+1}{2})^2 + mn + m + 2$; the optimum ordering uses $(\frac{m+1}{2})^2(\frac{n+1}{2})^2$ nodes for h, $n + |u_i\pi - v_i\pi|$ nodes for g_i, one node for each s_k, and two sink nodes, minus one node that is shared between h and some g_i. [See B. Bollig and I. Wegener, *IEEE Trans.* **C-45** (1996), 993–1002. D. Sieling, in *J. Computer and System Sci.* **74** (2008), 394–403, has proved that $B_{\min}(f)$ can't be approximated within a constant factor in polynomial time, unless P = NP.]

138. (a) Let $X_k = \{x_1, \ldots, x_k\}$. The QDD nodes at depth k represent the subfunctions that can arise when constants replace the variables of X_k. We can add an n-bit field DEP to each node, to specify exactly which variables of $X_n \setminus X_k$ it depends on. For example, the QDD for f in (92) has the following subfunctions and DEPs:

depth 0: 0011001001110010 [1111];
depth 1: 00110010 [0111], 01110010 [0111];
depth 2: 0010 [0011], 0011 [0010], 0111 [0011];
depth 3: 00 [0000], 01 [0001], 10 [0001], 11 [0000].

An examination of all DEP fields at depth k tells us the master profile weights between X_k and $X_k \cup x_l$, for $0 \le k < l \le n$.

(b) Represent the nodes at depth k as triples $N_{kp} = (l_{kp}, h_{kp}, d_{kp})$ for $0 \le p < q_k$, where (l_{kp}, h_{kp}) are the (LO, HI) pointers and d_{kp} records the DEP bits. If $k < n$, these nodes branch on x_{k+1}, so we have $0 \le l_{kp}, h_{kp} < q_{k+1}$; but if $k = n$, we have $l_{n0} = h_{n0} = 0$ and $l_{n1} = h_{n1} = 1$ to represent $\boxed{\bot}$ and $\boxed{\top}$. We define $d_{kp} = \sum\{2^{t-k-1} \mid N_{kp}$ depends on $x_t\}$; hence $0 \le d_{kp} < 2^{n-k}$. For example, the QDD (82) is equivalent to $N_{00} = (0,1,7)$; $N_{10} = (0,1,3)$, $N_{11} = (1,2,3)$; $N_{20} = (0,0,0)$, $N_{21} = (0,1,1)$, $N_{22} = (1,1,0)$; $N_{30} = (0,0,0)$, $N_{31} = (1,1,0)$.

To jump up from depth b to depth a, we essentially make two copies of the nodes at depths $b-1$, $b-2$, ..., a, one for the case $x_{b+1} = 0$ and one for the case $x_{b+1} = 1$. Those copies are moved down to depths b, $b-1$, ..., $a+1$, and reduced to eliminate duplicates. Then every original node at depth a is replaced by a node that branches on x_{b+1}; its LO and HI fields point respectively to the 0-copy and the 1-copy of the original.

This process involves some simple (but cool) list processing to update DEPs while bucket sorting: Nodes are unpacked into a work area consisting of auxiliary arrays r, s, t, u, and v, initially zero. Instead of using l_{kp} and h_{kp} for LO and HI, we store HI in cell u_p of the work area, and we let v_p link to the previous node (if any) with the same LO field; furthermore we make s_l point to the last node (if any) for which LO $= l$. The algorithm below uses UNPACK(p, l, h) as an abbreviation for "$u_p \leftarrow h$, $v_p \leftarrow s_l$, $s_l \leftarrow p+1$."

When nodes of depth k have been unpacked in this way to arrays s, u, and v, the following subroutine ELIM(k) packs them back into the main QDD structure with duplicates eliminated. It also sets r_p to the new address of node p.

hidden weighted bit
Friedman
Supowit
truth tables

E1. [Loop on l.] Set $q \leftarrow 0$ and $t_h \leftarrow 0$ for $0 \le h < q_{k+1}$. Do step E2 for $0 \le l < q_{k+1}$. Then set $q_k \leftarrow q$ and terminate.

E2. [Loop on p.] Set $p \leftarrow s_l$ and $s_l \leftarrow 0$. While $p > 0$, do step E3 and set $p \leftarrow v_{p-1}$. Then resume step E1.

E3. [Pack node $p-1$.] Set $h \leftarrow u_{p-1}$. (The unpacked node has $(\mathrm{LO}, \mathrm{HI}) = (l, h)$.) If $t_h \ne 0$ and $l_{k(t_h-1)} = l$, set $r_{p-1} \leftarrow t_h - 1$. Otherwise set $l_{kq} \leftarrow l$, $h_{kq} \leftarrow h$, $d_{kq} \leftarrow ((d_{(k+1)l} \,|\, d_{(k+1)h}) \ll 1) + [l \ne h]$, $r_{p-1} \leftarrow q$, $q \leftarrow q+1$, $t_h \leftarrow q$. Resume step E2. ▮

We can now use ELIM to jump up from b to a. (i) For $k = b-1, b-2, \ldots, a$, do the following steps: For $0 \le p < q_k$, set $l \leftarrow l_{kp}$, $h \leftarrow h_{kp}$; if $k = b-1$, $\mathrm{UNPACK}(2p, l_{bl}, h_{bl})$ and $\mathrm{UNPACK}(2p+1, l_{bh}, h_{bh})$, otherwise $\mathrm{UNPACK}(2p, r_{2l}, r_{2h})$ and $\mathrm{UNPACK}(2p+1, r_{2l+1}, r_{2h+1})$ (thereby making two copies of N_{kp} in the work area). Then $\mathrm{ELIM}(k+1)$. (ii) For $0 \le p < q_a$, $\mathrm{UNPACK}(p, r_{2p}, r_{2p+1})$. Then $\mathrm{ELIM}(a)$. (iii) If $a > 0$, set $l \leftarrow l_{(a-1)p}$, $h \leftarrow h_{(a-1)p}$, $l_{(a-1)p} \leftarrow r_l$, $h_{(a-1)p} \leftarrow r_h$, for $0 \le p < q_{a-1}$.

This jump-up procedure garbles the DEP fields above depth a, because the variables have been reordered. But we'll use it only when those fields are no longer needed.

(c) By induction, the first 2^{n-2} steps account for all subsets that do not contain n; then comes a jump-up from $n-1$ to 0, and the remaining steps account for all subsets that do contain n.

(d) Start by setting $y_k \leftarrow k$ and $w_k \leftarrow 2^k - 1$ for $0 \le k < n$. In the following algorithm, the y array represents the current variable ordering, and the bitmap $w_k = \sum\{2^{y_j} \mid 0 \le j < k\}$ represents the set of variables on the top k levels.

We augment the subroutine $\mathrm{ELIM}(k)$ so that it also computes the desired edge weights of the master profile: Counters c_j are initially 0 for $0 \le j < n-k$; after setting d_{kq} in step E3, we set $c_j \leftarrow c_j + 1$ for each j such that $2^j \subseteq d_{kq}$; finally we set $b_{w_k, y_{k+j}+1} \leftarrow c_j$ for $0 \le j < n-k$, using the notation of answer 133. [To speed this up, we could count bytes not bits, increasing $c_{j,(d_{kq} \gg 8j)\&\#\mathtt{ff}}$ by 1 for $0 \le j < (n-k)/8$.]

We initialize the DEP fields by doing the following for $k = n-1, n-2, \ldots, 0$: $\mathrm{UNPACK}(p, l_{kp}, h_{kp})$ for $0 \le p < q_k$; $\mathrm{ELIM}(k)$; if $k > 0$, set $l \leftarrow l_{(k-1)p}$, $h \leftarrow h_{(k-1)p}$, $l_{(k-1)p} \leftarrow r_l$, and $h_{(k-1)p} \leftarrow r_h$, for $0 \le p < q_{k-1}$.

The main loop of the algorithm now does the following for $1 \le i < 2^{n-1}$: Set $a \leftarrow \nu i - 1$ and $b \leftarrow \nu i + \rho i$. Set $(y_a, \ldots, y_b) \leftarrow (y_b, y_a, \ldots, y_{b-1})$ and $(w_{a+1}, \ldots, w_b) \leftarrow (2^{y_b} + w_a, \ldots, 2^{y_b} + w_{b-1})$. Jump up from b to a with the procedure of part (b); but use the original (non-augmented) ELIM routine for $\mathrm{ELIM}(a)$ in step (ii).

(e) The space required for nodes at depth k is at most $Q_k = \min(2^k, 2^{2^{n-k}})$; we also need space for $2\max(Q_1, \ldots, Q_n)$ elements in arrays r, u, v, plus $\max(Q_1, \ldots, Q_n)$ elements in arrays s and t. So the total is dominated by $O(2^n n)$ for the outputs $b_{w,x}$.

Subroutine $\mathrm{ELIM}(k)$ is called $\binom{n}{k}$ times in augmented form, for $0 \le k < n$, and $\binom{n-1}{k+1}$ times non-augmented. Its running time in either case is $O(q_k(n-k))$. Thus the total comes to $O(\sum_k \binom{n}{k} 2^k (n-k)) = O(3^n n)$, and it will be substantially less if the QDD never gets large. (For example, it's $O((1+\sqrt{2})^n n)$ for the function h_n.)

[The first exact algorithm to determine optimum variable ordering in a BDD was introduced by S. J. Friedman and K. J. Supowit, *IEEE Trans.* **C-39** (1990), 710–713. They used extended truth tables instead of QDDs, obtaining a method for $m = 1$ that required $\Theta(3^n/\sqrt{n})$ space and $\Theta(3^n n^2)$ time, improvable to $\Theta(3^n n)$.]

139. The same algorithm applies, almost unchanged: Consider all QDD nodes that branch on x_a to be at level 0, and all nodes that branch on x_{b+1} to be sinks. Thus we do 2^{b-a} jump-ups, not 2^{n-1}. (The algorithm doesn't rely on the assumptions that $q_0 = 1$ and $q_n = 2$, except in the space and time analyses of part (e).)

140. We can find shortest paths in a network without knowing the network in advance, by generating vertices and arcs "on the fly" as needed. Section 7.3 points out that the distance $d(X,Y)$ of each arc $X \to Y$ can be changed to $d'(X,Y) = d(X,Y) - l(X) + l(Y)$ for any function $l(X)$, without changing the shortest paths. If the revised distances d' are nonnegative, $l(X)$ is a lower bound on the distance from X to the goal; the trick is to find a good lower bound that focuses the search yet isn't difficult to compute.

If $|X| = l$, and if a QDD for f with X on its top l levels has q nonconstant nodes on the next level, then $l(X) = \max(q, n-l)$ is a suitable lower bound for the $B_{\min}$ problem. [See R. Drechsler, N. Drechsler, and W. Günther, *ACM/IEEE Conf. Design Automation* **35** (1998), 200–205.] However, a stronger lower bound is needed to make this approach competitive with the algorithm of exercise 138, unless f has a relatively short BDD that cannot be attained in very many ways.

141. False. Consider $g(x_1 \vee \cdots \vee x_6,\ x_7 \vee \cdots \vee x_{12},\ (x_{13} \vee \cdots \vee x_{16}) \oplus x_{18},\ x_{17},\ x_{19} \vee \cdots \vee x_{22})$, where $g(y_1, \ldots, y_5) = ((((\bar{y}_1 \vee y_5) \wedge y_4) \oplus y_3) \wedge ((y_1 \wedge y_2) \oplus y_4 \oplus y_5)) \oplus y_5$. Then $B(g) = 40 = B_{\min}(g)$ can't be achieved with $\{x_{13}, \ldots, x_{16}, x_{18}\}$ consecutive. [M. Teslenko, A. Martinelli, and E. Dubrova, *IEEE Trans.* **C-54** (2005), 236–237.]

142. (a) Suppose m is odd. The subfunctions that arise after $(x_1, \ldots, x_{m+1})$ are known are $[w_{m+2}x_{m+2} + \cdots + w_n x_n > 2^{m-1}m - 2^{m-2} - t]$, where $0 \le t \le 2^m$. The subcases $x_{m+2} + \cdots + x_n = (m-1)/2$ show that at least $\binom{m-1}{(m-1)/2}$ of these subfunctions differ.

But organ-pipe order, $\langle x_1 x_2^{2^m-1} x_3^1 x_4^{2^m-2} x_5^2 \ldots x_{n-2}^{2^m-2^{m-2}} x_{n-1}^{2^{m-2}} x_n^{2^{m-1}} \rangle$, is much better: Let $t_k = x_1 + (2^m-1)x_2 + x_3 + \cdots + (2^m - 2^{k-1})x_{2k} + 2^{k-1}x_{2k+1}$, for $1 \le k < m-1$. The remaining subfunction depends on at most $2k+2$ different values, $\lceil t_k/2^k \rceil$.

(b) Let $n = 1 + 4m^2$. The variables are x_0 and x_{ij} for $0 \le i,j < 2m$; the weights are $w_0 = 1$ and $w_{ij} = 2^i + 2^{2m+1+j}m$. Let X_l be the first l variables in some ordering, and suppose X_l includes elements in i_l rows and j_l columns of the matrix (x_{ij}). If $\max(i_l, j_l) = m$, we will prove that $q_l \ge 2^m$; hence $B(f) > 2^m$ by (85).

Let I and J be subsets of $\{1, \ldots, 2m\}$ with $|I| = |J| = m$ and $X_l \subseteq x_0 \cup \{x_{ij} \mid i \in I, j \in J\}$; let I' and J' be the complementary subsets. Choose m elements $X' \subseteq X_l \setminus x_0$, in different rows (or, if $i_l < m$, in different columns). Consider 2^m paths in the QDD defined as follows: $x_0 = 0$, and $x_{ij} = 0$ if $x_{ij} \in X_l \setminus X'$; also $x_{i'j} = x_{ij'} = \bar{x}_{i'j'} = \bar{x}_{ij}$ for $i \in I$, $j \in J$, where $i \leftrightarrow i'$ and $j \leftrightarrow j'$ are matchings between $I \leftrightarrow I'$ and $J \leftrightarrow J'$. Then there are 2^m distinct values $t = \sum_{i \in I, j \in J} w_{ij}x_{ij}$; but $\sum_{0 \le i,j < 2m} w_{ij}x_{ij} = (2^{2m}-1)(1+2^{2m+1}m)$ on each path. The paths must pass through distinct nodes on level l. Otherwise, if $t \ne t'$, one of the lower subpaths would lead to $\boxed{\bot}$, the other to $\boxed{\top}$.

[These results are due to K. Hosaka, Y. Takenaga, T. Kaneda, and S. Yajima, *Theoretical Comp. Sci.* **180** (1997), 47–60, who also proved that $|Q(f) - Q(f^R)| < n$. Do self-dual threshold functions always satisfy also $|B(f) - B(f^R)| < n$?]

143. In fact, the algorithm of exercises 133 and 138 proves that organ-pipe order is best for these weights: (1, 1023, 1, 1022, 2, 1020, 4, 1016, 8, 1008, 16, 992, 32, 960, 64, 896, 128, 768, 256, 512) gives the profile (1, 2, 2, 4, 3, 6, 4, 8, 5, 10, 4, 8, 3, 6, 2, 4, 1, 2, 2, 1, 2) and $B(f) = 80$. The worst ordering, (1022, 896, 512, 64, 8, 1, 4, 32, 1008, 1020, 768, 992, 1016, 1023, 960, 256, 128, 16, 2, 1), makes $B(f) = 1913$.

(One might think that properties of binary notation are crucial to this example. But $\langle x_1 x_2 x_3^2 x_4^4 x_5^8 x_6^{16} x_7^{31} x_8^{60} x_9^{116} x_{10}^{224} x_{11}^{224} x_{12}^{448} x_{13}^{564} x_{14}^{620} x_{15}^{649} x_{16}^{664} x_{17}^{672} x_{18}^{676} x_{19}^{678} x_{20}^{679} \rangle$ is actually the same function, by exercise 7.1.1–103(!).)

144. $(5,7,7,10,6,9,5,4,2)$; the QDD-not-BDD nodes correspond to f_1, f_2, f_3, 0, 1.

shortest paths
Drechsler
Drechsler
Günther
Teslenko
Martinelli
Dubrova
organ-pipe order
Hosaka
Takenaga
Kaneda
Yajima
f^R
organ-pipe order

145. $B_{\min} = 31$ is attained in (36). The worst ordering for $(x_3x_2x_1x_0)_2 + (y_3y_2y_1y_0)_2$ is y_0, y_1, y_2, y_3, x_2, x_1, x_0, x_3, making $B_{\max} = 107$. Incidentally, the worst ordering for the 24 inputs of 12-bit addition, $(x_{11} \ldots x_0)_2 + (y_{11} \ldots y_0)_2$, turns out to be y_0, y_1, $\ldots$, y_{11}, x_{10}, x_8, x_6, x_4, x_3, x_5, x_2, x_7, x_1, x_9, x_0, x_{11}, yielding $B_{\max} = 39111$.

[B. Bollig, N. Range, and I. Wegener, *Lecture Notes in Comp. Sci.* **4910** (2008), 174–185, have proved that $B_{\min} = 9n - 5$ for addition of two n-bit numbers whenever $n > 1$, and also that $B_{\min}(M_m) = 2n - 2m + 1$ for the 2^m-way multiplexer.]

146. (a) Obviously $b_0 \le q_0$; and if $q_0 = b_0 + a_0$, then $b_1 \le 2b_0 + a_0 = b_0 + q_0$. Also $q_0 - b_0 = a_0 \le b_1 + q_2 \le q_2^2$, the number of strings of length 2 on a q_2-letter alphabet; similarly $b_0 + b_1 + q_2 \le (b_1 + q_2)^2$. (The same relations hold between q_k, q_{k+2}, b_k, and b_{k+1}.)

(b) Let the subfunctions at level 2 have truth tables α_j for $1 \le j \le q_2$, and use them to construct beads β_1, $\ldots$, β_{b_1} at level 1. Let $(\gamma_1, \ldots, \gamma_{q_2+b_1})$ be the truth tables $(\alpha_1\alpha_1, \ldots, \alpha_{q_2}\alpha_{q_2}, \beta_1, \ldots, \beta_{b_1})$. If $b_0 \le b_1/2$, let the functions at level 0 have truth tables $\{\beta_{2i-1}\beta_{2i} \mid 1 \le i \le b_0\} \cup \{\beta_j\beta_j \mid 2b_0 < j \le b_1\} \cup \{\gamma_j\gamma_j \mid 1 \le j \le b_0 + q_0 - b_1\}$. Otherwise it's not difficult to define b_0 beads that include all the β's, and use them at level 0 together with the nonbeads $\{\gamma_j\gamma_j \mid 1 \le j \le q_0 - b_0\}$.

147. Before doing any reordering, we clear the cache and collect all garbage. The following algorithm interchanges levels $(u) \leftrightarrow (v)$ when $v = u + 1$. It works by creating linked lists of solitary, tangled, and hidden nodes, pointed to by variables S, T, and H (initially Λ), using auxiliary `LINK` fields that can be borrowed temporarily from the hash-table algorithm of the unique lists as they are being rebuilt.

- **T1.** [Build S and T.] For each (u)-node p, set $q \leftarrow$ `LO`(p), $r \leftarrow$ `HI`(p), and delete p from its hash table. If `V`$(q) \ne v$ and `V`$(r) \ne v$ (p is solitary), set `LINK`$(p) \leftarrow S$ and $S \leftarrow p$. Otherwise (p is tangled), set `REF`$(q) \leftarrow$ `REF`$(q) - 1$, `REF`$(r) \leftarrow$ `REF`$(r) - 1$, `LINK`$(p) \leftarrow T$, and $T \leftarrow p$.
- **T2.** [Build H and move the visible nodes.] For each (v)-node p, set $q \leftarrow$ `LO`(p), $r \leftarrow$ `HI`(p), and delete p from its hash table. If `REF`$(p) = 0$ (p is hidden), set `REF`$(q) \leftarrow$ `REF`$(q) - 1$, `REF`$(r) \leftarrow$ `REF`$(r) - 1$, `LINK`$(p) \leftarrow H$, and $H \leftarrow p$; otherwise (p is visible) set `V`$(p) \leftarrow u$ and INSERT(u, p).
- **T3.** [Move the solitary nodes.] While $S \ne \Lambda$, set $p \leftarrow S$, $S \leftarrow$ `LINK`(p), `V`$(p) \leftarrow v$, and INSERT(v, p).
- **T4.** [Transmogrify the tangled nodes.] While $T \ne \Lambda$, set $p \leftarrow T$, $T \leftarrow$ `LINK`(p), and do the following: Set $q \leftarrow$ `LO`(p), $r \leftarrow$ `HI`(p). If `V`$(q) > v$, set $q_0 \leftarrow q_1 \leftarrow q$; otherwise set $q_0 \leftarrow$ `LO`(q) and $q_1 \leftarrow$ `HI`(q). If `V`$(r) > v$, set $r_0 \leftarrow r_1 \leftarrow r$; otherwise set $r_0 \leftarrow$ `LO`(r) and $r_1 \leftarrow$ `HI`(r). Then set `LO`$(p) \leftarrow$ UNIQUE(v, q_0, r_0), `HI`$(p) \leftarrow$ UNIQUE(v, q_1, r_1), and INSERT(u, p).
- **T5.** [Kill the hidden nodes.] While $H \ne \Lambda$, set $p \leftarrow H$, $H \leftarrow$ `LINK`(p), and recycle node p. (All of the remaining nodes are alive.) ▮

The subroutine INSERT(v, p) simply puts node p into x_v's unique table, using the key (`LO`(p), `HI`(p)); this key will not already be present. The subroutine UNIQUE in step T4 is like Algorithm U, but instead of using answer 82 it treats reference counts quite differently in steps U1 and U2: If U1 finds $p = q$, it *increases* `REF`(p) by 1; if U2 finds r, it simply sets `REF`$(r) \leftarrow$ `REF`$(r) + 1$.

Internally, the branch variables retain their natural order 1, 2, $\ldots$, n from top to bottom. Mapping tables ρ and π represent the current permutation from the external user's point of view, with $\rho = \pi^-$; thus the user's variable x_v appears on level $v\pi - 1$,

Bollig
Range
Wegener
2^m-way multiplexer
clear the cache
collect all garbage
linked lists
Transmogrify
UNIQUE

and node UNIQUE(v,p,q) on level $v-1$ represents the user's function $(\bar{x}_{v\rho}?\ p\colon q)$. To maintain these mappings, set $j \leftarrow u\rho$, $k \leftarrow v\rho$, $u\rho \leftarrow k$, $v\rho \leftarrow j$, $j\pi \leftarrow v$, $k\pi \leftarrow u$.

148. False. For example, consider six sinks and nine source functions, with extended truth tables 1156, 2256, 3356, 4456, 5611, 5622, 5633, 5644, 5656. Eight of the nodes are tangled and one is visible, but none are hidden or solitary. There are 16 newbies: 15, 16, 25, 26, 35, 36, 45, 46, 51, 61, 52, 62, 53, 63, 54, 64. So the swap takes 15 nodes into 31. (We can use the nodes of $B(x_3 \oplus x_4, x_3 \oplus \bar{x}_4)$ for the sinks.)

149. The successive profiles are bounded by $(b_0, b_1, \ldots, b_n)$, $(b_0+b_1, 2b_0, b_2, \ldots, b_n)$, $(b_0+b_1, 2b_0+b_2, 4b_0, b_3, \ldots, b_n)$, $\ldots$, $(2^0b_0+b_1, \ldots, 2^{k-2}b_0+b_{k-1}, 2^{k-1}b_0, b_k, \ldots, b_n)$.

Similarly, we also have $B(f_1^\pi, \ldots, f_m^\pi) \le B(f_1, \ldots, f_m) + 2(b_0+\cdots+b_{k-1})$ in addition to Theorem J$^+$, because swaps contribute at most $2b_{k-1}$, $2b_{k-2}$, $\ldots$, $2b_0$ new nodes.

150. We may assume that $m=1$, as in exercise 52. Suppose we want to jump x_k to the position that is jth in the ordering, where $j \ne k$. First compute the restrictions of f when $x_k=0$ and $x_k=1$ (see exercise 57); call them g and h. Then renumber the remaining variables: If $j<k$, change $(x_j, \ldots, x_{k-1})$ to $(x_{j+1}, \ldots, x_k)$; otherwise change $(x_{k+1}, \ldots, x_j)$ to $(x_k, \ldots, x_{j-1})$. Then compute $f \leftarrow (\bar{x}_j \wedge g) \vee (x_j \wedge h)$, using the linear-time variant of Algorithm S in exercise 72.

To show that this method has the desired running time, it suffices to prove the following: *Let $g(x_1, \ldots, x_n)$ and $h(x_1, \ldots, x_n)$ be functions such that $g(x)=1$ implies $x_j=0$ and $h(x)=1$ implies $x_j=1$. Then the meld $g \diamond h$ has at most twice as many nodes as $g \vee h$.* But this is almost obvious, when truth tables are considered: For example, if $n=3$ and $j=2$, the truth tables for g and h have the respective forms $ab00cd00$ and $00st00uv$. The beads β of $g \vee h$ on levels $<j$ correspond uniquely to the beads $\beta' \diamond \beta''$ of $g \diamond h$ on those levels, because $\beta = \beta' \vee \beta''$ can be "factored" in only one way by putting 0s in the appropriate places. And the beads β of $g \vee h$ on levels $\ge j$ correspond to at most two beads of $g \diamond h$, namely to $\beta \diamond \boxed{\bot}$ and/or $\boxed{\bot} \diamond \beta$.

[See P. Savický and I. Wegener, *Acta Informatica* **34** (1997), 245–256, Theorem 1.]

151. Set $t_k \leftarrow 0$ for $1 \le k \le n$, and make the swapping operation $x_{j-1} \leftrightarrow x_j$ also swap $t_{j-1} \leftrightarrow t_j$. Then set $k \leftarrow 1$ and do the following until $k > n$: If $t_k = 1$ set $k \leftarrow k+1$; otherwise set $t_k \leftarrow 1$ and sift x_k.

(This method repeatedly sifts on the topmost variable that hasn't yet been sifted. Researchers have tried fancier strategies, such as to sift the largest level first; but no such method has turned out to dominate the simple-minded approach proposed here.)

152. Applying Algorithm J as in answer 151 yields $B(h_{100}^\pi) = 1{,}382{,}685{,}050$ after 17,179 swaps, which is almost as good as the result of the "hand-tuned" permutation (95). Another sift brings the size down to 300,451,396; and further repetitions converge down to just 231,376,264 nodes, after a total of 232,951 swaps.

If the loops of steps J2 and J5 are aborted when $S > 1.05s$, the results are even better(!), although fewer swaps are made. The first sift reduces the size to 1,342,191,700, and iteration produces $B(h_{100}^\pi) = 208{,}478{,}228$ after 139,245 swaps, where π is the following permutation:

3	4	6	8	10	12	14	16	18	20	22	24	27	28	30	32	35	67	37	39
43	41	45	51	47	49	55	80	53	83	85	92	93	94	78	75	77	95	73	71
96	98	97	68	57	58	60	65	63	62	61	87	64	59	66	88	56	69	70	99
100	72	76	91	79	74	90	89	86	84	52	82	81	48	54	50	46	44	42	40
38	36	34	33	31	29	26	25	23	21	19	17	15	13	11	9	7	5	2	1

extended truth tables
tangled
visible
hidden
solitary
meld
Savický
Wegener

Incidentally, if we sift the variables h_{100} in order of profile size, so that x_{60} is sifted first, then x_{59}, x_{61}, x_{58}, x_{57}, x_{62}, x_{56}, etc. (wherever they currently happen to be), the resulting BDD turns out to have 2,196,768,534 nodes.

Simple "downhill swapping" instead of full sifting is of no use whatever for h_{100}: The $\binom{100}{2}$ swaps $x_1 \leftrightarrow x_2$, $x_3 \leftrightarrow x_1$, $x_3 \leftrightarrow x_2$, ..., $x_{100} \leftrightarrow x_1$, ..., $x_{100} \leftrightarrow x_{99}$ completely reverse the order of all variables without changing the BDD size at any step.

153. Each gate is easily synthesized using recursions like (55). About 1 megabyte of memory and 3.5 megamems of computation suffice to construct the entire BDD base of 8242 nodes. Using exercise 138 we may conclude that the ordering x_7, x_3, x_9, x_1, o_9, o_1, o_3, o_7, x_4, x_6, o_6, o_4, o_2, o_8, x_2, x_8, o_5, x_5 is optimum, and that $B_{\min}(y_1, \ldots, y_9) = 5308$.

Reordering of variables is *not* advisable for a problem such as this, since there are only 18 variables. For example, autosifting whenever the size doubles would require more than 100 megamems of work, just to reduce 8242 nodes to about 6400.

154. Yes: `CA` was moved between `ID` and `OR` at the last sifting step, and we can work backwards all the way to deduce that the first sift moved `ME` between `MA` and `RI`.

155. The author's best attempt for (a) is

```
ME NH VT MA CT RI NY DE NJ MD PA DC VA OH WV KY NC SC GA FL AL IN MI IA
IL MO TN AR MS TX LA CO WI KS SD ND NE OK WY MN ID MT NM AZ OR CA WA UT NV
```

giving $B(f_1^\pi) = 403$, $B(f_2^\pi) = 677$, $B(f_1^\pi, f_2^\pi) = 1073$; and for (b) the ordering

```
NH ME MA VT CT RI NY DE NJ MD PA VA DC OH WV KY TN NC SC GA FL AL IN MI
IL IA AR MO MS TX LA CO KS OK WI SD NE ND MN WY ID MT AZ NM UT OR CA WA NV
```

gives $B(f_1^\pi) = 352$, $B(f_2^\pi) = 702$, $B(f_1^\pi, f_2^\pi) = 1046$.

156. One might expect two "siftups" to be at least as good as a single sifting process that goes both up and down. But in fact, benchmark tests by R. Rudell show that siftup alone is definitely unsatisfactory. Occasional jump-downs are needed to compensate for variables that temporarily jump up, although their optimum final position lies below.

157. A careful study of answer 128 shows that we always improve the size when the first address bit that follows a target bit is jumped up past all targets. [But simple swaps are too weak. For example, $M_2(x_1, x_6; x_2, x_3, x_4, x_5)$ and $M_3(x_1, x_{10}, x_{11}; x_2, x_3, \ldots, x_9)$ are locally optimal under the swapping of $x_{j-1} \leftrightarrow x_j$ for any j.]

158. Consider first the case when $m = 1$ and $n = 3t - 1 \geq 5$. Then if $n\pi = k$, the number of nodes that branch on j is a_j if $j\pi < k$, b_j if $j\pi = k$, and a_{n+2-j} if $j\pi > k$, where

$$a_j = j - 3\max(j - 2t, 0), \qquad b_j = \min(j, t, n + 1 - j).$$

The cases with $\{x_1, \ldots, x_{n-1}\}$ consecutive are $k = 1$ and $B(f^\pi) = 3t^2 + 2$; $k = n$ and $B(f^\pi) = 3t^2 + 1$. But when $k = \lceil n/2 \rceil$ we have $B(f^\pi) = \lfloor 3t/2 \rfloor (\lceil 3t/2 \rceil - 1) + n - \lfloor t/2 \rfloor + 2$.

Similar calculations apply when $m > 1$: We have $B(f^\pi) > 6\binom{p/3}{2} + B(g^\pi)$ when π makes $\{x_1, \ldots, x_p\}$ consecutive, but $B(f^\pi) \approx 2\binom{p/2}{2} + \frac{p}{3}B(g^\pi)$ when π puts $\{x_{p+1}, \ldots, x_{p+m}\}$ in the middle. Since g is fixed, $pB(g^\pi) = O(n)$ as $n \to \infty$.

[If g is a function of the same kind, we obtain examples where symmetric variables within g are best split up, and so on. But no Boolean functions are known for which the optimum $B(f^\pi)$ is less than 3/4 of the best that is obtainable under the constraint that no blocks of symmetric variables are split. See D. Sieling, *Random Structures & Algorithms* **13** (1998), 49–70.]

159. The function is almost symmetric, so there are only nine possibilities. When the center element x is placed in position $(1, 2, \ldots, 9)$ from the top, the BDD size is respectively $(43, 43, 42, 39, 36, 33, 30, 28, 28)$.

autosifting
Knuth
Rudell
swaps
Sieling
almost symmetric

160. (a) Compute $\bigwedge_{i=0}^{9}\bigwedge_{j=0}^{9}(\neg L_{ij}(X))$, a Boolean function of 64 variables — for example, by applying COMPOSE to the relatively simple L function of exercise 159, 100 times. With the author's experimental programs, about 320 megamems and 35 megabytes are needed to find this BDD, which has 251,873 nodes with the normal ordering. Then Algorithm C quickly finds the desired answer: 21,929,490,122. (The number of 11×11 solutions, 5,530,201,631,127,973,447, can be found in the same way.)

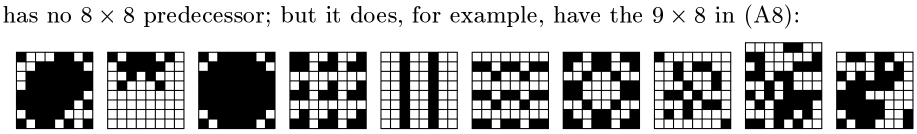
(b) The generating function is $1+64z+2016z^2+39740z^3+\cdots+80z^{45}+8z^{46}$, and Algorithm B rapidly finds the eight solutions of weight 46. Three of them are distinct under chessboard symmetry; the most symmetric solution is shown as (A0) below.

(c) The BDD for $\bigwedge_{i=1}^{8}\bigwedge_{j=1}^{8}(\neg L_{ij}(X))$ has 305,507 nodes and 21,942,036,750 solutions. So there must be 12,546,628 wild ones.

(d) Now the generating function is $40z^{14}+936z^{15}+10500z^{16}+\cdots+16z^{55}+z^{56}$; examples of weight 14 and 56 appear below as (A1) and (A2).

(e) Exactly 28 of weight 27 and 54 of weight 28, all tame; see (A3).

(f) There are respectively (26260, 5, 347, 0, 122216) solutions, found with about (228, 3, 32, 1, 283) megamems of calculation. Among the lightest and heaviest solutions to (1) are (A4) and (A5); the nicest solution to (2) is (A6); (A7) and (A9) solve (3) lightly and (5) heavily. Pattern (4), which is based on the binary representation of π, has no 8×8 predecessor; but it does, for example, have the 9×8 in (A8):



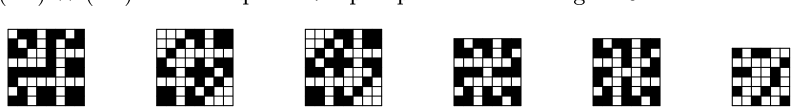
(A0) (A1) (A2) (A3) (A4) (A5) (A6) (A7) (A8) (A9)

161. (a) With the normal row-by-row ordering $(x_{11}, x_{12}, \ldots, x_{n(n-1)}, x_{nn})$, the BDD has 380,727 nodes and characterizes 4,782,725 solutions. The computational cost is about 2 gigamems, in 100 megabytes. (Similarly, the 29,305,144,137 still Lifes of size 10×10 can be enumerated with 14,492,923 nodes, after fewer than 50 gigamems.)

(b) This solution is essentially unique; see (B1) below. There's also a unique (and obvious) solution of weight 36.

(c) Now the BDD has 128 variables, with the ordering $(x_{11}, y_{11}, \ldots, x_{nn}, y_{nn})$. We could first set up BDDs for $[L(X)=Y]$ and $[L(Y)=X]$, then intersect them; but that turns out to be a bad idea, requiring some 36 million nodes even in the 7×7 case. Much better is to apply the constraints $L_{ij}(X)=y_{ij}$ and $L_{ij}(Y)=x_{ij}$ row by row, and also to add the lexicographic constraint $X<Y$ so that still Lifes are ruled out early. The computation can then be completed with about 20 gigamems and 1.6 gigabytes; there are 978,563 nodes and 582,769 solutions.

(d) Again the solution is unique, up to rotation; see the "spark plug" (B2) $\leftrightarrow$ (B3). (And (B4) $\leftrightarrow$ (B5) is the unique 7×7 flip-flop of constant weight 26. Life is astonishing.)



(B1) (B2) (B3) (B4) (B5) (B6)

162. Let $T(X)=[X \text{ is tame}]$ and $E_k(X)=[X \text{ escapes after } k \text{ steps}]$. We can compute the BDD for each E_k by using the recurrence

$$E_1(X)=\neg T(X); \qquad E_{k+1}(X)=\exists Y(T(X)\wedge[L(X)=Y]\wedge E_k(Y)).$$

Knuth
generating function
symmetry
π
spark plug
recurrence

(Here $\exists Y$ stands for $\exists y_{11}\,\exists y_{12}\,\cdots\exists y_{66}$. As noted in answer 103, this recurrence turns out to be much more efficient than the rule $E_{k+1}=T(X)\wedge E_k(L_{11}(X),\ldots,L_{66}(X))$, although the latter looks more "elegant.") The number of solutions, $|E_k|$, is found to be $(806544\cdot 2^{16},\ 657527179\cdot 2^4,\ 2105885159,\ 763710262,\ 331054880,\ 201618308,\ 126169394,\ 86820176,\ 63027572,\ 41338572,\ 30298840,\ 17474640,\ 9797472,\ 5258660,\ 3058696,\ 1416132,\ 523776,\ 204192,\ 176520,\ 62456,\ 13648,\ 2776,\ 2256,\ 440,\ 104,\ 0)$ for $k=(1,\ 2,\ \ldots,\ 26)$; thus $\sum_{k=1}^{25}|E_k|=67{,}166{,}017{,}379$ of the $2^{36}=68{,}719{,}476{,}736$ possible configurations eventually escape from the 6×6 cage. (One of the 104 procrastinators in E_{25} is shown in (B6) above.)

quantification
truth table

BDD techniques are excellent for this problem when k is small; for example, $B(E_1)=101$ and $B(E_2)=14441$. But E_k eventually becomes a complicated "nonlocal" function: The size peaks at $B(E_6)=28{,}696{,}866$, after which the number of solutions gets small enough to keep the size down. More than 80 million nodes are present in the formula $T(X)\wedge[L(X)=Y]\wedge E_5(Y)$ before quantification; this stretches memory limits. Indeed, the BDD for $\bigvee_{k=1}^{25}E_k(X)$ takes up more space than its 2^{33}-byte truth table. Therefore a "forward" method for this exercise would be preferable to the use of BDDs.

(Cages larger than 6×6 appear to be impossibly difficult, by *any* known method.)

163. Suppose first that $\circ$ is $\wedge$. We obtain the BDD for $f=g\wedge h$ by taking the BDD for g and replacing its $\boxed{\top}$ sink by the root of the BDD for h. To represent also $\bar f$, make a separate copy of the BDD for g, and use a BDD base for both h and $\bar h$; replace the $\boxed{\bot}$ in the copy by $\boxed{\top}$, and replace the $\boxed{\top}$ in the copy by the root of the BDD for $\bar h$. This decision diagram is reduced because h isn't constant.

Similarly, if $\circ$ is $\oplus$, we obtain a BDD for $f=g\oplus h$ (and possibly $\bar f$) from the BDD for g (and possibly $\bar g$) after replacing $\boxed{\bot}$ and $\boxed{\top}$ by the roots of BDDs for h and $\bar h$.

The other binary operations $\circ$ are essentially the same, because $B(f)=B(\bar f)$. For example, if $f=g\supset h=\overline{g\wedge\bar h}$, we have $B(f)=B(\bar f)=B(g)+B(\bar h)-2=B(g)+B(h)-2$.

164. Let $U_1(x_1)=V_1(x_1)=x_1$, $U_{n+1}(x_1,\ldots,x_{n+1})=x_1\oplus V_n(x_2,\ldots,x_{n+1})$, and $V_{n+1}(x_1,\ldots,x_{n+1})=U_n(x_1,\ldots,x_n)\wedge x_{n+1}$. Then one can show by induction that $B(f)\le B(U_n)=2^{\lceil(n+1)/2\rceil}+2^{\lfloor(n+1)/2\rfloor}-1$ for all read-once f, and also that we always have $B(f,\bar f)\le B(V_n,\overline{V_n})=2^{\lceil n/2\rceil+1}+2^{\lfloor n/2\rfloor+1}-2$. (But an optimum ordering reduces these sizes dramatically, to $B(U_n^\pi)=\lfloor\frac32 n+2\rfloor$ and $B(V_n^\pi,\overline{V_n}^\pi)=2n+2$.)

165. By induction, we prove also that $B(u_{2m},\bar u_{2m})=2^mF_{2m+3}+2$, $B(u_{2m+1},\bar u_{2m+1})=2^{m+1}F_{2m+3}+2$, $B(v_{2m},\bar v_{2m})=2^{m+1}F_{2m+1}+2$, $B(v_{2m+1},\bar v_{2m+1})=2^{m+1}F_{2m+3}+2$.

166. We may assume as in answer 163 that $\circ$ is either $\wedge$ or $\oplus$. By renumbering, we can also assume that $j\sigma=j$ for $1\le j\le n$, hence $f^\sigma=f$. Let $(b_0,\ldots,b_n)$ be the profile of f, and $(b_0',\ldots,b_n')$ the profile of $(f,\bar f)$; let $(c_{1\pi},\ldots,c_{(n+1)\pi})$ and $(c_{1\pi}',\ldots,c_{(n+1)\pi}')$ be the profiles of f^π and $(f^\pi,\bar f^\pi)$, where $(n+1)\pi=n+1$. Then $c_{j\pi}$ is the number of subfunctions of $f^\pi=g^\pi\circ h^\pi$ that depend on $x_{j\pi}$ after setting the variables $\{x_{1\pi},\ldots,x_{(j-1)\pi}\}$ to fixed values. Similarly, $c_{j\pi}'$ is the number of such subfunctions of f^π or $\bar f^\pi$. We will try to prove that $b_{j\pi-1}\le c_{j\pi}$ and $b_{j\pi-1}'\le c_{j\pi}'$ for all j.

Case 1: $\circ$ is $\wedge$. We may assume that $n\pi=n$, since $\wedge$ is commutative. *Case 1a:* $1\le j\pi\le k$. Then $b_{j\pi-1}$ and $b_{j\pi-1}'$ count subfunctions in which only the variables $x_{i\pi}$ with $1\le i<j$ and $1\le i\pi\le k$ are specified. These subfunctions of $g\wedge h$ or $\bar g\vee\bar h$ have counterparts that are counted in $c_{j\pi}$ and $c_{j\pi}'$, because h^π is not constant in any subfunction when $n\pi=n$. *Case 1b:* $k<j\pi\le n$. Then $b_{j\pi-1}$ and $b_{j\pi-1}'$ count subfunctions of h or $\bar h$, which have counterparts counted in $c_{j\pi}$ and $c_{j\pi}'$.

Case 2: $\circ$ is $\oplus$. We may assume that $1\pi=1$, since $\oplus$ is commutative. Then an argument analogous to Case 1 applies. [*Discrete Applied Math.* **103** (2000), 237–258.]

recursively
Sauerhoff
Wegener
Werchner
Chang
Spitkovsky
curious properties
random bit generation
threshold function
prime implicants

167. Let $f = f_{1n}$; proceed recursively to compute $c_{ij} = B_{\min}(f_{ij})$, $c'_{ij} = B_{\min}(f_{ij}, \bar{f}_{ij})$, and a permutation π_{ij} of $\{i, \ldots, j\}$ for each subfunction $f_{ij}(x_i, \ldots, x_j)$ as follows: If $i = j$, we have $f_{ij}(x_i) = x_i$; let $c_{ij} = 3$, $c'_{ij} = 4$, $\pi_{ij} = i$. Otherwise $i < j$, and we have $f_{ij}(x_i, \ldots, x_j) = f_{ik}(x_i, \ldots, x_k) \circ f_{(k+1)j}(x_{k+1}, \ldots, x_j)$ for some k and some operator $\circ$. If $\circ$ is like $\wedge$, let $c_{ij} = c_{ik} + c_{(k+1)j} - 2$, and either ($c'_{ij} = 2c_{ik} + c'_{(k+1)j} - 4$, $\pi_{ij} = \pi_{ik}\pi_{(k+1)j}$) or ($c'_{ij} = 2c_{(k+1)j} + c'_{ik} - 4$, $\pi_{ij} = \pi_{(k+1)j}\pi_{ik}$), whichever minimizes c'_{ij}. If $\circ$ is like $\oplus$, let $c'_{ij} = c'_{ik} + c'_{(k+1)j} - 2$, and either ($c_{ij} = c_{ik} + c'_{(k+1)j} - 2$, $\pi_{ij} = \pi_{ik}\pi_{(k+1)j}$) or ($c_{ij} = c_{(k+1)j} + c'_{ik} - 2$, $\pi_{ij} = \pi_{(k+1)j}\pi_{ik}$), whichever minimizes c_{ij}.

(The permutations π_{ij} represented as strings in this description would be represented as linked lists inside a computer. We could also construct an optimum BDD for f recursively in $O(B_{\min}(f))$ steps, using answer 163.)

168. (a) This statement transforms and simplifies the recurrences (112) and (113).

(b) True by induction; also $x \geq n$.

(c) Easily verified. Notice that T is a reflection about the $22\frac{1}{2}°$ line $y = (\sqrt{2}-1)x$.

(d) If $z \in S_k$ and $z' \in S_{n-k}$ we have $|z| = q^\beta$ and $|z'| = q'^\beta$, where $q \leq k$ and $q' \leq n-k$ by induction. By symmetry we may let $q = (1-\delta)t$ and $q' = (1+\delta)t$, where $t = \frac{1}{2}(q+q') \leq \frac{1}{2}n$. Then if the first hint is true, we have $|z \bullet z'| \leq (2t)^\beta \leq n^\beta$. And we also will have $|z \circ z'| \leq n^\beta$, by (c), since $|z^T| = |z|$.

To prove the first hint, we note that the maximum $|z \bullet z'|$ occurs when $y = y'$. For when $y \geq y'$ we have $|z \bullet z'|^2 = (x+x'+y')^2 + y^2 = r^2 + 2(x'+y')x + (x'+y')^2$; the largest value, given z', occurs when $y = y'$. A similar argument applies when $y' \geq y$.

Now when $y = y'$ we have $y = \sqrt{rr'}\sin\theta$ for some θ; and one can show that $x + x' \leq (r+r')\cos\theta$. Thus $z \bullet z' = (x+x'+y, y)$ lies in the ellipse of the second hint. On that ellipse we have $(a\cos\theta + b\sin\theta)^2 + (b\sin\theta)^2 = a^2/2 + b^2 + u\sin 2\theta + v\cos 2\theta = a^2/2 + b^2 + w\sin(2\theta+\tau)$, where $u = ab$, $v = \frac{1}{2}a^2 - b^2$, $w^2 = u^2 + v^2$, and $\cos\tau = u/w$. Hence $|z\bullet z'|^2 \leq \frac{1}{2}a^2 + b^2 + w$. And $4w^2 = (r+r')^4 + 4(rr')^2 \leq (r^2 + (2\sqrt{5}-2)rr' + r'^2)^2$, so

$$|z \bullet z'|^2 \leq r^2 + (\sqrt{5}+1)rr' + r'^2, \qquad r = (1-\delta)^\beta,\ r' = (1+\delta)^\beta.$$

The remaining task is to prove that this quantity is at most $2^{2\beta} = 2\phi^2$; equivalently, $f_t(2) \leq f_t(2\beta)$, where $f_t(\alpha) = (e^{t/\alpha} + e^{-t/\alpha})^\alpha - 2^\alpha$ and $t = \beta\ln((1-\delta)/(1+\delta))$. One can show, in fact, that f_t is an increasing function of α when $\alpha \geq 2$.

[The $O(n^\beta)$ bound on S_n seems to require a delicate analysis; an earlier attempt by Sauerhoff, Wegener, and Werchner was flawed. The proof given here is due to A. X. Chang and V. I. Spitkovsky in 2007.]

169. This conjecture has been verified for $m \leq 7$. [Many other curious properties also remain unexplained. A paper that describes what is known so far is currently being prepared by members of the "curious research group."]

170. (a) 2^{2n-1}. There are four choices at j when $1 \leq j < n$, namely LO = $\boxed{\bot}$ or LO = $\boxed{\top}$ or HI = $\boxed{\bot}$ or HI = $\boxed{\top}$; and there are two choices for n.

(b) 2^{n-1}, since half the choices at each branch are ruled out.

(c) Indeed, if $t = (t_1 \ldots t_n)_2$ we have LO = $\boxed{\bot}$ at j when $t_j = 1$ and HI = $\boxed{\top}$ at j when $t_j = 0$. (This idea was applied to random bit generation in exercise 3.4.1–25. Since there are 2^{n-1} such values of t, we've shown that every monotone, skinny function is a threshold function, with weights $\{2^{n-1}, \ldots, 2, 1\}$. The other skinny functions are obtained by complementing individual variables.)

(d) $\bar{f}_t(\bar{x}) = [(\bar{x})_2 < t] = [(x)_2 > \bar{t}] = [(x)_2 > 2^n - 1 - t] = f_{2^n - t}(x)$.

(e) By Theorem 7.1.1Q, the shortest DNF is the OR of the prime implicants, and its general pattern is exhibited by the case $n = 10$ and $t = (1100010111)_2$: $(x_1 \wedge x_2 \wedge x_3) \vee$

$(x_1\wedge x_2\wedge x_4)\vee(x_1\wedge x_2\wedge x_5)\vee(x_1\wedge x_2\wedge x_6\wedge x_7)\vee(x_1\wedge x_2\wedge x_6\wedge x_8\wedge x_9\wedge x_{10})$. (One term for each 0 in t, and one more.) The shortest CNF is the dual of the shortest DNF of the dual, which corresponds to $2^n-t=(0011101001)_2$: $(x_1)\wedge(x_2)\wedge(x_3\vee x_4\vee x_5\vee x_6)\wedge(x_3\vee x_4\vee x_5\vee x_7\vee x_8)\wedge(x_3\vee x_4\vee x_5\vee x_7\vee x_9)\wedge(x_3\vee x_4\vee x_5\vee x_7\vee x_{10})$.

171. Note that the classes of read-once, regular, skinny, and monotone functions are each closed under the operations of taking duals and restrictions. A skinny function is clearly read-once; a monotone threshold function with $w_1\ge\cdots\ge w_n$ is regular; and a regular function is monotone. We must show that a regular read-once function is skinny.

Suppose $f(x_1,\ldots,x_n)=g(x_{i_1},\ldots,x_{i_k})\circ h(x_{j_1},\ldots,x_{j_l})$, where $\circ$ is a nontrivial binary operator and we have $i_1<\cdots<i_k$, $j_1<\cdots<j_l$, $k+l=n$, and $\{i_1,\ldots,i_k,j_1,\ldots,j_l\}=\{1,\ldots,n\}$. (This condition is weaker than being "read-once.") We can assume that $i_1=1$. By taking restrictions and using induction, both g and h are skinny and monotone; thus their prime implicants have the special form in exercise 170(e). The operator $\circ$ must be monotone, so it is either $\vee$ or $\wedge$. By duality we can assume that $\circ$ is $\vee$.

Case 1: f has a prime implicant of length 1. Then x_1 is a prime implicant of f, by regularity. Hence $f(x_1,\ldots,x_n)=x_1\vee f(0,x_2,\ldots,x_n)$, and we can use induction.

Case 2: All prime implicants of g and h have length >1. Then $x_{j_1}\wedge\cdots\wedge x_{j_p}$ is a prime implicant, for some $p\ge 2$, but $x_{j_1-1}\wedge x_{j_2}\wedge\cdots\wedge x_{j_p}$ is not, contradicting regularity. [See T. Eiter, T. Ibaraki, and K. Makino, *Theor. Comp. Sci.* **270** (2002), 493–524.]

172. By examining the CNF for f_t in exercise 170(e), we see that when $t=(t_1\ldots t_n)_2$ the number of Horn functions obtainable by complementing variables is one more than the number for $(t_2\ldots t_n)_2$ when $t_1=0$, but twice that number when $t_1=1$. Thus the example $t=(1100010111)_2$ corresponds to $2\times(2\times(1+(1+(1+(2\times(1+(2\times(2\times 2)))))))$ Horn functions. Summing over all t gives s_n where $s_n=(2^{n-2}+s_{n-1})+2s_{n-1}$, where $s_1=2$; and the solution to this recurrence is 3^n-2^{n-1}.

To make both f and $\bar f$ Horn functions, assume (by duality) that $t \bmod 4=3$. Then we must complement x_j if and only if $t_j=0$, except for the string of 1s at the right of t. For example, when $t=(1100010111)_2$, we should complement x_3, x_4, x_5, x_7, and then at most one of $\{x_8,x_9,x_{10}\}$. This gives $\rho(t+1)+1\ge 3$ choices related to f_t. Summing over all t with $t \bmod 4=3$ gives 2^n-1; so the answer is $2^{n+1}-2$.

173. Consider monotone functions first. We can write $t=(0^{a_1}1^{a_2}\ldots 0^{a_{2k-1}}1^{a_{2k}})_2$, where $a_1+\cdots+a_{2k}=n$, $a_1\ge 0$, $a_j\ge 1$ for $1<j<2k$, and $a_{2k}\ge 2$ when $t \bmod 4=3$. When $t \bmod 4=1$, 2^n-t has this form. Then f_t has $a_1!\,a_2!\ldots a_{2k}!$ automorphisms, so it is equivalent to $n!/(a_1!\,a_2!\ldots a_{2k}!)-1$ others, none of which are skinny. Summing over all t gives $2(P_n-nP_{n-1})$ monotone Boolean functions that are reorderable to skinny form, when $n\ge 2$, where P_n is the number of weak orderings (exercise 5.3.1–3). [See J. S. Beissinger and U. N. Peled, *Graphs and Combinatorics* **3** (1987), 213–219.]

Every such monotone function corresponds to 2^n different unate functions that are equally skinny, when variables are complemented. (These are the functions with the property that all of their restrictions are canalizing, known also as "unate cascades," "1-decision list functions," or "generalized read-once threshold functions.")

174. (a) Assign the numbers $0,\ldots,n{-}1,n,n{+}1$ to nodes 1, …, n, $\boxed{\top}$, $\boxed{\bot}$; and let the (LO, HI) branches from node k go to nodes (a_{2k+1},a_{2k+2}) for $0\le k<n$. Then define p_k as follows, for $1\le k\le 2n$: Let $l=\lfloor(k-1)/2\rfloor$ and $P_l=\{p_1,\ldots,p_{2l}\}$. Set $p_k\leftarrow a_k$ if $a_k\notin P_l$; otherwise, if a_k is the mth smallest element of $P_l\cap\{l{+}1,\ldots,n{+}1\}$, set p_k to the mth smallest element of $\{n+2,\ldots,n+l+1\}\setminus P_l$. (This construction is due to T. Dahlheimer.)

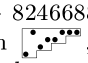
duals
restrictions
read-once functions, generalized
Eiter
Ibaraki
Makino
recurrence
automorphisms
weak orderings
Beissinger
Peled
restrictions
canalizing
unate cascades
1-decision list functions
read-once threshold functions
Dahlheimer

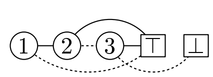
Genocchi numbers
median Genocchi numbers
Euler
Bernoulli numbers
Genocchi
Seidel

(b) The inverse $p_1^{-1}\ldots p_{2n}^{-1}$ of a Dellac permutation satisfies $2(k-n)-1 \le p_k^{-1} \le 2k$. It corresponds to a Genocchi derangement $q_1\ldots q_{2n+2}$ when $q_2 = 1$, $q_{2n+1} = 2n+2$, and $q_{2k+2} = 1 + p_k^{-1}$, $q_{2k-1} = 1 + p_{k+n}^{-1}$ for $1 \le k \le n$.

(c) Given a permutation $q_1\ldots q_{2n+2}$, let r_k be the first element of the sequence q_k^{-1}, q_k^{-2}, $\ldots$ that is $\ge k$. This transformation takes Genocchi permutations into Dumont pistols, and has the property that $q_k = k$ if and only if $r_k = k \notin \{r_1, \ldots, r_{k-1}\}$.

(d) Each node (j,k) represents a set of strings $r_1\ldots r_j$, where $(1,0) = \{1\}$ and the other sets are defined by the following transition rules: Suppose $r_1\ldots r_j \in (j,k)$, and let $l = 2k$. If $k = 0$ then $(j+1,k)$ contains $1r_1^+\ldots r_j^+$ when j is even, $2r_1^+\ldots r_j^+$ when j is odd, where r^+ denotes $r+1$. If $k > 0$ then $(j+1,k)$ contains $r_1^+\ldots r_l^+(l{+}1)r_{l+1}^+\ldots r_j^+$ when j is even, $r_1^\pm\ldots r_{l-1}^\pm(l)r_l^\pm\ldots r_j^\pm$ when j is odd, where $r^\pm$ denotes $r+1$ when $r \ge l$, $r-1$ when $r < l$. Going vertically, if $l \le j-3$ and j is odd, $(j,k+1)$ contains $r_1\ldots r_l r_{l+2} r_{l+3}(l{+}3)r_{l+4}\ldots r_j$. On the other hand if $k = 1$ and j is even, $(j,0)$ contains $r_2 r_1 r_3\ldots r_j$. Finally if $k > 1$ and j is even, $(j,k-1)$ contains the string $r_1'\ldots r_{l-3}'(l{-}2)r_{l-2}'r_{l-1}'r_{l+1}'\ldots r_j'$, where r' denotes l when $r = l-2$, otherwise $r' = r$. (One can show that the elements of $(2j,k)$ are the Dumont pistols for Genocchi permutations of order $2j$ whose largest fixed point is $2k$.)

All of these constructions are invertible. For example, the path (1,0) $\to$ (2,0) $\to$ (3,0) $\to$ (3,1) $\to$ (4,1) $\to$ (5,1) $\to$ (6,1) $\to$ (7,1) $\to$ (7,2) $\to$ (7,3) $\to$ (8,3) $\to$ (8,2) $\to$ (8,1) $\to$ (8,0) corresponds to the pistols 1 $\to$ 22 $\to$ 133 $\to$ 333 $\to$ 4244 $\to$ 53355 $\to$ 624466 $\to$ 7335577 $\to$ 7355577 $\to$ 7355777 $\to$ 82448688 $\to$ 82646888 $\to$ 82466888 $\to$ 28466888. The latter pistol, which can be represented by the diagram , corresponds to the Genocchi derangement $q_1\ldots q_8 = 61537482$. And this derangement corresponds to $p_1^{-1}\ldots p_6^{-1} = 231546$ and the Dellac permutation $p_1\ldots p_6 = 312546$. That permutation, in turn, corresponds to $a_1\ldots a_6 = 312343$, which stands for the thin BDD



Let d_{jk} be the number of pistols in (j,k), which is also the number of directed paths from $(1,0)$ to (j,k). These numbers are readily found by addition, beginning with

												38227	38227	···
										2073	2073	38227	76454	···
								155	155	2073	4146	36154	112608	···
						17	17	155	310	1918	6064	32008	144616	···
				3	3	17	34	138	448	1608	7672	25944	170560	···
		1	1	3	6	14	48	104	552	1160	8832	18272	188832	···
1	1	1	2	2	8	8	56	56	608	608	9440	9440	198272	··· ;

and the column totals $D_j = \sum_k d_{jk}$ are $(D_1, D_2, \ldots) = (1, 1, 2, 3, 8, 17, 56, 155, 608, 2073, 9440, 38227, 198272, 929569, \ldots)$. The even-numbered elements of this sequence, D_{2n}, have long been known as the Genocchi numbers G_{2n+2}. The odd-numbered elements, D_{2n+1}, have therefore been called “median Genocchi numbers.” The number S_n of thin BDDs is $d_{(2n+2)0} = D_{2n+1}$.

References: L. Euler discussed the Genocchi numbers in the second volume of his *Institutiones Calculi Differentialis* (1755), Chapter 7, where he showed that the odd integers G_{2n} are expressible in terms of the Bernoulli numbers: In fact, $G_{2n} = (2^{2n+1}-2)|B_{2n}|$, and $z\tan\frac{z}{2} = \sum_{n=1}^{\infty} G_{2n}z^{2n}/(2n)!$. A. Genocchi examined these numbers further in *Annali di Scienze Matematiche e Fisiche* **3** (1852), 395–405; and L. Seidel, in *Sitzungsberichte math.-phys. Classe, Akademie Wissen. München* **7** (1877), 157–187, discovered that they could be computed additively via the numbers d_{jk}. Their combinatorial significance was not discovered until much later; see D. Dumont, *Duke Math. J.* **41** (1974), 305–318; D. Dumont and A. Randrianarivony, *Discrete Math.* **132** (1994), 37–49. Meanwhile H. Dellac had proposed an apparently unrelated problem, equivalent to enumerating what we have called Dellac permutations; see L'*Intermédiaire des Math.* **7** (1900), 9–10, 328; *Annales de la Faculté sci. Marseille* **11** (1901), 141–164.

There's also a *direct* connection between thin BDDs and the paths of (d), discovered in 2007 by Thorsten Dahlheimer. Notice first that unrestricted Dumont pistols of order $2n+2$ correspond to thin BDDs that are ordered but not necessarily reduced, because we can let $r_1 \ldots r_{2n} r_{2n+1} r_{2n+2} = (2a_1)\ldots(2a_{2n})(2n{+}2)(2n{+}2)$. The number of such pistols in which $\min\{i \mid r_{2i-1} = r_{2i}\} = l$ turns out to be $d_{(2n+2)(n+1-l)}$.

To prove this, we can use new transition rules instead of those in answer (d): Suppose $r_1 \ldots r_j \in (j,k)$, and let $l = j - 2k$. Then $(j+1,k)$ contains $r_1^+ \ldots r_l^+ r_l^+ \ldots r_j^+$ when j is odd, $r_1^\pm \ldots r_{l-1}^\pm (l{-}1) r_l^\pm \ldots r_j^\pm$ when j is even. If j is odd, $(j,k+1)$ contains $1r_1r_3 \ldots r_j$ when $l = 3$, and when $l > 3$ it contains $r'_1 \ldots r'_{l-4}(l{-}4)r'_{l-3}r'_{l-2}r'_l \ldots r'_j$, where $r' = r + 2[r = l{-}4]$. Finally, if j is even and $k > 0$, $(j,k-1)$ contains $r_1 \ldots r_{l-1} q r_{l+2} r_{l+2} \ldots r_j$, where $q = l$ if $r_l = r_{l+1}$, otherwise $q = r_{l+1}$.

With these magic transitions the path above corresponds to $1 \to 22 \to 313 \to 133 \to 2244 \to 31355 \to 424466 \to 5153577 \to 5135577 \to 1535577 \to 22646688 \to 26446688 \to 26466688 \to 26466888$; so $a_1 \ldots a_6 = 132334$.

175. This problem seems to require a different approach from the methods that worked when $b_0 = \cdots = b_{n-1} = 1$. Suppose we have a BDD base of N nodes including the two sinks $\boxed{\bot}$ and $\boxed{\top}$ together with various branches labeled (2), …, (n), and assume that exactly s of the nodes are sources (having in-degree zero). Let $c(b,s,t,N)$ be the number of ways to introduce b additional nodes labeled (1), in such a way that exactly $s+b-t$ source nodes remain. (Thus $0 \le t \le 2b$; exactly t of the old source nodes are now reachable from a (1) branch.) Then the number of nonconstant Boolean functions $f(x_1,\ldots,x_n)$ having the BDD profile $(b_0,\ldots,b_n)$ is equal to $T(b_0,\ldots,b_{n-1};1)$, where

$$T(b_0;s) = 2[s = b_0 = 1] + [s = 2][b_0 = 0] + [s = 2][b_0 = 2];$$

$$T(b_0,\ldots,b_{n-1};s) = \sum_{t=\max(0,b_0-s)}^{2b_0} c(b_0, s{+}t{-}b_0, t, b_1{+}\cdots{+}b_{n-1}{+}2)\, T(b_1,\ldots,b_{n-1}; s{+}t{-}b_0).$$

One can show that $c(b,s,t,N) = \sum_{r=0}^{2b} a_{rb}\, p_{tr}(s,N)/b!$, where we have $\bigl(N(N-1)\bigr)^{\underline{b}} = \sum_{r=0}^{2b} a_{rb} N^r$ and $p_{tr}(s,N) = \sum_k \binom{r}{k}\left\{{k \atop t}\right\} s^{\underline{t}}(N-s)^{r-k} = \sum_k \left\{{r \atop k}\right\}\binom{k}{t} s^{\underline{t}}(N-s)^{\underline{k-t}} = r!\,[w^t z^r]\, e^{(N-s)z}(we^z - w + 1)^s$.

176. (a) If $p \ne p'$ we have $\sum_{a\in A, b\in B}[h_{a,b}(p) = h_{a,b}(p')] \le |A||B|/2^l$, by the definition of universal hashing. Let $r_i(a,b)$ be the number of $p \in P$ such that $h_{a,b}(p) = i$. Then

$$\begin{aligned}\sum_{a\in A,b\in B}\ \sum_{0\le i<2^l} r_i(a,b)^2 &= \sum_{a\in A,b\in B}\ \sum_{p\in P}\ \sum_{p'\in P}[h_{a,b}(p) = h_{a,b}(p')]\\ &\le |P||A||B| + \sum_{p\in P}\ \sum_{p'\in P}[p \ne p']\frac{|A||B|}{2^l} = 2^t|A||B|\Bigl(1 + \frac{2^t-1}{2^l}\Bigr).\end{aligned}$$

On the other hand $\sum_{i=0}^{2^l-1} r_i(a,b)^2 = \sum_{i=0}^{2^l-1}\bigl(r_i(a,b) - 2^t/|I|\bigr)^2 + 2^{2t}/|I| \ge 2^{2t}/|I|$, for any a and b. Similar formulas apply when there are $s_j(a,b)$ solutions to $h_{a,b}(q) = j$.

Dumont
Dumont
Randrianarivony
Dellac
Dahlheimer
ordered
reduced
BDD base

So there must be $a \in A$ and $b \in B$ such that carries

$$\frac{2^{2t}}{|I|} + \frac{2^{2t}}{|J|} \;\le\; \sum_{i\in I} r_i(a,b)^2 + \sum_{j\in J} s_j(a,b)^2 \;\le\; 2^{t+1}\Bigl(1+\frac{2^t-1}{2^l}\Bigr) \;\le\; \frac{2^{2t}}{2^l} + \frac{2^{2t}}{(1-\epsilon)2^l}.$$

(b) The middle l bits of aq_k+b and $aq_{k+2}+b$ differ by at least 2, so the middle $l-1$ bits of aq_k and aq_{k+2} must be different.

(c) Let q and q' be different elements of Q^* with $(g(q')-g(q)) \bmod 2^{l-1} \ge 2^{l-2}$. (Otherwise we can swap $q \leftrightarrow q'$.) If $l \ge 3$, the condition $g(p)+g(q) = 2^{l-1}$ implies that $f_q(p) = 0$. Now $(g(p)+g(q')) \bmod (2^{l-1}) = (g(q')-g(q)) \bmod (2^{l-1})$; furthermore $g(q')$ and $g(p)$ are both even. Therefore no carry can propagate to change the middle bit, and we have $f_{q'}(p) = 1$.

(d) The set Q'' has at least $(1-\epsilon)2^{l-1}$ elements, and so does the analogous set P''. At most 2^{l-2} elements of Q'' have $g(q)$ odd; and at most $2^{l-1}+1-|P''|$ of the elements with $g(q)$ even are not in Q^*. Thus $|Q^*| \ge (1-\epsilon)2^{l-1} - 2^{l-2} - 2^{l-1} - 1 + (1-\epsilon)2^{l-1} = (1-4\epsilon)2^{l-2} - 1$, and we have $B_{\min}(Z_{n,y}) \ge (1-4\epsilon)2^{l-1} - 2$ by (85).

Finally, choose $l = t-4$ and $\epsilon = 1/9$. The theorem is obvious when $n < 14$.

177. Suppose $k \ge n/2$ and $x = 2^{k+1}x_h + x_l$, $y = 2^k y_h + y_l$. Then $(xy \gg k) \bmod 2^{n-k}$ depends on $2x_h y_l$, $x_l y_h$, and $x_l y_l \gg k$, modulo 2^{n-k}, so $q_{2k+1} \le 2^{n-k-1+n-k+n-k}$.

Summing up, we get $\sum_{k=0}^{2n} q_k \le \sum_{0\le k\le 6n/5} 2^k + \sum_{6n/5<k\le 2n} 2^{3n-2\lfloor k/2\rfloor-\lceil k/2\rceil}$. If $n = 5t+(0,1,2,3,4)$ the total comes to exactly $(2^{\lceil 6n/5\rceil}\cdot(19,10,12,13,17)-12)/7$.

178. We can write $x = 2^k x_h + x_l$ as in the proof of Theorem A; but now $x_l = \hat{x}_l + (x \bmod 2)$, where $\hat{x}_l$ is even and $x \bmod 2$ is not yet known. Similarly $y = 2^k y_h + y_l = 2^k y_h + \hat{y}_l + (y \bmod 2)$. Let $\hat{z}_l = \hat{x}_l\hat{y}_l \bmod 2^k$. At level $2k-2$, for $n/2 \le k < n$, we need only "remember" three $(n-k)$-bit numbers $\hat{x}_l \bmod 2^{n-k}$, $\hat{y}_l \bmod 2^{n-k}$, $(\hat{x}_l\hat{y}_l \gg k) \bmod 2^{n-k}$, and three "carries" $c_1 = (\hat{x}_l+\hat{z}_l) \gg k$, $c_2 = (\hat{y}_l+\hat{z}_l) \gg k$, $c_3 = (\hat{x}_l+\hat{y}_l+\hat{z}_l) \gg k$. These six quantities will suffice to determine the middle bit, after x_h, y_h, $x \bmod 2$, and $y \bmod 2$ become known.

There are only six possibilities for the carries: $c_1c_2c_3 = 000$, 001, 011, 101, 111, or 112. Thus $q_{2k-2} \le 6\cdot 2^{(n-k-1)+(n-k-1)+(n-k)}$. Similarly, when $n/2 \le k < n-1$, we have $q_{2k-1} \le 6\cdot 2^{(n-k-2)+(n-k-1)+(n-k)}$. With these estimates, together with $q_k \le 2^k$, we get $\sum_{k=0}^{2n-4} q_k \le (2^{6t}\cdot(37,86,184,464,1024)-268)/28$ when $n = 5t+(0,1,2,3,4)$.

The actual BDD sizes, for the function f of Theorem A and the function g of this exercise, are $B(f) = (169$, 381, 928, 2188, 5248, 12373, 29400, 68777, 162768, 377359, $879709)$ and $B(g) = (165$, 352, 806, 1802, 4195, 9774, 22454, 52714, 121198, 278223, $650188)$ for $6 \le n \le 16$; so this variant appears to save about 25%. A slightly better ordering is obtained by testing (lo-bit(x), hi-bit(y), hi-bit(x), lo-bit(y)) on the last four levels, giving $B(h) = B(g)-20$ for $n \ge 6$. Then $B(h)/B_{\min}(f) \approx (1.07$, 1.05, 1.04, 1.04, 1.04, 1.01, $1.02)$ for $6 \le n \le 12$, so this ordering may be close to optimal as $n \to \infty$.

180. By letting $a_{m+1} = a_{m+2} = \cdots = 0$, we may assume that $m \ge p$. Let $a = (a_p \ldots a_1)_2$, and write $x = 2^k x_h + x_l$ as in the proof of Theorem A. If $p \le n$, we have $q_k \le 2^{p-k}$ for $0 \le k < p$, because the given function $f = Z^{(p)}_{m,n}(a;x)$ depends only on a, x_h, and $(ax_l \gg k) \bmod 2^{p-k}$. We may therefore assume that $p > n$.

Consider the multiset $A = \{2^k x_h a \bmod 2^{p-1} \mid 0 \le x_h < 2^{n-k}\}$. Write $A = \{2^{p-1}-\alpha_1, \ldots, 2^{p-1}-\alpha_s\}$, where $s = 2^{n-k}$ and $0 < \alpha_1 \le \cdots \le \alpha_s = 2^{p-1}$, and let $\alpha_{s+i} = \alpha_i + 2^{p-1}$ for $0 \le i \le s$. Then $q_k \le 2s$, because f depends only on a, x_h, and the index $i \in [0\,.\,.\,2s)$ such that $\alpha_i \le ax_l \bmod 2^p < \alpha_{i+1}$.

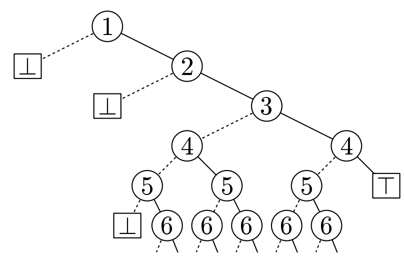
Consequently $\sum_{k=0}^{n} q_k \le \sum_{k=0}^{n} \min(2^k, 2^{n+1-k}) = 2^{\lfloor n/2\rfloor+1} + 2^{\lceil n/2\rceil+1} - 3$.

Bollig
zeads
de Bruijn cycle
braces

181. For every $(x_1,\ldots,x_m)$ the remaining function of $(y_1,\ldots,y_n)$ requires $O(n)$ nodes, by exercise 170.

182. Yes; B. Bollig [*Lecture Notes in Comp. Sci.* **4978** (2008), 306–317] has shown that it is $\Omega(2^{n/432})$. Incidentally, $B_{\min}(L_{12,12}) = 1158$ is obtained with the strange ordering $L_{12,12}(x_{18},x_{17},x_{16},x_{15},x_{14},x_{12},x_{10},x_8,x_6,x_4,x_2,x_1;x_{19},x_{20},x_{21},x_{22},x_{23},x_{13},x_{11},x_9,x_7,x_5,x_3,x_{24})$; and $B_{\max}(L_{12,12}) = 9302$ arises with $L_{12,12}(x_{24},x_{23},x_{20},x_{19},x_{22},x_{11},x_6,x_7,x_8,x_9,x_{10},x_{13};x_1,x_2,x_3,x_4,x_5,x_{21},x_{18},x_{17},x_{16},x_{15},x_{14},x_{12})$. Similarly $B_{\min}(L_{8,16}) = 606$ and $B_{\max}(L_{8,16}) = 3415$ aren't terribly far apart. Could $B_{\min}(L_{m,n})$ and $B_{\max}(L_{m,n})$ both conceivably be $\Theta(2^{\min(m,n)})$?

183. The profile $(b_0,b_1,\ldots)$ begins (1, 1, 1, 2, 3, 5, 7, 11, 15, 23, 31, 47, 63, 95, ...). When $k>0$ there's a node on level $2k$ for every pair of integers (a,b) such that $2^{k-1}\le a,b<2^k$ and $ab<2^{2k-1}<(a+1)(b+1)$; this node represents the function $[((a+x)/2^k)((b+y)/2^k)\ge\frac12]$. When b is given, in the appropriate range, there are $\lceil 2^{2k-1}/b\rceil-\lfloor 2^{2k-1}/(b+1)\rfloor$ choices for a; hence $b_{2k}=\sum_{2^{k-1}\le b<2^k}(\lceil 2^{2k-1}/b\rceil-\lfloor 2^{2k-1}/(b+1)\rfloor)$, which telescopes to 2^k-1. A similar argument shows that $b_{2k+1}=2^k+2^{k-1}-1$.



184. Two kinds of beads contribute to $b_{m(i-1)+j-1}$: One for every choice of i columns, at least one of which is $<j$; and one for every choice of $i-1$ columns, missing at least one element $\ge j$. Thus $b_{m(i-1)+j-1}=\bigl(\binom{m}{i}-\binom{m+1-j}{i}\bigr)+\bigl(\binom{m}{i-1}-\binom{j-1}{m+1-i}\bigr)$. Summing over $1\le i,j\le m$ gives $B(P_m)=(2m-3)2^m+5$. (Incidentally, $q_k=b_k+1$ for $2\le k<m^2$.)

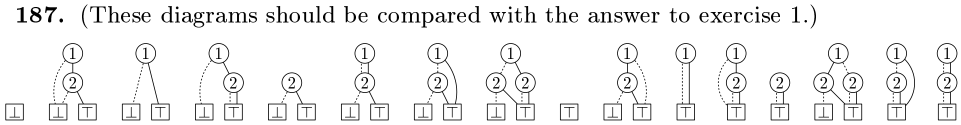
The ZDD has simply $z_{m(i-1)+j-1}=\binom{n-1}{i-1}$ for $1\le i,j\le m$, one for every choice of $i-1$ columns $\ne j$; hence $Z(P_m)=m2^{m-1}+2\approx\frac14 B(P_m)$. (The lower bound of Theorem K applies also to ZDD nodes, because only such nodes get tickets; therefore the natural ordering of variables is optimum for ZDDs. The natural ordering might be optimum also for BDDs; this conjecture is known to be true for $m\le 5$.)

185. Suppose $f(x)=t_{\nu x}$ for some binary vector $t_0\ldots t_n$. Then the subfunctions of order $d>0$ correspond to the distinct substrings $t_i\ldots t_{i+d}$. Such substrings τ correspond to beads if and only if $\tau\ne 0^{d+1}$ and $\tau\ne 1^{d+1}$; they correspond to zeads if and only if $\tau\ne 0^{d+1}$ and $\tau\ne 10^d$.

Thus the maximum $Z(f)$ is the function S_n of answer 44. To attain this worst case we need a binary vector of length $2^{d+1}+d-2$ that contains all $(d+1)$-tuples except 0^{d+1} and 10^d as substrings; such vectors can be characterized as the first $2^{d+1}+d-2$ elements of any de Bruijn cycle of period 2^{d+1}, beginning with 0^d1.

186. $\bar x_1\wedge\bar x_2\wedge x_3\wedge\bar x_4\wedge\bar x_5\wedge\bar x_6$.

187. (These diagrams should be compared with the answer to exercise 1.)



1 4 3 4 3 4 4 5 1 4 2 3 2 5 3 3

188. To avoid nested braces, let ϵ, a, b, and ab stand for the subsets $\emptyset$, $\{1\}$, $\{2\}$, and $\{1,2\}$. The families are then $\emptyset$, $\{ab\}$, $\{a\}$, $\{a,ab\}$, $\{b\}$, $\{b,ab\}$, $\{a,b\}$, $\{a,b,ab\}$, $\{\epsilon\}$, $\{\epsilon,ab\}$, $\{\epsilon,a\}$, $\{\epsilon,a,ab\}$, $\{\epsilon,b\}$, $\{\epsilon,b,ab\}$, $\{\epsilon,a,b\}$, $\{\epsilon,a,b,ab\}$, in truth-table order.

189. When $n=0$, only the constant functions; when $n>0$, only 0 and $x_1\wedge\cdots\wedge x_n$. (But there are many functions, such as $x_2\wedge(x_1\vee\bar x_3)$, with $(b_0,\ldots,b_n)=(z_0,\ldots,z_n)$.)

190. (a) Only $x_1 \oplus \cdots \oplus x_n$ and $1 \oplus x_1 \oplus \cdots \oplus x_n$, for $n \geq 0$. (b) This condition holds if and only if all subtables of order 1 are either 01 or 11. So there are $2^{2^{n-1}}$ solutions when $n > 0$, namely all functions such that $f(x_1, \ldots, x_{n-1}, 1) = 1$.

191. The language L_n of truth tables for all such functions has the context-free grammar $L_0 \to 1$; $L_{n+1} \to L_n L_n \mid L_n 0^{2^n}$. The desired number $l_n = |L_n|$ therefore satisfies $l_0 = 1$, $l_{n+1} = l_n(l_n + 1)$; so $(l_0, l_1, l_2, \ldots)$ is the sequence (1, 2, 6, 1806, 3263442, ...). Asymptotically, $l_n = \theta^{2^n} - \frac{1}{2} - \epsilon$, where $0 < \epsilon < \theta^{-2^n}/8$ and

$$\theta = 1.59791\,02180\,31873\,17833\,80701\,18157\,45531\,23622+.$$

[See *CMath* exercises 4.37 and 4.59, where l_n+1 is called e_{n+1} (a "Euclid number") and θ is called E^2. The numbers l_n+1 were introduced by J. J. Sylvester in connection with his study of Egyptian fractions, *Amer. J. Math.* **3** (1880), 388. Notice that a monotone decreasing function, like a function representing independent sets, always has $z_n = 1$.]

192. (a) 10101101000010110.

(b) True, by induction on $|\tau|$, because $\alpha \neq \beta \neq 0^n$ if and only if $\alpha^Z \neq \beta^Z \neq 0^n$.

(c) The beads of f of order k are the zeads of f^Z of order k, for $0 < k \leq n$. Hence the beads of f^Z are also the zeads of $(f^Z)^Z = f$. Therefore, if $(b_0, \ldots, b_n)$ and $(z_0, \ldots, z_n)$ are the profile and z-profile of f while $(b'_0, \ldots, b'_n)$ and $(z'_0, \ldots, z'_n)$ are the profile and z-profile of f^Z, we have $b_k = z'_k$ and $z_k = b'_k$ for $0 \leq k < n$.

(We also have $z_n = z'_n$, but they might both be 1 instead of 2. The *quasi-profiles* of f and f^Z may differ, but only by at most 1 at each level, because of all-0 subtables.)

193. $S_{\geq k}(x_1, \ldots, x_n)$, by induction on n. (Hence we also have $S^Z_{\geq k}(x_1, \ldots, x_n) = S_k(x_1, \ldots, x_n)$. Exercise 249 gives similar examples.)

194. Define $a_1 \ldots a_{2n}$ as in answer 174, but use the ZDD instead of the BDD. Then $(1, \ldots, 1)$ is the z-profile if and only if $(2a_1) \ldots (2a_{2n})$ is an unrestricted Dumont pistol of order $2n$. So the answer is the Genocchi number G_{2n+2}.

195. The z-profile is $(1, 2, 4, 4, 3, 2, 2)$. We get an optimum z-profile $(1, 2, 3, 2, 3, 2, 2)$ from $M_2(x_4, x_2; x_5, x_6, x_3, x_1)$, and a pessimum z-profile $(1, 2, 4, 8, 12, 2, 2)$ comes from $M_2(x_5, x_6; x_1, x_2, x_3, x_4)$ as in (78). (Incidentally, the algorithm of exercise 197 can be used to show that $Z_{\min}(M_4) = 116$ is obtained with the strikingly peculiar ordering $M_4(x_8, x_5, x_{17}, x_2; x_{20}, x_{19}, x_{18}, x_{16}, x_{15}, x_{13}, x_{14}, x_{12}, x_{11}, x_9, x_{10}, x_4, x_7, x_6, x_3, x_1)$!)

196. For example, $M_m(x_1, \ldots, x_m; e_{m+1}, \ldots, e_n)$, where $n = m + 2^m$ and e_j is the elementary function of exercise 203. Then we have $Z(f) = 2(n - m) + 1$ and $Z(\bar{f}) = (n - m + 7)(n - m)/2 - 2$.

197. The key idea is to change the significance of the DEP fields so that d_{kp} is now $\sum\{2^{t-k-1} \mid N_{kp} \text{ supports } x_t\}$, where we say that $g(x_1, \ldots, x_m)$ *supports* x_j if there is a solution to $g(x_1, \ldots, x_m) = 1$ with $x_j = 1$.

To implement this change, we introduce an auxiliary array $(\zeta_0, \ldots, \zeta_n)$, where we will have $\zeta_k = q$ if N_{kq} denotes the subfunction 0 and $\zeta_k = -1$ if that subfunction does not appear on level k. Initially $\zeta_n \leftarrow 0$, and we set $\zeta_k \leftarrow -1$ at the beginning of step E1. In step E3, the operation of setting d_{kq} should become the following: "If $d_{(k+1)h} \neq \zeta_{k+1}$, set $d_{kq} \leftarrow ((d_{(k+1)l} \mid d_{(k+1)h}) \ll 1) + 1$; otherwise set $d_{kq} \leftarrow d_{(k+1)l} \ll 1$. Also set $\zeta_k \leftarrow q$ if $d_{(k+1)l} = d_{(k+1)h} = \zeta_{k+1}$."

(The master z-profile chart can be used as before to minimize $z_0 + \cdots + z_{n-1}$; but additional work is needed to consider z_n if the *absolute* minimum is important.)

198. Reinterpreting (50), we represent an arbitrary family of sets f as $(\bar{x}_v?\ f_l\colon f_h)$, where $v = f_v$ indexes the first variable that f *supports*; see answer 197. Thus f_l is the

subtables
context-free grammar
Asymptotically
CMath
Euclid number
Sylvester
Egyptian fractions
monotone decreasing function
independent sets
zeads
quasi-profiles
subtables
Dumont pistol
Genocchi number
2^m-way mux
elementary function
supports
family of sets
supports

ZUNIQUE
Reference counts
sanity check
debugging
complementation
tautology
power set
$\wp$ (power set)

subfamily of f that doesn't support x_v, and f_h is the subfamily that does (but with x_v deleted). We also let $f_v = \infty$ if f has no support (i.e., if f is either $\emptyset$ or $\{\emptyset\}$, represented internally by $\boxed{\bot}$ or $\boxed{\top}$; see answer 200). In (52), $v = \min(f_v, g_v)$ now indexes the first variable *supported* by either f or g; thus $f_h = \emptyset$ if $f_v > g_v$, and $g_h = \emptyset$ if $f_v < g_v$.

Subroutine AND(f,g), ZDD-style, is now the following instead of (55): "Represent f and g as in (52). While $f_v \neq g_v$, return $\emptyset$ if either $f = \emptyset$ or $g = \emptyset$; otherwise set $f \leftarrow f_l$ if $f_v < g_v$, set $g \leftarrow g_l$ if $f_v > g_v$. Swap $f \leftrightarrow g$ if $f > g$. Return f if $f = g$ or $f = \emptyset$. Otherwise, if $f \wedge g = r$ is in the memo cache, return r. Otherwise compute $r_l \leftarrow \mathrm{AND}(f_l, g_l)$ and $r_h \leftarrow \mathrm{AND}(f_h, g_h)$; set $r \leftarrow \mathrm{ZUNIQUE}(v, r_l, r_h)$, using an algorithm like Algorithm U except that the first step returns p when $q = \emptyset$ instead of when $q = p$; put '$f \wedge g = r$' into the memo cache, and return r." (See also the suggestion in answer 200.)

Reference counts are updated as in exercise 82, with slight changes; for example, step U1 will now decrease the reference count of $\boxed{\bot}$ (and only of this node), when $q = \emptyset$. It is important to write a "sanity check" routine that double-checks all reference counts and other redundancies in the entire BDD/ZDD base, so that subtle errors are nipped in the bud. The sanity checker should be invoked frequently until all subroutines have been thoroughly tested.

199. (a) If $f = g$, return f. If $f > g$, swap $f \leftrightarrow g$. If $f = \emptyset$, return g. If $f \vee g = r$ is in the memo cache, return r. Otherwise

set $v \leftarrow f_v$, $r_l \leftarrow \mathrm{OR}(f_l, g_l)$, $r_h \leftarrow \mathrm{OR}(f_h, g_h)$, if $f_v = g_v$;
set $v \leftarrow f_v$, $r_l \leftarrow \mathrm{OR}(f_l, g)$, $r_h \leftarrow f_h$, increase `REF(`f_h`)` by 1, if $f_v < g_v$;
set $v \leftarrow g_v$, $r_l \leftarrow \mathrm{OR}(f, g_l)$, $r_h \leftarrow g_h$, increase `REF(`g_h`)` by 1, if $f_v > g_v$.

Then set $r \leftarrow \mathrm{ZUNIQUE}(v, r_l, r_h)$; cache it and return it as in answer 198.

(b) If $f = g$, return $\emptyset$. Otherwise proceed as in (a), but use $(\oplus, \mathrm{XOR})$ not $(\vee, \mathrm{OR})$.

(c) If $f = \emptyset$ or $f = g$, return $\emptyset$. If $g = \emptyset$, return f. Otherwise, if $g_v < f_v$, set $g \leftarrow g_l$ and begin again. Otherwise

set $r_l \leftarrow \mathrm{BUTNOT}(f_l, g_l)$, $r_h \leftarrow \mathrm{BUTNOT}(f_h, g_h)$, if $f_v = g_v$;
set $r_l \leftarrow \mathrm{BUTNOT}(f_l, g)$, $r_h \leftarrow f_h$, increase `REF(`f_h`)` by 1, if $f_v < g_v$.

Then set $r \leftarrow \mathrm{ZUNIQUE}(f_v, r_l, r_h)$ and finish as usual.

200. If $f = \emptyset$, return g. If $f = h$, return $\mathrm{OR}(f, g)$. If $g = h$, return g. If $g = \emptyset$ or $f = g$, return $\mathrm{AND}(f, h)$. If $h = \emptyset$, return $\mathrm{BUTNOT}(g, f)$. If $f_v < g_v$ and $f_v < h_v$, set $f \leftarrow f_l$ and start over. If $h_v < f_v$ and $h_v < g_v$, set $h \leftarrow h_l$ and start over. Otherwise check the cache and proceed recursively as usual.

201. In applications of ZDDs where projection functions and/or the complementation operation are permitted, it's best to fix the set of Boolean variables at the beginning, when everything is being initialized. Otherwise, *every* external function in a ZDD base must change whenever a new variable enters the fray.

Suppose therefore that we've decided to deal with functions of $(x_1, \ldots, x_N)$, where N is prespecified. In answer 198, we let $f_v = N + 1$, not ∞, when $f = \emptyset$ or $f = \{\emptyset\}$. Then the tautology function $1 = \wp$ has the $(N+1)$-node ZDD (1) (2) $\cdots$ (N) $\boxed{\top}$, which we construct as soon as N is known. Let t_j be node (j) of this structure, with $t_{N+1} = \boxed{\top}$. The ZDD for x_j is now (1) $\cdots$ (j) — t_{j+1} $\boxed{\bot}$; thus the ZDD base for the set of all x_j will occupy $\binom{N+1}{2}$ nodes in addition to the representations of $\emptyset$ and $\wp$.

If N is small, all N projection functions can be prepared in advance. But N is large in many applications of ZDDs; and projection functions are rarely needed when "family algebra" is used to build the structures as in exercises 203–207. So it's generally best to wait until a projection function is actually required, before creating it.

Incidentally, the partial-tautology functions t_j can be used to speed up the synthesis operations of exercises 198–199: If $v = f_v \le g_v$ and $f = t_v$, we have $\mathrm{AND}(f,g) = g$, $\mathrm{OR}(f,g) = f$, and (if $v \le h_v$) also $\mathrm{MUX}(f,g,h) = h$, $\mathrm{MUX}(g,h,f) = \mathrm{OR}(g,h)$.

202. In the transmogrification step T4, change '$q_0 \leftarrow q_1 \leftarrow q$' to '$q_0 \leftarrow q$, $q_1 \leftarrow \emptyset$' and '$r_0 \leftarrow r_1 \leftarrow r$' to '$r_0 \leftarrow r$, $r_1 \leftarrow \emptyset$'. Also use ZUNIQUE instead of UNIQUE; within T4, this subroutine increases $\mathtt{REF}(p)$ by 1 if step U1 finds $q = \emptyset$.

A subtler change is needed to keep the partial-tautology functions of answer 201 up to date, because of their special meaning. Correct behavior is to keep t_u unchanged and set $t_v \leftarrow \mathtt{LO}(t_u)$.

203. (a) $f \sqcup g = \{\{1,2\},\{1,3\},\{1,2,3\},\{3\}\} = (e_1 \sqcup ((e_2 \sqcup (e_3 \cup \epsilon)) \cup e_3)) \cup e_3$; the other is $(e_1 \sqcup e_2) \cup \epsilon$, because $f \sqcap g = (e_1 \sqcup (e_2 \cup \epsilon)) \cup e_3 \cup \epsilon$ and $f \boxplus e_1 = e_1 \cup e_2 \cup e_3$.

(b) $(f \sqcup g)(z) = \exists x\, \exists y\, (f(x) \wedge g(y) \wedge (z \equiv x \vee y))$; $(f \sqcap g)(z) = \exists x\, \exists y\, (f(x) \wedge g(y) \wedge (z \equiv x \wedge y))$; $(f \boxplus g)(z) = \exists x\, \exists y\, (f(x) \wedge g(y) \wedge (z \equiv x \oplus y))$. Another formula is $(f \boxplus g)(z) = \bigvee\{f(z \oplus y) \mid g(y) = 1\} = \bigvee\{g(z \oplus x) \mid f(x) = 1\}$.

(c) Both (i) and (ii) are true; also $f \boxplus (g \cup h) = (f \boxplus g) \cup (f \boxplus h)$. Formula (iii) fails in general, although we do have $f \sqcup (g \sqcap h) \subseteq (f \sqcup g) \sqcap (f \sqcup h)$. Formula (iv) makes little sense; the right-hand side is $(f \sqcup f) \cup (f \sqcup h) \cup (g \sqcup f) \cup (g \sqcup h)$, by (i). Formula (v) is true because all three parts are $\emptyset$. And (vi) is true if and only if $f \ne \emptyset$.

(d) Only (ii) is always true. For (i), the condition should be $f \sqcap g \subseteq \epsilon$, since $f \sqcap g = \emptyset$ implies $f \perp g$. For (iii), notice that $|f \sqcup g| = |f \sqcap g| = |f \boxplus g| = 1$ whenever $|f| = |g| = 1$. Finally, in statement (iv), we do have $f \perp g \implies f \sqcup g = f \boxplus g$; but the converse fails when, say, $f = g = e_1 \cup \epsilon$.

(e) $f = \emptyset$ in (i) and $f = \epsilon$ in (ii); also $\epsilon \boxplus g = g$ for all g. There's no solution to (iii), because f would have to be $\{\{1,2,3,\ldots\}\}$ and we are considering only finite sets. But in the finite universe of answer 201 we have $f = \{\{1,\ldots,N\}\}$. (This family U has the property that $(f \boxplus U) \sqcup (g \boxplus U) = (f \sqcap g) \boxplus U$.) The general solution to (iv) is $f = e_1 \sqcup e_2 \sqcup f'$, where f' is an arbitrary family; similarly, the general solution to (v) is $f = (e_1 \sqcup f') \cup (e_2 \sqcup f'') \cup (e_1 \sqcup e_2 \sqcup (f' \cup f'' \cup f'''))$, where f', f'', and f''' are arbitrary. In (vi), $f = ((((e_1 \sqcup e_2) \cup \epsilon) \sqcup f') \cup ((e_1 \cup e_2) \sqcup f'')) \sqcup (e_3 \cup \epsilon)$, where $f' \cup f'' \perp e_1 \cup e_2 \cup e_3$; this representation follows from exercise 204(f). In (vii), $|f| = 1$. Finally, (viii) characterizes Horn functions (Theorem 7.1.1H).

204. (a) This relation is obvious from the definition. (Also $(f \cup g)/h \supseteq (f/h) \cup (g/h)$.)

(b) $f/e_2 = \{\{1\},\emptyset\} = e_1 \cup \epsilon$; $f/e_1 = e_2 \cup e_3$; $f/\epsilon = f$; hence $f/(e_1 \cup \epsilon) = e_2 \cup e_3$.

(c) Division by $\emptyset$ gives trouble, because *all* sets α belong to $f/\emptyset$. (But if we restrict consideration to families of subsets of $\{1,\ldots,N\}$, as in exercises 201 and 207, we have $f/\emptyset = \wp$; also $\wp/\wp = \epsilon$, and $f/\wp = \emptyset$ when $f \ne \wp$.) Clearly $f/\epsilon = f$. And $f/f = \epsilon$ when $f \ne \emptyset$. Finally, $(f \bmod g)/g = \emptyset$ when $g \ne \emptyset$, because $\alpha \in (f \bmod g)/g$ and $\beta \in g$ implies that $\alpha \cup \beta \in f$, $\alpha \in f/g$, and $\alpha \cup \beta \notin (f/g) \sqcup g$ — a contradiction.

(d) If $\beta \in g$, we have $\beta \cup \alpha \in f$ and $\beta \cap \alpha = \emptyset$ for all $\alpha \in f/g$; this proves the hint. Hence $f/g \subseteq f/(f/(f/g))$. Also $f/h \subseteq f/g$ when $h \supseteq g$, by (a); let $h = f/(f/g)$.

(e) Let $f/\!/g$ be the family in the new definition. Then $f/g \subseteq f/\!/g$, because $g \sqcup (f/g) \subseteq f$ and $g \perp (f/g)$. Conversely, if $\alpha \in f/\!/g$ and $\beta \in g$, we have $\alpha \in h$ for some h with $g \sqcup h \subseteq f$ and $g \perp h$; consequently $\alpha \cup \beta \in f$ and $\alpha \cap \beta = \emptyset$.

(f) If f has such a representation, we must have $g = f/e_j$ and $h = f \bmod e_j$. Conversely, those families satisfy $e_j \perp g \cup h$. (This law is the fundamental recursive

family algebra
partial-tautology
AND
OR
MUX
transmogrification
Horn functions
power set
recursive principle, underlying ZDDs+

principle underlying ZDDs — just as the unique representation $f = (x_j?\ g{:}\ h)$, with g and h independent of x_j, underlies BDDs.)

(g) Both true. (To prove them, represent f and g as in part (f).)

[R. K. Brayton and C. McMullen introduced the quotient and remainder operations in *Proc. Int. Symp. Circuits and Systems* (IEEE, 1982), 49–54, but in a slightly different context: They dealt with families of incomparable sets of subcubes.]

205. In all cases we construct a recursion based on exercise 204(f). For example, if $f_v = g_v = v$, we have $f \sqcup g = (\bar v?\ f_l \sqcup g_l{:}\ (f_l \sqcup g_h) \cup (f_h \sqcup g_l) \cup (f_h \sqcup g_h))$; $f \sqcap g = (\bar v?\ (f_l \sqcap g_l) \cup (f_l \sqcap g_h) \cup (f_h \sqcap g_l){:}\ f_h \sqcap g_h)$; $f \boxplus g = (\bar v?\ (f_l \boxplus g_l) \cup (f_h \boxplus g_h){:}\ (f_h \boxplus g_l) \cup (f_l \boxplus g_h))$.

(a) If $f_v < g_v$ or ($f_v = g_v$ and $f > g$), swap $f \leftrightarrow g$. If $f = \emptyset$, return f; if $f = \epsilon$, return g. If $f \sqcup g = r$ is in the memo cache, return r. If $f_v > g_v$, set $r_l \leftarrow \mathrm{JOIN}(f, g_l)$ and $r_h \leftarrow \mathrm{JOIN}(f, g_h)$; otherwise set $r_l \leftarrow \mathrm{JOIN}(f_l, g_l)$, $r_{lh} \leftarrow \mathrm{JOIN}(f_l, g_h)$, $r_{hl} \leftarrow \mathrm{JOIN}(f_h, g_l)$, $r_{hh} \leftarrow \mathrm{JOIN}(f_h, g_h)$, $r_h \leftarrow \mathrm{OROR}(r_{lh}, r_{hl}, r_{hh})$, and dereference r_{lh}, r_{hl}, r_{hh}. Finish with $r \leftarrow \mathrm{ZUNIQUE}(g_v, r_l, r_h)$; cache it and return it as in exercise 198.

(We could also compute r_h via the formula $\mathrm{OR}(r_{lh}, \mathrm{JOIN}(f_h, \mathrm{OR}(g_l, g_h)))$, or via $\mathrm{OR}(r_{hl}, \mathrm{JOIN}(\mathrm{OR}(f_l, f_h), g_h))$. Sometimes one way is much better than the other two.)

The DISJOIN operation, which produces the family of *disjoint* unions $\{\alpha \cup \beta \mid \alpha \in f,\ \beta \in g,\ \alpha \cap \beta = \emptyset\}$, is similar but with r_{hh} omitted.

(b) If $f_v < g_v$ or ($f_v = g_v$ and $f > g$), swap $f \leftrightarrow g$. If $f \le \epsilon$, return f. (We consider $\emptyset < \epsilon$ and $\epsilon <$ all others.) Otherwise, if $\mathrm{MEET}(f, g)$ hasn't been cached, there are two cases. If $f_v > g_v$, set $r_h \leftarrow \mathrm{OR}(g_l, g_h)$, $r \leftarrow \mathrm{MEET}(f, r_h)$, and dereference r_h; otherwise proceed analogously to (a) but with $l \leftrightarrow h$. Cache and return r as usual.

(c) This operation is similar to (a), but $r_l \leftarrow \mathrm{OR}(r_{ll}, r_{hh})$ and $r_h \leftarrow \mathrm{OR}(r_{lh}, r_{hl})$.

(d) First we implement the important simple cases f/e_v and $f \bmod e_v$:

$$\mathrm{EZDIV}(f,v) = \begin{cases} \text{If } f_v = v, \text{ return } f_h; \text{ if } f_v > v, \text{ return } \emptyset. \text{ Otherwise look for} \\ f/e_v = r \text{ in the cache; if it isn't present, compute it via} \\ r \leftarrow \mathrm{ZUNIQUE}(f_v, \mathrm{EZDIV}(f_l, v), \mathrm{EZDIV}(f_h, v)). \end{cases}$$

$$\mathrm{EZMOD}(f,v) = \begin{cases} \text{If } f_v = v, \text{ return } f_l; \text{ if } f_v > v, \text{ return } f. \text{ Otherwise look for} \\ f \bmod e_v = r \text{ in the cache; if it isn't present, compute it via} \\ r \leftarrow \mathrm{ZUNIQUE}(f_v, \mathrm{EZMOD}(f_l, v), \mathrm{EZMOD}(f_h, v)). \end{cases}$$

Now $\mathrm{DIV}(f, g) =$ "If $g = \emptyset$, see below; if $g = \epsilon$, return f. Otherwise, if $f \le \epsilon$, return $\emptyset$; if $f = g$, return ϵ. If $g_l = \emptyset$ and $g_h = \epsilon$, return $\mathrm{EZDIV}(f, g_v)$. Otherwise, if $f/g = r$ is in the memo cache, return r. Otherwise set $r_l \leftarrow \mathrm{EZDIV}(f, g_v)$, $r \leftarrow \mathrm{DIV}(r_l, g_h)$, and dereference r_l. If $r \ne \emptyset$ and $g_l \ne \emptyset$, set $r_h \leftarrow \mathrm{EZMOD}(f, g_v)$ and $r_l \leftarrow \mathrm{DIV}(r_h, g_l)$, dereference r_h, set $r_h \leftarrow r$ and $r \leftarrow \mathrm{AND}(r_l, r_h)$, dereference r_l and r_h. Insert $f/g = r$ in the memo cache and return r." Division by $\emptyset$ returns $\wp$ if there is a fixed universe $\{1, \ldots, N\}$ as in exercise 201. Otherwise it's an error (because the universal family $\wp$ doesn't exist).

(e) If $g = \emptyset$, return f. If $g = \epsilon$, return $\emptyset$. If $(g_l, g_h) = (\emptyset, \epsilon)$, return $\mathrm{EZMOD}(f, g_v)$. If $f \bmod g = r$ is cached, return it. Otherwise set $r \leftarrow \mathrm{DIV}(f, g)$ and $r_h \leftarrow \mathrm{JOIN}(r, g)$, dereference r, set $r \leftarrow \mathrm{BUTNOT}(f, r_h)$, and dereference r_h. Cache and return r.

[S. Minato gave $\mathrm{EZDIV}(f, v)$, $\mathrm{EZREM}(f, v)$, and $\mathrm{DELTA}(f, e_v)$ in his original paper on ZDDs. His algorithms for $\mathrm{JOIN}(f, g)$ and $\mathrm{DIV}(f, g)$ appeared in the sequel, *ACM/IEEE Design Automation Conf.* **31** (1994), 420–424.]

206. The upper bound $O(Z(f)^3 Z(g)^3)$ is not difficult to prove for cases (a) and (b), as well as $O(Z(f)^2 Z(g)^2)$ for case (c). But are there examples that take such a long time? And can the running time for (d) be exponential? All five routines seem to be reasonably fast in practice.

Brayton
McMullen
subcubes
clutters
OROR
dereference
disjoint unions, family of
dereference
$\wp$
Minato

207. If $f = e_{i_1} \cup \cdots \cup e_{i_l}$ and $k \geq 0$, let $\mathrm{SYM}(f, v, k)$ be the Boolean function that is true if and only if exactly k of the variables $\{x_{i_1}, \ldots, x_{i_l}\} \cap \{x_v, x_{v+1}, \ldots\}$ are 1 and $x_1 = \cdots = x_{v-1} = 0$. We compute $(e_{i_1} \cup \cdots \cup e_{i_l}) \S\, k$ by calling $\mathrm{SYM}(f, 1, k)$.

$\mathrm{SYM}(f, v, k) =$ "While $f_v < v$, set $f \leftarrow f_l$. If $f_v = N+1$ and $k > 0$, return $\emptyset$. If $f_v = N+1$ and $k = 0$, return the partial-tautology function t_v (see answer 201). If $f \S v \S k = r$ is in the cache, return r. Otherwise set $r \leftarrow \mathrm{SYM}(f, f_v+1, k)$. If $k > 0$, set $q \leftarrow \mathrm{SYM}(f_l, f_v+1, k-1)$ and $r \leftarrow \mathrm{ZUNIQUE}(f_v, r, q)$. While $f_v > v$, set $f_v \leftarrow f_v - 1$, increase `REF(r)` by 1, and set $r \leftarrow \mathrm{ZUNIQUE}(f_v, r, r)$. Put $f \S v \S k = r$ in the cache, and return r." The running time is $O((k+1)N)$. Notice that $\emptyset \S 0 = \wp$.

208. Just omit the factors $2^{v_{s-1}-1}$, $2^{v_l - v_k - 1}$, and $2^{v_h - v_k - 1}$ from steps C1 and C2. (And we get the generating function by setting $c_k \leftarrow c_l + z c_h$ in step C2; see exercise 25.) *The number of solutions equals the number of paths in the ZDD from the root to* $\boxed{\top}$.

209. Initially compute $\delta_n \leftarrow \bot$ and $\delta_j \leftarrow (\bar{x}_{j+1} \circ x_{j+1}) \bullet \delta_{j+1}$ for $n > j \geq 1$. Then, where answer 31 says '$\alpha \leftarrow (\bar{x}_j \circ x_j) \bullet \alpha$', change it to '$\alpha \leftarrow (\bar{x}_j \bullet \alpha) \circ (x_j \bullet \delta_j)$'. Also make the analogous changes with β and γ in place of α.

210. In fact, when $x = x_1 \ldots x_n$ we can replace νx in the definition of g by any linear function $c(x) = c_1 x_1 + \cdots + c_n x_n$, thus characterizing all of the optimal solutions to the general Boolean programming problem treated by Algorithm B.

For each branch node x of the ZDD, with fields $\mathtt{V}(x)$, $\mathtt{LO}(x)$, $\mathtt{HI}(x)$, we can compute its optimum value $\mathtt{M}(x)$ and new links $\mathtt{L}(x)$, $\mathtt{H}(x)$ as follows: Let $m_l = \mathtt{M}(\mathtt{LO}(x))$ and $m_h = c_{\mathtt{V}(x)} + \mathtt{M}(\mathtt{HI}(x))$, where $\mathtt{M}(\boxed{\bot}) = -\infty$ and $\mathtt{M}(\boxed{\top}) = 0$. Then $\mathtt{L}(x) \leftarrow \mathtt{LO}(x)$ if $m_l \geq m_h$, otherwise $\mathtt{L}(x) \leftarrow \boxed{\bot}$; $\mathtt{H}(x) \leftarrow \mathtt{HI}(x)$ if $m_l \leq m_h$, otherwise $\mathtt{H}(x) \leftarrow \boxed{\bot}$. The ZDD for g is obtained by reducing the `L` and `H` links accessible from the root. Notice that $Z(g) \leq Z(f)$, and the entire computation takes $O(Z(f))$ steps. (This nice property of ZDDs was pointed out by O. Coudert; see answer 237.)

211. Yes, unless the matrix has all-zero rows. Without such rows, in fact, the profile and z-profile of f satisfy $b_k \geq q_k - 1 \geq z_k$ for $0 \leq k < n$, because the only level-k subfunction independent of x_{k+1} is the constant 0.

212. The best alternative in the author's experiments was to make ZDDs for each term $T_j = S_1(X_j)$ in (129), using the algorithm of exercise 207, and then to AND them together. For example, in problem (128) we have $X_1 = \{x_1, x_2\}$, $X_2 = \{x_1, x_3, x_4\}$, $\ldots$, $X_{64} = \{x_{105}, x_{112}\}$; to make the term $S_1(X_2) = S_1(x_1, x_3, x_4)$, whose ZDD has 115 nodes, just form the 5-node ZDD for $e_1 \cup (e_3 \cup e_4)$ and compute $T_2 \leftarrow (e_1 \cup e_3 \cup e_4) \S 1$.

But in what order should the ANDs be done, after we've got the individual terms $T_1, \ldots, T_n$ of (129)? Consider problem (128). *Method 1:* $T_1 \leftarrow T_1 \wedge T_2$, $T_1 \leftarrow T_1 \wedge T_3$, $\ldots$, $T_1 \leftarrow T_1 \wedge T_{64}$. This "top-down" method fills in the upper levels first, and takes about 6.2 megamems. *Method 2:* $T_{64} \leftarrow T_{64} \wedge T_{63}$, $T_{64} \leftarrow T_{64} \wedge T_{62}$, $\ldots$, $T_{64} \leftarrow T_{64} \wedge T_1$. By filling in the lower levels first ("bottom-up"), the time goes down to about 1.75 megamems. *Method 3:* $T_2 \leftarrow T_2 \wedge T_1$, $T_4 \leftarrow T_4 \wedge T_3$, $\ldots$, $T_{64} \leftarrow T_{64} \wedge T_{63}$; $T_4 \leftarrow T_4 \wedge T_2$, $T_8 \leftarrow T_8 \wedge T_6$, $\ldots$, $T_{64} \leftarrow T_{64} \wedge T_{62}$; $T_8 \leftarrow T_8 \wedge T_4$, $T_{16} \leftarrow T_{16} \wedge T_{12}$, $\ldots$, $T_{64} \leftarrow T_{64} \wedge T_{60}$; $\ldots$; $T_{64} \leftarrow T_{64} \wedge T_{32}$. This "balanced" approach also takes about 1.75 megamems. *Method 4:* $T_{33} \leftarrow T_{33} \wedge T_1$, $T_{34} \leftarrow T_{34} \wedge T_2$, $\ldots$, $T_{64} \leftarrow T_{64} \wedge T_{32}$; $T_{49} \leftarrow T_{49} \wedge T_{33}$, $T_{50} \leftarrow T_{50} \wedge T_{34}$, $\ldots$, $T_{64} \leftarrow T_{64} \wedge T_{48}$; $T_{57} \leftarrow T_{57} \wedge T_{49}$, $T_{58} \leftarrow T_{58} \wedge T_{50}$, $\ldots$, $T_{64} \leftarrow T_{64} \wedge T_{56}$; $\ldots$; $T_{64} \leftarrow T_{64} \wedge T_{63}$. This is a much better way to balance the work, needing only about 850 kilomems. *Method 5:* An analogous balancing strategy that uses the ternary ANDAND operation turns out to be still better, costing just 675 kilomems. (In all five cases, add 190 kilomems for the time to form the 64 initial terms T_j.)

partial-tautology
power set
$\wp$
generating function from ZDD
solutions
linear function
Boolean programming
Coudert
profile
z-profile
Knuth
top-down
bottom-up
balanced
ternary ANDAND

Incidentally, we can reduce the ZDD size from 2300 to 1995 by insisting that $x_1 = 0$ and $x_2 = 1$ in (128) and (129), because the "transpose" of every covering is another covering. This idea does not, however, reduce the running time substantially.

The rows of (128) appear in decreasing lexicographic order, and that may not be ideal. But dynamic variable ordering is unhelpful when so many variables are present. (Sifting reduces the size from 2300 to 1887, but takes a *long* time.)

Further study, with a variety of exact cover problems, would clearly be desirable.

213. It is a bipartite graph with 30 vertices in one part and 32 in the other. (Think of a chessboard as a *checkerboard*: Every domino joins a white square to a black square, and we've removed two black squares.) A row sum of $(1,\ldots,1,1,*,*)$ has 1s in at least 31 "white" positions, so its last two coordinates must be either $(2,1)$ or $(3,2)$.

214. Add further constraints to the covering condition (128), namely $\bigwedge_{j=1}^{14} S_{\geq 1}(Y_j)$, where Y_j is the set of x_i that cross the jth potential fault line. (For example, $Y_1 = \{x_2, x_4, x_6, x_8, x_{10}, x_{12}, x_{14}, x_{15}\}$ is the set of ways to place a domino vertically in the top two rows of the board; each $|Y_j| = 8$.) The resulting ZDD has 9812 nodes, and characterizes 25,506 solutions. Incidentally, the BDD size is 26622. [Faultfree domino tilings of $m \times n$ boards exist if and only if mn is even, $m \geq 5$, $n \geq 5$, and $(m,n) \neq (6,6)$; see R. L. Graham, *The Mathematical Gardner* (Wadsworth International, 1981), 120–126. The solution in (127) is the only 8×8 example that is symmetric under both horizontal and vertical reflection; see Fig. 29(b) for symmetry under 90° rotation.]

215. This time we add the constraints $\bigwedge_{j=1}^{49} S_{\geq 1}(Z_j)$, where Z_j is the set of four placements x_i that surround an internal corner point. (For example, $Z_1 = \{x_1, x_2, x_4, x_{16}\}$.) These constraints reduce the ZDD size to 66. There are just two solutions, one the transpose of the other, and they can readily be found by hand. [See Y. Kotani, *Puzzlers' Tribute* (A. K. Peters, 2002), 413–420.]

Conjecture: The generating function for the number of $m \times n$ tatami tilings, when $n \geq m - 2 \geq 0$ and m is even, is $(1+z)^2(z^{m-2} + z^m)/(1 - z^{m-1} - z^{m+1})$.

216. (a) Assign three variables (a_i, b_i, c_i) to each row of (128), corresponding to the domino's color if row i is chosen. Every branch node of the ZDD for f in (129) now becomes three branch nodes. We can take advantage of symmetry under transposition by replacing f by $f \wedge x_2$; this reduces the ZDD size from 2300 to 1995, which grows to 5981 when each branch node is triplicated.

Now we AND in the adjacency constraints, for all 682 cases $\{i, i'\}$ where rows i and i' are adjacent domino positions. Such constraints have the form $\neg\big((a_i \wedge a_{i'}) \vee (b_i \wedge b_{i'}) \vee (c_i \wedge c_{i'})\big)$, and we apply them bottom-up as in Method 2 of answer 212. This computation inflates the ZDD until it reaches more than 800 thousand nodes; but eventually it settles down and ends up with size 584,205.

The desired answer turns out to be 13,343,246,232 (which, of course, is a multiple of $3! = 6$, because each permutation of the three colors yields a different solution).

(b) This question is distinct from part (a), because many coverings (including Fig. 29(b)) can be 3-colored in several ways; we want to count them only once.

Suppose $f(a_1, b_1, c_1, \ldots, a_m, b_m, c_m) = f(x_1, \ldots, x_{3m})$ is a function with $a_i = x_{3i-2}$, $b_i = x_{3i-1}$, and $c_i = x_{3i}$, such that $f(x_1, \ldots, x_{3m}) = 1$ implies $a_i + b_i + c_i \leq 1$ for $1 \leq i \leq m$. Let's define the *uncoloring* $\$f$ of f to be

$$\$f(x_1,\ldots,x_m) = \exists y_1 \cdots \exists y_{3m}\big(f(y_1,\ldots,y_{3m}) \\ \wedge (x_1 = y_1 + y_2 + y_3) \wedge \cdots \wedge (x_m = y_{3m-2} + y_{3m-1} + y_{3m})\big).$$

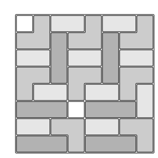
transpose
symmetry breaking
lexicographic order
dynamic variable ordering
Sifting
bipartite graph
checkerboard
BDD versus ZDD
Graham
Kotani
symmetry under transposition
bottom-up
uncoloring

BDDs instead of ZDDs
Knuth
bottom-up
balancing
ZDD versus BDD
dancing links

A straightforward recursive subroutine will compute the ZDD for $\$f$ from the ZDD for f. This process transforms the 584,205 nodes obtained in part (a) into a ZDD of size 33,731, from which we deduce the answer: 3,272,232.

(The running time is 1.2 gigamems for part (a), plus 1.3 gigamems to uncolor; the total memory requirement is about 44 megabytes. A similar computation based on BDDs instead of ZDDs cost $13.6+1.5$ gigamems and occupied 185 megabytes.)

217. The separation condition adds 4198 further constraints of the form $\neg(x_i \wedge x_{i'})$, where rows i and i' specify adjacent placements of congruent pieces. Applying these constraints while also evaluating $\bigwedge_{j=1}^{468} S_1(X_j)$ turned out to be a bad idea, in the author's experiments; even worse was an attempt to construct a separate ZDD for the new constraints alone. Much better was to build the 512,227-node ZDD as before, then to incorporate the new constraints one by one, first constraining the variables at the lowest levels. The resulting ZDD of size 31,300,699 was finally completed after 286 gigamems of work, proving that exactly 7,099,053,234,102 separated solutions exist.



We might also ask for *strongly* separated solutions, where congruent pieces are not allowed to touch even at their corners; this requirement adds 1948 more constraints. There are 42,159,777,732 strongly separated coverings, found after 304 gigamems with a ZDD of size 20,659,124. (Other methods may well be better than ZDDs for this problem.)

218. This is an exact cover problem. For example, the matrix when $n=3$ is

001001010	(−−2−−2)
010001001	(−3−−−3)
010010010	(−2−−2−)
010100100	(−1−1−−)
100010001	(3−−−3−)
100100010	(2−−2−−)
101000100	(1−1−−−)

and in general there are $3n$ columns and $\binom{2n-1}{2}-\binom{n}{2}$ rows. Consider the case $n=12$: The ZDD on 187 variables has 192,636 nodes. It can be found with a cost of 300 megamems, using Method 4 of answer 212 (binary balancing); Method 5 turns out to be 25% slower than Method 4 in this case. The BDD is much larger (2,198,195 nodes) and it costs more than 900 megamems.

Thus the ZDD is clearly preferable to the BDD for this problem, and it identifies the $L_{12}=108{,}144$ solutions with reasonable efficiency. (However, the "dancing links" technique of Section 7.2.2 is about four times faster, and it needs far less memory.)

219. (a) 1267; (b) 2174; (c) 2958; (d) 3721; (e) 4502. (To form the ZDD for $\mathtt{WORDS}(n)$ we do $n-1$ ORs of the 7-node ZDDs for $\mathtt{w}_1\sqcup\mathtt{h}_2\sqcup\mathtt{i}_3\sqcup\mathtt{c}_4\sqcup\mathtt{h}_5$, $\mathtt{t}_1\sqcup\mathtt{h}_2\sqcup\mathtt{e}_3\sqcup\mathtt{r}_4\sqcup\mathtt{e}_5$, etc.)

220. (a) There is one a_2 node for the descendants of each initial letter that can be followed by `a` in the second position (`aargh`, `babel`, ..., `zappy`); 23 letters qualify, all except `q`, `u`, and `x`. And there's one b_2 node for each initial letter that can be followed by `b` (`abbey`, `ebony`, `oboes`). However, the actual rule isn't so simple; for example, there are three z_2 nodes, not four, because of sharing between `czars` and `tzars`.

(b) There's no v_5 because no five-letter word ends with `v`. (The SGB collection doesn't include `arxiv` or `webtv`.) The three nodes for w_5 arise because one stands for cases where the letters $< w_5$ must be followed by `w` (`aglo` and many others); another node stands for cases where either `w` or `y` must follow (`stra`, or `resa`, or when we've seen `allo` but not `allot`); and there's also a w_5 node for the case when `unse` is not followed by `e` or `t`, because it must then be followed by either `w` or `x`. Similarly, the two nodes for x_5 represent the cases where `x` is forced, or where the last letter must be either `x` or `y` (following `rela`). There's only one y_5 node, because no four letters can be followed by both `y` and `z`. Of course there's just one z_5 node, and two sinks.

zead
inclusion and exclusion
Knuth
joke

221. We compute, for every possible zead ζ, the probability that ζ will occur, and sum over all ζ. For definiteness, consider a zead that corresponds to branching on r_3, and suppose it represents a subfamily of 10 three-letter suffixes. There are exactly $\binom{6084}{10} - \binom{5408}{10} \approx 1.3 \times 10^{31}$ such zeads, and by the principle of inclusion and exclusion they each arise with probability $\sum_{k\ge1}\binom{676}{k}(-1)^{k+1}\binom{11881376-6084k}{5757-10k}/\binom{11881376}{5757} \approx 2.5 \times 10^{-32}$. [*Hint:* $|\{r,s,t,u,v,w,x,y,z\}| = 9$, $676 = 26^2$, and $6084 = 9 \times 26^2$.] Thus such zeads contribute about 0.33 to the total. The r_3-zeads for subfamilies of sizes 1, 2, 3, 4, 5, …, contribute approximately 11.5, 32.3, 45.1, 41.9, 29.3, …, by a similar analysis; so we expect about 188.8 branches on r_3 altogether, on average. The grand total

$$\sum_{l=1}^{5}\sum_{j=1}^{26}\sum_{s=1}^{5757}\left(\binom{26^{5-l}(27-j)}{s} - \binom{26^{5-l}(26-j)}{s}\right) \\ \times \sum_{k=1}^{\infty}\binom{26^{l-1}}{k}(-1)^{k+1}\binom{26^5 - 26^{5-l}(27-j)k}{5757-sk}\Big/\binom{26^5}{5757},$$

plus 2 for the sinks, comes to ≈ 7151.986. The average z-profile is $\approx$ (1.00, …, 1.00; 25.99, …, 25.99; 188.86, …, 171.43; 86.31, …, 27.32; 3.53, …, 1.00; 2).

222. (a) It's the set of all subsets of the words of F. (There are 50,569 such subwords, out of $27^5 = 14{,}348{,}907$ possibilities. They are described by a ZDD of size 18,784, constructed from F and $\wp$ via answer 205(b) at a cost of about 15 megamems.)

(b) This formula gives the same result as $F \sqcap \wp$, because every member of F contains exactly one element of each X_j. But the computation turns out to be much slower — about 370 megamems — in spite of the fact that $Z(X) = 132$ is almost as small as $Z(\wp) = 131$. (Notice that $|\wp| = 2^{130}$ while $|X| = 26^5 \approx 2^{23.5}$.)

(c) $(F/P) \sqcup P$, where $P = \mathtt{t}_1 \sqcup \mathtt{u}_3 \sqcup \mathtt{h}_5$ is the pattern. (The words are `touch`, `tough`, `truth`. This computation costs about 3000 mems with the algorithms of answer 205.) Other contenders for simple formulas are $F \cap Q$, where Q describes the admissible words. If we set $Q = \mathtt{t}_1 \sqcup X_2 \sqcup \mathtt{u}_3 \sqcup X_4 \sqcup \mathtt{h}_5$, we have $Z(Q) = 57$ and the cost once again is $\approx 3000\mu$. With $Q = (\mathtt{t}_1 \cup \mathtt{u}_3 \cup \mathtt{h}_5)\,\S\,3$, on the other hand, we have $Z(Q) = 132$ and the cost rises to about 9000 mems. (Here $|Q|$ is 26^2 in the first case, but 2^{127} in the second — *reversing* any intuition gained from (a) and (b)! Go figure.)

(d) $F \cap ((V_1 \cup \cdots \cup V_5)\,\S\,k)$. The number of such words is (24, 1974, 3307, 443, 9, 0) for $k = (0, \ldots, 5)$, respectively, from ZDDs of sizes (70, 1888, 3048, 686, 34, 1). ("See exercise 7–34 for the words $F \bmod \mathtt{y}_1 \bmod \mathtt{y}_2 \bmod \cdots \bmod \mathtt{y}_5$," said the author `wryly`.)

(e) The desired patterns satisfy $P = (F \sqcap \wp) \cap Q$, where $Q = ((X_1 \cup \cdots \cup X_5)\,\S\,3)$. We have $Z(Q) = 386$, $Z(P) = 14221$, and $|P| = 19907$.

(f) The formula for this case is trickier. First, $P_2 = F \sqcap F$ gives F together with all patterns satisfied by two distinct words; we have $Z(P_2) = 11289$, $|P_2| = 21234$, and $|P_2 \cap Q| = 7753$. But $P_2 \cap Q$ is *not* the answer; for example, it omits the pattern `*atc*`, which occurs eight times but only in the context `*atch`. The correct answer is given by $P_2' \cap Q$, where $P_2' = (P_2 \setminus F) \sqcap \wp$. Then $Z(P_2') = 8947$, $Z(P_2' \cap Q) = 7525$, $|P_2' \cap Q| = 10472$.

(g) $G_1 \cup \cdots \cup G_5$, where $G_j = (F/(\mathtt{b}_j \cup \mathtt{o}_j)) \sqcup \mathtt{b}_j$. The answers are `bared`, `bases`, `basis`, `baths`, `bobby`, `bring`, `busts`, `herbs`, `limbs`, `tribs`.

(h) Patterns that admit all vowels in second place: `b*lls`, `b*nds`, `m*tes`, `p*cks`.

(i) The first gives all words whose middle three letters are vowels. The second gives all patterns with first and last letter specified, for which there's at least one match with three vowels inserted. There are 30 solutions to the first, but only 27 to the second (because, e.g., `louis` and `luaus` yield the same pattern). Incidentally, the complementary family $\wp \setminus F$ has $2^{130} - 5757$ members, and 46316 nodes in its ZDD.

223. (a) $d(\alpha,\mu) + d(\beta,\mu) + d(\gamma,\mu) = 5$, since $d(\alpha,\mu) = [\alpha_1 \neq \mu_1] + \cdots + [\alpha_5 \neq \mu_5]$.

(b) Given families f, g, h, the family $\{\mu \mid \mu = \langle\alpha\beta\gamma\rangle$ for some $\alpha \in f$, $\beta \in g$, $\gamma \in h$ with $\alpha \neq \mu$, $\beta \neq \mu$, $\gamma \neq \mu$, and $\alpha \cap \beta \cap \gamma = \emptyset\}$ can be defined recursively to allow ZDD computation, if we consider eight variants in which subsets of the inequality constraints are relaxed. In the author's experimental system, the ZDDs for medians of `WORDS`(n) for $n =$ (100, 1000, 5757) have respectively (595, 14389, 71261) nodes and characterize (47, 7310, 86153) five-letter solutions. Among the 86153 medians when $n = 5757$ are `chads`, `stent`, `blogs`, `ditzy`, `phish`, `bling`, and `tetch`; in fact, `tetch` = ⟨`fetch teach total`⟩ arises already when $n = 1000$. (The running times of about (.01, 2, 700) gigamems, respectively, were not especially impressive; ZDDs are probably not the best tool for this problem. Still, the programming was instructive.)

(c) When $n = 100$, exactly $(1, 14, 47)$ medians of `WORDS`(n) belong to `WORDS`(100), `WORDS`(1000), `WORDS`(5757), respectively; the solution with most common words is `while` = ⟨`white whole still`⟩. When $n = 1000$, the corresponding numbers are (38, 365, 1276); and when $n = 5757$ they are (78, 655, 4480). The most common English words that *aren't* medians of three other English words are `their`, `first`, and `right`.

224. Every arc $u \longrightarrow v$ of the dag corresponds to a vertex v of the forest. The ZDD has exactly one branch node for every arc. The LO pointer of that node leads to the right sibling of the corresponding vertex v, or to $\boxed{\bot}$ if v has no right sibling. The HI pointer leads to the left child of v, or to $\boxed{\top}$ if v is a leaf. The arcs can be ordered in many ways (e.g., preorder, postorder, level order), without changing this ZDD.

225. As in exercise 55, we try to number the vertices in such a way that the "frontier" between early and late vertices remains fairly small; then we needn't remember too much about what decisions were made on the early vertices. In the present case we also want the source vertex s to be number 1.

In answer 55, the relevant state from previous branches corresponded to an equivalence relation (a set partition); but now we express it by a table $mate[i]$ for $j \le i \le l$, where $j = u_k$ is the smaller vertex of the current edge u_k — v_k and where $l = \max\{v_1,\ldots,v_{k-1}\}$. Let $mate[i] = i$ if vertex i is untouched so far; let $mate[i] = 0$ if vertex i has been touched twice already. Otherwise $mate[i] = r$ and $mate[r] = i$, if previous edges form a simple path with endpoints $\{i, r\}$. Initially we set $mate[i] \leftarrow i$ for $1 \le i \le n$, except that $mate[1] \leftarrow t$ and $mate[t] \leftarrow 1$. (If $t > l$, the value of $mate[t]$ need not be stored, because it can be determined from the values of $mate[i]$ for $j \le i \le l$.)

Let $j' = u_{k+1}$ and $l' = \max\{v_1,\ldots,v_k\}$ be the values of j and l after edge k has been considered; and suppose $u_k = j$, $v_k = m$, $mate[j] = \hat{\jmath}$, $mate[m] = \hat{m}$. We cannot choose edge j — m if $\hat{\jmath} = 0$ or $\hat{m} = 0$. Otherwise, if $\hat{\jmath} \neq m$, the new $mate$ table after choosing edge j — m can be computed by doing the assignments $mate[j] \leftarrow 0$, $mate[m] \leftarrow 0$, $mate[\hat{\jmath}] \leftarrow \hat{m}$, $mate[\hat{m}] \leftarrow \hat{\jmath}$ (in that order).

Otherwise we have $\hat{\jmath} = m$ and $\hat{m} = j$; we must contemplate the endgame. Let i be the smallest integer such that $i > j$, $i \neq m$, and either $i > l'$ or $mate[i] \neq 0$ and $mate[i] \neq i$. The new state after choosing edge j — m is $\emptyset$ if $i \le l'$, otherwise it is ϵ.

complementary family
Knuth
left-child/right-sibling links
right-sibling/left-child links
frontier

Hamiltonian
generating function
standard deviation
variance

Whether or not the edge is chosen, the new state will be $\emptyset$ if $mate[i] \neq 0$ and $mate[i] \neq i$ for some i in the range $j \le i < j'$.

For example, here are the first steps for paths from 1 to 9 in a 3×3 grid (see (132)):

k	j	l	m	$mate[1] \ldots mate[9]$	$\hat{\jmath}$	$\hat{m}$	$mate'[1] \ldots mate'[9]$
1	1	1	2	9 2 3 4 5 6 7 8 1	9	2	0 9 3 4 5 6 7 8 2
2	1	2	3	9 2 3 4 5 6 7 8 1	9	3	0 2 9 4 5 6 7 8 3
2	1	2	3	0 9 3 4 5 6 7 8 2	0	3	—
3	2	3	4	0 2 9 4 5 6 7 8 3	2	4	0 4 9 2 5 6 7 8 3
3	2	3	4	0 9 3 4 5 6 7 8 2	9	4	0 0 3 9 5 6 7 8 4

where $mate'$ describes the next state if edge j — m is chosen. The state transitions $mate_{j..l} \mapsto mate'_{j'..l'}$ are $9 \mapsto (\overline{12}?\ 92{:}\ 09)$; $92 \mapsto (\overline{13}?\ \emptyset{:}\ 29)$; $09 \mapsto (\overline{13}?\ 93{:}\ \emptyset)$; $29 \mapsto (\overline{24}?\ 294{:}\ 492)$; $93 \mapsto (\overline{24}?\ 934{:}\ 039)$.

After all reachable states have been found, the ZDD can be obtained by reducing equivalent states, using a procedure like Algorithm R. (In the 3×3 grid problem, 57 branch nodes are reduced to 28, plus two sinks. The 22-branch ZDD illustrated in the text was obtained by subsequently optimizing with exercise 197.)

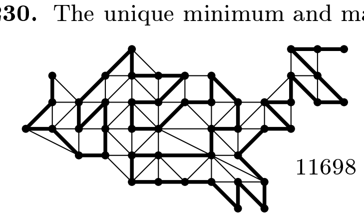
226. Just omit the initial assignments '$mate[1] \leftarrow t$, $mate[t] \leftarrow 1$.'

227. Change the test '$mate[i] \neq 0$ and $mate[i] \neq i$' to just '$mate[i] \neq 0$' in two places. Also, change '$i \le l'$' to '$i \le n$'.

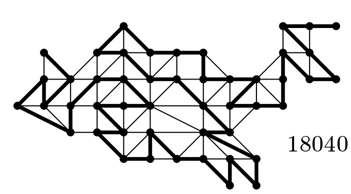
228. Use the previous answer with the following further changes: Add a dummy vertex $d = n+1$, with new edges v — d for all $v \neq s$; accepting this new edge will mean "end at v." Initialize the *mate* table with $mate[1] \leftarrow d$, $mate[d] \leftarrow 1$. Leave d out of the maximization when calculating l and l'. When beginning to examine a stored *mate* table, start with $mate[d] \leftarrow 0$ and then, if encountering $mate[i] = d$, set $mate[d] \leftarrow i$.

229. 149,692,648,904 of the latter paths go from `VA` to `MD`; graph (133) omits `DC`. (However, the graphs of (18) have fewer *Hamiltonian* paths than (133), because (133) has 1,782,199 Hamiltonian paths from `CA` to `ME` that do not go from `VA` to `MD`.)

230. The unique minimum and maximum routes from `ME` both end at `WA`:



11698 miles;



18040 miles.

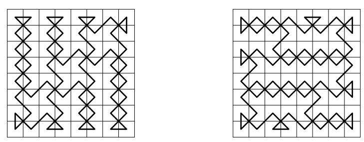
Let $g(z) = \sum z^{\text{miles}(r)}$, summed over all routes r. The average cost, $g'(1)/g(1) = 1022014257375/68656026 \approx 14886.01$, can be computed rapidly as in answer 29.

(Similarly, $g''(1) = 15243164303013274$, so the standard deviation is ≈ 666.2.)

231. The algorithm of answer 225 gives a proto-ZDD with 8,062,831 branch nodes; it reduces to a ZDD with 3,024,214 branches. The number of solutions, via answer 208, is 50,819,542,770,311,581,606,906,543.

232. With answer 227 we find h = 721,613,446,615,109,970,767 Hamiltonian paths from a corner to its horizontal neighbor, and d = 480,257,285,722,344,701,834 of them to its diagonal neighbor; in both cases the relevant ZDD has about 1.3 million nodes. The number of oriented Hamiltonian cycles is $2h + d$ = 1,923,484,178,952,564,643,368. (Divide by 2 to get the number of *undirected* Hamiltonian cycles.)

Essentially only two king's tours achieve the maximal length $8 + 56\sqrt{2}$:


.

233. A similar procedure can be used but with $mate[i] = r$ and $mate[r] = -i$ when the previous choices define an oriented path from i to r. Process all arcs $u_k \longrightarrow v_k$ and $u_k \longleftarrow v_k$ consecutively when $u_k = j < v_k = m$. Define $\hat{\jmath} = -j$ if $mate[j] = j$, otherwise $\hat{\jmath} = mate[j]$. Choosing $j \longrightarrow m$ is illegal if $\hat{\jmath} \geq 0$ or $\hat{m} \leq 0$. The updating rule for that choice, when legal, is: $mate[j] \leftarrow 0$, $mate[m] \leftarrow 0$, $mate[-\hat{\jmath}] \leftarrow \hat{m}$, $mate[\hat{m}] \leftarrow \hat{\jmath}$.

234. The 437 oriented cycles can be represented by a ZDD of ≈ 800 nodes. The shortest are, of course, `AL` $\longrightarrow$ `LA` $\longrightarrow$ `AL` and `MN` $\longrightarrow$ `NM` $\longrightarrow$ `MN`. There are 37 of length 17 (the maximum), such as (`ALARINVTNMIDCOKSC`) — i.e., `AL` $\longrightarrow$ `LA` $\longrightarrow \cdots \longrightarrow$ `SC` $\longrightarrow$ `CA` $\longrightarrow$ `AL`.

Incidentally, the directed graph in question is the arc-digraph D^* of the digraph D on 26 vertices $\{$`A`, `B`, $\ldots$, `Z`$\}$ whose 49 arcs are `A` $\longrightarrow$ `L`, `A` $\longrightarrow$ `R`, $\ldots$, `W` $\longrightarrow$ `Y`. Every oriented walk of D^* is an oriented walk of D, and conversely (see exercise 2.3.4.2–21); but the oriented *cycles* of D^* are not necessarily simple in D. In fact, D has only 37 oriented cycles, the longest of which is unique: (`ARINMOKSDC`).

If we extend consideration to the 62 postal codes in exercise 7–54(c), the number of oriented cycles rises to 38336, including the unique 1-cycle (`A`), as well as 192 that have length 23, such as (`APRIALASCTNMNVINCOKSDCA`). About 17000 ZDD nodes suffice to characterize the entire family of oriented cycles in this case.

235. The digraph has 7912 arcs; but we can prune them dramatically by removing arcs from vertices of in-degree zero, or arcs to vertices of out-degree zero. For example, `owner` $\longrightarrow$ `nerdy` goes away, because `nerdy` is a dead end; in fact, all successors of `owner` are likewise eliminated, so `crown` is out too. Eventually we're left with only 112 arcs among 85 words, and the problem can basically be done by hand.

There are just 74 oriented cycles. The unique shortest one, `slant` $\longrightarrow$ `antes` $\longrightarrow$ `tesla` $\longrightarrow$ `slant`, can be abbreviated to '(`slante`)' as in the previous answer. The two longest are $(\alpha\omega)$ and $(\beta\omega)$, where $\alpha =$ `picastepsomaso`, $\beta =$ `pointrotherema`, and $\omega =$ `nicadrearedidoserumorelicitеslabsitaresetuplenactoricedarerunichesto`.

236. (a) Suppose $\alpha \in f$ and $\beta \in g$. If $\alpha \subseteq \beta$, then $\alpha \in f \sqcap g$. If $\alpha \cap \beta \in f$, then $\alpha \cap \beta \notin f \nearrow g$. A similar argument, or the use of part (b), shows that $f \searrow g = f \setminus (f \sqcup g)$.

Notes: The complementary operations "$f \nwarrow g = f \setminus (f \searrow g) = \{\alpha \in f \mid \alpha \supseteq \beta$ for some $\beta \in g\}$" for supersets, and "$f \swarrow g = f \setminus (f \nearrow g) = \{\alpha \in f \mid \alpha \subseteq \beta$ for some $\beta \in g\}$" for subsets, are also important in applications. They were omitted from this exercise only because five operations are already rather intimidating. The superset operation was introduced by O. Coudert, J. C. Madre, and H. Fraisse [*ACM/IEEE Design Automation Conference* **30** (1993), 625–630]. The identity $f \nwarrow g = f \cap (f \sqcup g)$ was noted by H. G. Okuno, S. Minato, and H. Isozaki [*Information Processing Letters* **66** (1998), 195–199], who also listed several of the laws in (d).

(b) Elementary set theory suffices. (The first six identities appear in pairs, each of which is equivalent to its mate. Strictly speaking, f^C involves infinite sets, and U is the AND of infinitely many variables; but the formulas hold in any finite universe. Notice that, when cast in the language of Boolean functions, $f^C(x) = f(\bar{x})$ is the complement of f^D, the Boolean dual; see exercise 7.1.1–2. Is there any use for the dual of $f^\sharp$, namely $\{\alpha \mid \beta \in f$ implies $\alpha \cup \beta \neq U\}^\uparrow$? If so, we might denote it by $f^\flat$.)

arc-digraph
notation $f \nwarrow g$
notation $f \swarrow g$
Coudert
Madre
Fraisse
Okuno
Minato
Isozaki
Boolean functions versus families of sets
Boolean dual
notation

(c) All true except (ii), which should have said that $x_1^\uparrow = x_1^{C\downarrow C} = \bar{x}_1^{\downarrow C} = \epsilon^C = U$.

(d) The "identities" to cross out here are (ii), (viii), (ix), (xiv), and (xvi); the others are worth remembering. Regarding (ii)–(vi), notice that $f = f^\uparrow$ if and only if $f = f^\downarrow$, if and only if f is a clutter. Formula (xiv) should be $f \searrow g^\downarrow = f \searrow g$, the dual of (xiii). Formula (xvi) is almost right; it fails only when $f = \emptyset$ or $g = \emptyset$. Formula (ix) is perhaps the most interesting: We actually have $f^{\sharp\sharp} = f$ if and only if f is a clutter.

(e) Assuming that the universe of all vertices is finite, we have (i) $f = \wp \searrow g$ and (ii) $g = (\wp \setminus f)^\downarrow$, where $\wp$ is the universal family of exercises 201 and 222, because g is the family of minimal dependent sets. (Purists should substitute $\wp_V = \bigsqcup_{v\in V}(\epsilon \cup e_v)$ for $\wp$ in these formulas. The same relations hold in any hypergraph for which no edge is contained in another.)

clutter
power set
$\wp$
hypergraph
recurrences
Coudert
Knuth
contiguous USA
caching
garbage-collection

237. MAXMAL(f) = "If $f = \emptyset$ or $f = \epsilon$, return f. If $f^\uparrow = r$ is cached, return r. Otherwise set $r \leftarrow$ MAXMAL(f_l), $r_h \leftarrow$ MAXMAL(f_h), $r_l \leftarrow$ NONSUB(r, r_h), dereference r, and $r \leftarrow$ ZUNIQUE(f_v, r_l, r_h); cache and return r."

MINMAL(f) = "If $f = \emptyset$ or $f = \epsilon$, return f. If $f^\downarrow = r$ is cached, return r. Otherwise set $r_l \leftarrow$ MINMAL(f_l), $r \leftarrow$ MINMAL(f_h), $r_h \leftarrow$ NONSUP(r, r_l), dereference r, and $r \leftarrow$ ZUNIQUE(f_v, r_l, r_h); cache and return r."

NONSUB(f, g) = "If $g = \emptyset$, return f. If $f = \emptyset$ or $f = \epsilon$ or $f = g$, return $\emptyset$. If $f \nearrow g = r$ is cached, return r. Otherwise represent f and g as in (52). If $v < g_v$, set $r_l \leftarrow$ NONSUB(f_l, g), $r_h \leftarrow f_h$, and increase `REF(`f_h`)` by 1; otherwise set $r_h \leftarrow$ NONSUB(f_l, g_l), $r \leftarrow$ NONSUB(f_l, g_h), $r_l \leftarrow$ AND(r, r_h), dereference r and r_h, and set $r_h \leftarrow$ NONSUB(f_h, g_h). Finally $r \leftarrow$ ZUNIQUE(v, r_l, r_h); cache and return r."

NONSUP(f, g) = "If $g = \emptyset$, return f. If $f = \emptyset$ or $g = \epsilon$ or $f = g$, return $\emptyset$. If $f_v > g_v$, return NONSUP(f, g_l). If $f \searrow g = r$ is cached, return r. Otherwise set $v = f_v$. If $v < g_v$, set $r_l \leftarrow$ NONSUP(f_l, g) and $r_h \leftarrow$ NONSUP(f_h, g); otherwise set $r_l \leftarrow$ NONSUP(f_h, g_h), $r \leftarrow$ NONSUP(f_h, g_l), $r_h \leftarrow$ AND(r, r_l), dereference r and r_l, and set $r_l \leftarrow$ NONSUP(f_l, g_l). Finally $r \leftarrow$ ZUNIQUE(v, r_l, r_h); cache and return r."

CROSS(f) = "If $f = \emptyset$, return ϵ. If $f = \epsilon$, return $\emptyset$. If $f^\sharp = r$ is cached, return r. Otherwise set $r \leftarrow$ OR(f_l, f_h), $r_l \leftarrow$ CROSS(r), dereference r, $r \leftarrow$ CROSS(f_l), $r_h \leftarrow$ NONSUP(r, r_l), dereference r, and $r \leftarrow$ ZUNIQUE(f_v, r_l, r_h); cache and return r."

As in exercise 206, the worst-case running times of these routines are unknown. Although NONSUB and NONSUP can be computed via JOIN or MEET and BUTNOT, by exercise 236(a), this direct implementation tends to be faster. It may be preferable to replace '$f = \epsilon$' by '$\epsilon \in f$' in MINMAL and CROSS; also '$g = \epsilon$' by $\epsilon \in g$' in NONSUP.

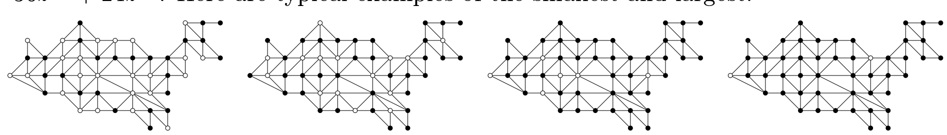
[Olivier Coudert introduced and implemented the operators $f^\uparrow$, $f \nearrow g$, and $f \searrow g$ in *Proc. Europ. Design and Test Conf.* (IEEE, 1997), 224–228. He also gave a recursive implementation of the interesting operator $f \odot g = (f \sqcup g)^\uparrow$; however, in the author's experiments, much better results have been obtained without it. For example, if f is the 177-node ZDD for the independent sets of the contiguous USA, the operation $g \leftarrow$ JOIN(f, f) costs about 350 kilomems and $h \leftarrow$ MAXMAL(g) costs about 3.6 megamems; but more than 69 *giga*mems are needed to compute $h \leftarrow$ MAXJOIN(f, f) all at once. Improved caching and garbage-collection strategies may, of course, change the picture.]

238. We can compute the 177-node ZDD for the family f of independent sets, using the ordering (104), in two ways: With Boolean algebra (67), $f = \neg \bigvee_{u - v}(x_u \wedge x_v)$; the cost is about 1.1 megamems with the algorithms of answers 198–201. With family algebra, on the other hand, we have $f = \wp \searrow \bigvee_{u - v}(e_u \sqcup e_v)$ by exercise 236(e); the cost, via answer 237, is less than 175 kilomems.

generating functions
maximum versus maximal
Berge

The subsets that give 2-colorable and 3-colorable subgraphs are $g = f \sqcup f$ and $h = g \sqcup f$, respectively; the maximal ones are $g^{\uparrow}$ and $h^{\uparrow}$. We have $Z(g) = 1009$, $Z(g^{\uparrow}) = 3040$, $Z(h) = 179$, $Z(h^{\uparrow}) = 183$, $|g| = 9{,}028{,}058{,}789{,}780$, $|g^{\uparrow}| = 2{,}949{,}441$, $|h| = 543{,}871{,}144{,}820{,}736$, and $|h^{\uparrow}| = 384$. The successive costs of computing g, $g^{\uparrow}$, h, and $h^{\uparrow}$ are approximately 350 Kμ (kilomems), 3.6 Mμ, 1.1 Mμ, and 230 Kμ. (We could compute $h^{\uparrow}$ by, say, $(g^{\uparrow} \sqcup f)^{\uparrow}$; but that turns out to be a bad idea.)

The maximal induced bipartite and tripartite subgraphs have the respective generating functions $7654z^{25} + \cdots + 9040z^{33} + 689z^{34}$ and $128z^{43} + 84z^{44} + 112z^{45} + 36z^{46} + 24z^{47}$. Here are typical examples of the smallest and largest:



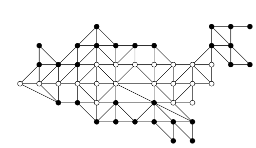
(Compare with the smallest and largest "1-partite" subgraphs in 7–(61) and 7–(62).)

Notice that the families g and h tell us exactly which induced subgraphs can be 2-colored and 3-colored, but they *don't* tell us how to color them.

239. Since $h = ((e_1 \cup \cdots \cup e_{49}) \S\, 2) \setminus g$ is the set of nonedges of G, the cliques are $f = \wp \searrow h$, and the maximal cliques are $f^{\uparrow}$. For example, we have $Z(f) = 144$ for the 214 cliques of the USA graph, and $Z(f^{\uparrow}) = 130$ for the 60 maximal ones. In this case the maximal cliques consist of 57 triangles (which are easily visible in (18)), together with three edges that aren't part of any triangle: `AZ` — `NM`, `WI` — `MI`, `NH` — `ME`.

Let f_k describe the sets coverable by k cliques. Then $f_1 = f$, and $f_{k+1} = f_k \sqcup f$ for $k \ge 1$. (It's not a good idea to compute f_{16} as $f_8 \sqcup f_8$; much faster is to do each join separately, even if the intermediate results are not of interest.)

The maxim*um* elements of f_k in the USA graph have sizes 3, 6, 9, ..., 36, 39, 41, 43, 45, 47, 48, 49 for $1 \le k \le 19$; these maxima can readily be determined by hand, in a small graph such as this. But the question of maxim*al* elements is much more subtle, and ZDDs are probably the best tool for investigating them. The ZDDs for $f_1, \ldots, f_{19}$ are quickly found after about 30 megamems of calculation, and they aren't large: $\max Z(f_k) = Z(f_{11}) = 9547$. Another 400 megamems produces the ZDDs for $f_1^{\uparrow}, \ldots, f_{19}^{\uparrow}$, which likewise are small: $\max Z(f_k^{\uparrow}) = Z(f_{11}^{\uparrow}) = 9458$.



We find, for example, that the generating function for $f_{18}^{\uparrow}$ is $12z^{47} + 13z^{48}$; eighteen cliques suffice to cover all but one of the 49 vertices, if we leave out `CA`, `DC`, `FL`, `IL`, `LA`, `MI`, `MN`, `MT`, `SC`, `TN`, `UT`, `WA`, or `WV`. There also are twelve cases where we can maximally cover 47 vertices; for example, if all but `NE` and `NM` are covered by 18 cliques, then neither of those states are covered. An unusual example of maximal clique covering is illustrated here: If the 29 "black" states are covered by 12 cliques, none of the "white" states will also be covered.

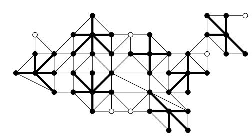
240. (a) In fact, the subformula $f(x) = \bigwedge_v (x_v \vee \bigvee_{u \,—\, v} x_u)$ of (67) precisely characterizes the dominating sets x. And if any element of a kernel is removed, it isn't dominated by the others. [C. Berge, *Théorie des graphes et ses applications* (1958), 44.]

(b) The Boolean formula of part (a) yields a ZDD with $Z(f) = 888$ after about 1.5 Mμ of computation; then another 1.5 Mμ with the MINMAL algorithm of answer 237 gives the minimal elements, with $Z(f^{\downarrow}) = 2082$.

A more clever way is to start with $h = \bigvee_v (e_v \sqcup \bigsqcup_{u \,—\, v} e_u)$, and then to compute $h^{\sharp}$, because $h^{\sharp} = f^{\downarrow}$. However, cleverness doesn't pay in this case: About 80 Kμ suffice to compute h, but the computation of $h^{\sharp}$ by the CROSS algorithm costs about 350 Mμ.

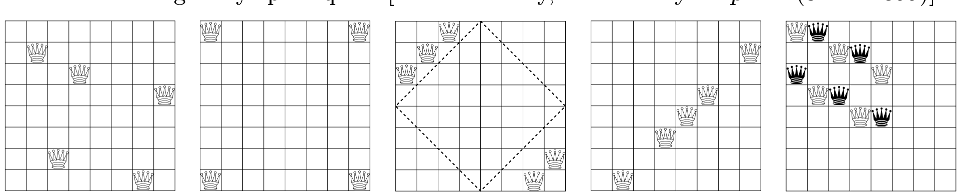
Either way, we deduce that there are exactly 7,798,658 minimal dominating sets. More precisely, the generating function has the form $192z^{11}+58855z^{12}+\cdots+4170z^{18}+40z^{19}$ (which can be compared to $80z^{11}+7851z^{12}+\cdots+441z^{18}+18z^{19}$ for kernels).

(c) Proceeding as in answer 239, we can determine the sets of vertices d_k that are dominated by subsets of size $k = 1, 2, 3, \ldots$, because $d_{k+1} = d_k \sqcup d_1$. Here it's much faster to start with $d_1 = \wp \sqcap h$ instead of $d_1 = h$, even though $Z(\wp \sqcap h) = 313$ while $Z(h) = 213$, because we aren't interested in details about the small-cardinality members of d_k. Using the fact that the generating function for d_7 is $\cdots+61z^{42}+z^{43}$, one can verify that the illustrated solution is unique. (Total cost ≈ 300 Mμ.)

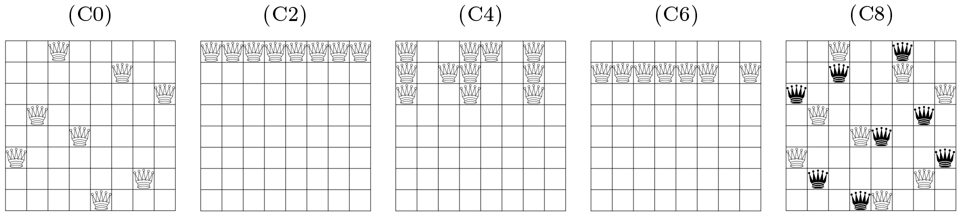


241. Let g the family of all 728 edges. Then, as in previous exercises, $f = \wp \searrow g$ is the family of independent sets, and the cliques are $c = \wp \searrow (((\bigcup_v e_v) \mathbin{\S} 2) \setminus g)$. We have $Z(g) = 699$, $Z(f) = 20244$, $Z(c) = 1882$.

(a) Among $|f| = 118969$ independent sets, there are $|f^{\uparrow}| = 10188$ kernels, with $Z(f^{\uparrow}) = 8577$ and generating function $728z^5+6912z^6+2456z^7+92z^8$. The 92 maximum independent sets are the famous solutions to the classic 8-queens problem, which we shall study in Section 7.2.2; example (C1) is the only solution with no three queens in a straight line, as noted by Sam Loyd in the *Brooklyn Daily Eagle* (20 December 1896). The $728 = 91 \times 8$ minimum kernels were first listed by C. F. de Jaenisch, *Traité des applications de l'analyse math. au jeu des échecs* **3** (1863), 255–259, who ascribed them to "M$^{\rm r}$ de R∗∗∗." The upper left queen in (C0) can be replaced by king, bishop, or pawn, still dominating every open square [H. E. Dudeney, *The Weekly Dispatch* (3 Dec 1899)].



(C0) (C2) (C4) (C6) (C8)



(C1) (C3) (C5) (C7) (C9)

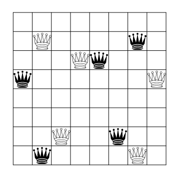
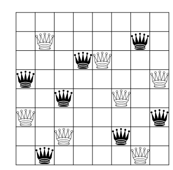
(b) Here $Z(c^{\uparrow}) = 866$; the 310 maximal cliques are described in exercise 7–129.

(c) These subsets are computationally more difficult: The ZDD for all dominating sets d has $Z(d) = 12{,}663{,}505$, $|d| = 18{,}446{,}595{,}708{,}474{,}987{,}957$; the minimal ones have $Z(d^{\downarrow}) = 11{,}363{,}849$, $|d^{\downarrow}| = 28{,}281{,}838$, and generating function $4860z^5+1075580z^6+14338028z^7+11978518z^8+873200z^9+11616z^{10}+36z^{11}$. One can compute the ZDD for d in 1.5 Gμ by Boolean algebra, and then the ZDD for $d^{\downarrow}$ in another 680 Gμ; alternatively, the "clever" approach of answer 240 obtains $d^{\downarrow}$ in 775 Gμ without computing d. The 11-queen arrangement in (C5) is the only such minimal dominating set that is confined to three rows. H. E. Dudeney presented (C4), the only 5-queen solution that avoids the central diamond, in *Tit Bits* (1 Jan 1898), 257. The set of all 4860 minimum solutions was first enumerated by K. von Szily [*Deutsche Schachzeitung* **57** (1902), 199]; his complete list appears in W. Ahrens, *Math. Unterhaltungen und Spiele* **1** (1910), 313–318.

generating function
kernels
8-queens problem
no three queens in a straight line
Loyd
de Jaenisch
Dudeney
Dudeney
von Szily
Ahrens

(d) Here it suffices to compute $(c \cap d)^{\downarrow}$ instead of $c \cap (d^{\downarrow})$, if we don't already know $d^{\downarrow}$, because $c \sqcap \wp = c$. We have $Z(c \cap d^{\downarrow}) = 342$ and $|c \cap d^{\downarrow}| = 92$, with generating function $20z^5 + 56z^6 + 16z^7$. Once again, Dudeney was first to discover all 20 of the 5-queen solutions [*The Weekly Dispatch* (30 July 1899)].

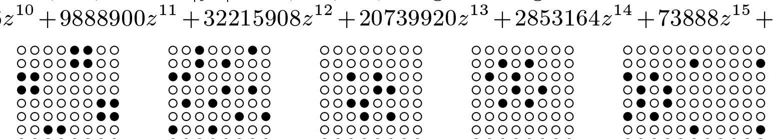
(e) We have $Z(f \sqcup f) =$ 91,780,989 at a cost of 24 Gμ; then $Z((f \sqcup f)^{\uparrow}) =$ 11,808,436 after another 290 Gμ. There are 27,567,390 maximal induced bipartite subgraphs, with generating function $109894z^{10} + 2561492z^{11} + 13833474z^{12} + 9162232z^{13} + 1799264z^{14} + 99408z^{15} + 1626z^{16}$. Any 8 independent queens can be combined with their mirror reflection to obtain a 16-queen solution, as (C1) yields (C9). But the disjoint union of minimum kernels is not always a maximal induced bipartite subgraph; for example, consider the union of (C0) with its reflection:

 $\subseteq$  .

Parts (a), (b), (d), and possibly (c) can be solved just as well without the use of ZDDs; see, for example, exercise 7.1.3–132 for (a) and (b). But the ZDD approach seems best for (e). And the computation of all the maximal *tripartite* subgraphs of Q_8 may be beyond the reach of *any* feasible algorithm.

[In larger queen graphs Q_n, the smallest kernels and the minimum dominating sets are each known to have sizes either $\lceil n/2 \rceil$ or $\lceil n/2 \rceil + 1$ for $12 \le n \le 120$. See P. R. J. Östergård and W. D. Weakley, *Electronic J. Combinatorics* **8** (2001), #R29; D. Finozhenok and W. D. Weakley, *Australasian J. Combinatorics* **37** (2007), 295–200. The largest minimal dominating sets have been investigated by A. P. Burger, E. J. Cockayne, and C. M. Mynhardt, *Discrete Mathematics* **163** (1997), 47–66.]

242. These are the kernels of an interesting 3-regular hypergraph with 1544 edges. Its 4,113,975,079 independent subsets f (that is, its subsets with no three collinear points) have $Z(f) =$ 52,322,105, computable with about 12 gigamems using family algebra as in answer 236(e). Another 575 Gμ will compute the kernels $f^{\uparrow}$, for which we have $Z(f^{\uparrow}) =$ 31,438,750 and $|f^{\uparrow}| =$ 66,509,584; the generating function is $228z^8 + 8240z^9 + 728956z^{10} + 9888900z^{11} + 32215908z^{12} + 20739920z^{13} + 2853164z^{14} + 73888z^{15} + 380z^{16}$.



[The problem of finding an independent set of size 16 was first posed by H. E. Dudeney in *The Weekly Dispatch* (29 Apr 1900 and 13 May 1900), where he gave the leftmost pattern shown above. Later, in the London *Tribune* (7 Nov 1906), Dudeney asked puzzlists to find the second pattern, which has two points in the center. The full set of maximum kernels, including 51 that are distinct under symmetry, was found by M. A. Adena, D. A. Holton, and P. A. Kelly, *Lecture Notes in Math.* **403** (1974), 6–17, who also noted the existence of an 8-point kernel. The middle pattern above is the only such kernel with all points in the central 4×4. The other two patterns yield kernels that have respectively $(8, 8, 10, 10, 12, 12, 12)$ points in $n \times n$ grids for $n = (8, 9, \ldots, 14)$; they were found by S. Ainley and described in a letter to Martin Gardner, 27 Oct 1976.]

243. (a) This result is readily verified even for infinite sets. (Notice that, as a Boolean function, $f^{\cap}$ is the least Horn function that is $\supseteq f$, by Theorem 7.1.1H.)

(b) We could form $f^{(2)} = f \sqcap f$, then $f^{(4)} = f^{(2)} \sqcap f^{(2)}, \ldots,$ until $f^{(2^{k+1})} = f^{(2^k)}$, using exercise 205. But it's faster to devise a recurrence that goes to the limit all at once. If $f = f_0 \cup (e_1 \sqcup f_1)$ we have $f^{\cap} = f' \cup (e_1 \sqcup f_1^{\cap})$, where $f' = f_0^{\cap} \cup (f_0^{\cap} \sqcap f_1^{\cap})$.

Dudeney
5-queen
Östergård
Weakley
Finozhenok
Burger
Cockayne
Mynhardt
kernels
3-regular hypergraph
independent subsets of a hypergraph
family algebra
Dudeney
symmetry
Adena
Holton
Kelly
Ainley
Gardner
infinite sets
Horn function
recurrence

[An alternative formula is $f' = (f_0 \cup f_1)^{\cap} \setminus (f_1^{\cap} \nearrow f_0)$; see S. Minato and H. Arimura, *Transactions of the Japanese Society for Artificial Intelligence* **22** (2007), 165–172.]

(c) With the first suggestion of (b), the computation of $F^{(2)}$, $F^{(4)}$, and $F^{(8)} = F^{(4)}$ costs about (610 + 450 + 460) megamems. In this example it turns out that $F^{(4)} = F^{(3)}$, and that just three patterns belong to $F^{(3)} \setminus F^{(2)}$, namely `c***f`, `*k*t*`, and `***sp`. (The words that match `***sp` are `clasp`, `crisp`, and `grasp`.) A direct computation of $F^{\cap}$ using the recurrence based on $f_0^{\cap} \sqcap f_1^{\cap}$ costs only 320 Mμ; and in this example the alternative recurrence based on $(f_0 \cup f_1)^{\cap}$ costs 470 Mμ. The generating function is $1 + 124z + 2782z^2 + 7753z^3 + 4820z^4 + 5757z^5$.

Minato
Arimura
Z-transforms
Fredman
Khachiyan
Dughmi
Post
bases
matroid

244. To convert Fig. 22 from a BDD to a ZDD, we add appropriate nodes with LO = HI where links jump levels, obtaining the z-profile (1, 2, 2, 4, 5, 5, 5, 5, 5, 2, 2, 2). To convert it from a ZDD to a BDD, we add nodes in the same places, but with HI = $\boxed{\bot}$, obtaining the profile (1, 2, 2, 4, 5, 5, 5, 5, 5, 2, 2, 2). (In fact, the connectedness function and the spanning tree function are Z-transforms of each other; see exercise 192.)

245. See exercise 7.1.1–26. (It should be interesting to compare the performance of the Fredman–Khachiyan algorithm in exercise 7.1.1–27 with the ZDD-based algorithm CROSS in answer 237, on a variety of different functions.)

246. If a nonconstant function doesn't depend on x_1, we can replace x_1 in the formulas by x_v, as in (50). Let P and Q be the prime implicants of functions p and q. (For example, if $P = e_2 \cup (e_3 \sqcup e_4)$ then $p = x_2 \vee (x_3 \wedge x_4)$.) By (137) and induction on $|f|$, the function f described in the theorem is sweet if and only if p and q are sweet and $\mathrm{PI}(f_0) \cap \mathrm{PI}(f_1) = \emptyset$. The latter equality holds if and only if $p \subseteq q$.

247. We can characterize them with BDDs as in (49) and exercise 75; but this time

$$\sigma_n(x_1,\ldots,x_{2^n}) = \sigma_{n-1}(x_1,\ldots,x_{2^n-1}) \wedge \Bigl(\bigl(\bar x_2 \wedge \cdots \wedge \bar x_{2^n}\bigr) \vee \Bigl(\sigma_{n-1}(x_2,\ldots,x_{2^n}) \wedge \bigwedge_{j=0}^{2^{k-1}} \Bigl(\bar x_{2j+1} \vee \bigvee_{i\subset j} x_{2i+2}\Bigr)\Bigr)\Bigr).$$

The answers $|\sigma_n|$ for $0 \le n \le 7$ are (2, 3, 6, 18, 106, 2102, 456774, 7108935325). (This computation builds a BDD of size $B(\sigma_7) = 7{,}701{,}683$, using about 900 megamems and 725 megabytes altogether.)

248. False; for example, $(x_1 \vee x_2) \wedge (x_2 \vee x_3)$ isn't sweet. (But the conjunction *is* sweet if f and g depend on disjoint sets of variables, or if x_1 is the only variable on which they both depend.)

249. (Solution by Shaddin Dughmi and Ian Post.) A nonzero monotone Boolean function is ultrasweet if and only if its prime implicants are the bases of a matroid; see Section 7.6.1. By extending answer 247 we can determine the number of ultrasweet functions $f(x_1,\ldots,x_n)$ for $0 \le n \le 7$: (2, 3, 6, 17, 69, 407, 3808, 75165).

250. Exhaustive analysis shows that ave $B(f) = 76726/7581 \approx 10.1$; ave $Z(\mathrm{PI}(f)) = 71513/7581 \approx 9.4$; $\Pr\bigl(Z(\mathrm{PI}(f)) > B(f)\bigr) = 151/7581 \approx .02$; and max $Z(\mathrm{PI}(f))/B(f) = 8/7$ occurs uniquely when f is $(x_1{\wedge}x_4) \vee (x_1{\wedge}x_5) \vee (x_2{\wedge}x_3{\wedge}x_4) \vee (x_2{\wedge}x_5)$.

251. More strongly, could it be that $\limsup Z(\mathrm{PI}(f))/B(f) = 1$?

252. The ZDD should describe all words on $\{e_1, e_1', \ldots, e_n, e_n'\}$ that have exactly j unprimed letters and $k - j$ primed letters, and no occurrences of both e_i and e_i' in the same word, for some set of pairs (j,k). For example, if $n = 9$ and $f(x) = v_{\nu x}$, where $v = 110111011$, the pairs are $(0,8)$, $(3,6)$, and $(8,8)$. Regardless of the set of pairs, the

z-profile elements will all be $O(n^2)$, hence $Z(\mathrm{PI}(f)) = O(n^3)$. (We order the variables so that x_i and x_i' are adjacent.) And $f(x) = S_{\lfloor n/3\rfloor,\ldots,\lfloor 2n/3\rfloor}(x)$ has $Z(\mathrm{PI}(f)) = \Omega(n^3)$.

253. Let $\mathrm{I}(f)$ be the family of all *implicants* of f; then $\mathrm{PI}(f) = \mathrm{I}(f)^{\downarrow}$. The formula $\mathrm{I}(f) = \mathrm{I}(f_0 \wedge f_1) \cup (e_1' \sqcup \mathrm{I}(f_0)) \cup (e_1 \sqcup \mathrm{I}(f_1))$ is easy to verify. Thus $\mathrm{I}(f)^{\downarrow} = A \cup (e_1' \sqcup (\mathrm{PI}(f_0) \searrow A)) \cup (e_1 \sqcup (\mathrm{PI}(f_1) \searrow A))$, as in exercise 237. But $\mathrm{PI}(f_0) \searrow A = \mathrm{PI}(f_0) \setminus A$, since $A \subseteq \mathrm{I}(f)$.

[This recurrence for prime implicants is due to O. Coudert and J. C. Madre, *ACM/IEEE Design Automation Conf.* **29** (1992), 36–39. Partial results had previously been formulated by B. Reusch, *IEEE Trans.* **C–24** (1975), 924–930.]

254. By (53) and (137), we need to show that $\mathrm{PI}(g_h) \setminus \mathrm{PI}(f_h \cup g_l) = \bigl(\mathrm{PI}(g_h)\setminus\mathrm{PI}(g_l)\bigr) \setminus \bigl(\mathrm{PI}(f_h)\setminus\mathrm{PI}(f_l)\bigr)$. But both of these are equal to $\mathrm{PI}(g_h) \setminus (\mathrm{PI}(f_h) \cup \mathrm{PI}(g_l))$, because $f_l \subseteq f_h \subseteq g_h$ and $f_l \subseteq g_l \subseteq g_h$.

[This recurrence produces a ZDD directly from the BDDs for f and g, and it yields $\mathrm{PI}(g)$ when $f = 0$. Thus it is easier to implement than (137), which requires also the set-difference operator on ZDDs. And it sometimes runs much faster in practice.]

255. (a) A typical item α like $e_2 \sqcup e_5 \sqcup e_6$ has a very simple ZDD. We can readily devise a BUMP routine that sets $g \leftarrow g \oplus \alpha$ and returns $[\alpha \in g]$, given ZDDs g and α.

To insert α into the multifamily f, start with $k \leftarrow c \leftarrow 0$; then while $c = 0$, set $c \leftarrow \mathrm{BUMP}(f_k)$ and $k \leftarrow k+1$. To delete α, assuming that it is present, start with $k \leftarrow 0$ and $c \leftarrow 1$; while $c = 1$, set $c \leftarrow \mathrm{BUMP}(f_k)$ and $k \leftarrow k+1$.

(b) Suppose f_k and g_k are $\emptyset$ for $k \geq m$. Set $k \leftarrow 0$ and $t \leftarrow \emptyset$ (the ZDD $\boxed{\bot}$). While $k < m$, set $h_k \leftarrow f_k \oplus g_k \oplus t$ and $t \leftarrow \langle f_k g_k t\rangle$. Finally set $h_m \leftarrow t$.

[This representation and its insertion algorithm are due to S. Minato and H. Arimura, *Proc. Workshop, Web Information Retrieval and Integration* (IEEE, 2005), 3–10.]

256. (a) Reflect the binary representation from left to right, and append 0s until the number of bits is 2^n for some n. The result is the truth table of the corresponding Boolean function $f(x_1, \ldots, x_n)$, with x_k corresponding to $2^{2^{n-k}} \in U$. When $x = 41$, for example, 10010100 is the truth table of $(x_1 \wedge \bar{x}_2 \wedge x_3) \vee (\bar{x}_1 \wedge x_2 \wedge x_3) \vee (\bar{x}_1 \wedge \bar{x}_2 \wedge \bar{x}_3)$.

(b) If $x < 2^{2^n}$, we have $Z(x) \leq U_n = O(2^n/n)$, by (79) and exercise 192.

(c) There's a simple recursive routine $\mathrm{ADD}(x, y, c)$, which takes a "carry bit" c and pointers to the ZDDs for x and y and returns a pointer to the ZDD for $x + y + c$. This routine is invoked at most $4Z(x)Z(y)$ times.

(d) We cannot claim that $Z(x \mathbin{\dot-} y) = O(Z(x)Z(y))$, because $Z(x \mathbin{\dot-} y) = n + 1$ and $Z(x) = 3$ and $Z(y) = 1$ when $x = 2^{2^n}$ and $y = 1$. But by computing $x \mathbin{\dot-} y = (x + 1 + ((2^{2^n} - 1) \oplus y)) - 2^{2^n}$ when $y \leq x < 2^{2^n}$, we can show that $Z(x \mathbin{\dot-} y) = O(Z(x)Z(y) \log\log x)$. (See the ZDD nodes t_j in answer 201.) So the answer is "yes."

(e) No. For example, if $x = (2^{2^{2^k+k}} - 1)/(2^{2^k} - 1)$, we have $Z(x) = 2^k + 1$ but $Z(x^2) = 3 \cdot (2^{2^k} - 1) = U_{2^k+k+1} - 2$, where U_{2^k+k+1} is the largest possible ZDD size for numbers with $\lg\lg x^2 < 2^k + k + 1$ (see part (b)).

[This exercise was inspired by Jean Vuillemin, who began to experiment with such sparse integers about 1993. Unfortunately the numbers that are of greatest importance in combinatorial calculations, such as Fibonacci numbers, factorials, binomial coefficients, etc., rarely turn out to be sparse in practice.]

257. See *Proc. Europ. Design and Test Conf.* (IEEE, 1995), 449–454. With signed coefficients one can use $\{-2, 4, -8, \ldots\}$ instead of $\{2, 4, 8, \ldots\}$, as in negabinary arithmetic.

[In the special case where the degree is at most 1 in each variable and where addition is done modulo 2, the polynomials of this exercise are equivalent to the

implicants
Coudert
Madre
Reusch
Minato
Arimura
Reflect
carry bit
partial tautology
Vuillemin
Fibonacci numbers
negabinary arithmetic

multilinear representations of Boolean functions (see 7.1.1–(19)), and the ZDDs are equivalent to "binary moment diagrams" (BMDs). See R. E. Bryant and Y.-A. Chen, *ACM/IEEE Design Automation Conf.* **32** (1995), 535–541.]

258. If n is odd, the BDD must depend on all its variables, and there must be at least $\lceil \lg n \rceil$ of them. Thus $B(f) \geq \lceil \lg n \rceil + 2$ when $n > 1$, and the skinny functions of exercise 170(c) achieve this bound. If n is even, add an unused variable to the solution for $n/2$.

The ZDD question is easily seen to be equivalent to finding a shortest addition chain, as in Section 4.6.3. Thus the smallest $Z(f)$ for $|f| = n$ is $l(n)+1$, including $\boxed{\top}$.

259. The theory of nested parentheses (see, for example, exercise 2.2.1–3) tells us that $N_n(x) = 1$ if and only if $\bar{x}_1 + \cdots + \bar{x}_k \geq x_1 + \cdots + x_k$ for $0 \leq k \leq 2n$, with equality when $k = 2n$. Equivalently, $k - n \leq x_1 + \cdots + x_k \leq k/2$ for $0 \leq k \leq 2n$. So the BDD for N_n is rather like the BDD for $S_n(x)$, but simpler; in fact, the profile elements are $b_k = \lfloor k/2 \rfloor + 1$ for $0 \leq k \leq n$ and $b_k = n + 1 - \lceil k/2 \rceil$ for $n \leq k < 2n$. Hence $B(N_n) = b_0 + \cdots + b_{2n-1} + 2 = \binom{n+2}{2} + 1$. The z-profile has $z_k = b_k - [k \text{ even}]$ for $0 \leq k < 2n$, because of HI branches to $\boxed{\bot}$ on even levels; hence $Z(N_n) = B(N_n) - n$.

[An interesting BDD base for the $n+1$ Boolean functions that correspond to C_{nn}, $C_{(n-1)(n+1)}, \ldots, C_{0(2n)}$ in 7.2.1.6–(21) can be constructed by analogy with exercise 49.]

260. (a, b) Arrange the variables $x_{n,0}$, $x_{n,1}$, $\ldots$, $x_{n,n-1}$, $x_{n-1,0}$, $\ldots$, $x_{1,0}$, from top to bottom. Then the HI branch from the ZDD root of R_n is the ZDD root of R_{n-1}. (This ordering actually turns out to minimize $Z(R_n)$ for $n \leq 6$, probably also for all n.) The z-profile is 1, $\ldots$, 1; $n-2$, $\ldots$, 2, 1, 1; $n-3$, $\ldots$, 2, 1, 1; $\ldots$; hence $Z(R_n) = \binom{n}{3} + 2n + 1 \approx \frac{1}{6}n^3$ and $Z(R_{100}) = 161{,}901$. The ordinary profile is 1, 2, 2, 3, 4, $\ldots$, $n-1$; $n-1$, $2n-4$, $2n-5$, $\ldots$, $n-1$; $n-2$, $2n-6$, $\ldots$, $n-2$; $\ldots$; altogether $B(R_n) = 3\binom{n}{3} + \binom{n+1}{2} + 3$ for $n \geq 5$, and $B(R_{100}) = 490{,}153$.

[See I. Semba and S. Yajima, *Trans. Inf. Proc. Soc. Japan* **35** (1994), 1666–1667. Incidentally, the method of exercise 7.2.1.5–26 leads to a ZDD for set partitions that has only $\binom{n}{2}$ variables and $\binom{n}{2} + 1$ nodes. But the connection between that representation and the partitions themselves is less direct, thus harder to restrict in a natural way.]

(c) Now there are 573 variables instead of 5050 when $n = 10$; the number of variables in general is $nl - 2^l + 1$, where $l = \lceil \lg n \rceil$, by Eq. 5.3.1–(3). We examine the bits of a_n, a_{n-1}, $\ldots$, with the most significant bit first. Then $B(R'_{100}) = 31{,}861$, and one can show that $B(R'_n) = \binom{n}{2}l - \frac{1}{6}4^l - \frac{1}{2}2^l - \nu(n-1) + l + \frac{8}{3}$ for $n > 2$. The ZDD size is more complicated, and appears to be roughly 60% larger; we have $Z(R'_{100}) = 50{,}154$.

261. Given a Boolean function $f(x_1, \ldots, x_n)$, the set of all binary strings $x_1 \ldots x_n$ such that $f(x_1, \ldots, x_n) = 1$ is a finite language, so it is regular. The minimum-state deterministic automaton $\mathcal{A}$ for this language is the QDD for f. (In general, when L is regular, the state of $\mathcal{A}$ after reading $x_1 \ldots x_k$ accepts the language $\{\alpha \mid x_1 \ldots x_k \alpha \in L\}$.)

[The quoted theorem was discovered in a more general context by D. A. Huffman, *Journal of the Franklin Institute* **257** (1954), 161–190, and independently by E. F. Moore, *Annals of Mathematics Studies* **34** (1956), 129–153.]

An interesting example of the connection between this theory and the theory of BDDs can be found in early work by Yuri Breitbart that is summarized in *Doklady Akad. Nauk SSSR* **180** (1968), 1053–1055. Lemma 7 of Breitbart's paper states, in essence, that $B_{\min}(\psi) = \Omega(2^{n/4})$, where ψ is the function of $2n$ variables $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ defined by $\psi(x, y) = x_{\nu y} \oplus y_{\nu x}$, with the understanding that $x_0 = y_0 = 0$. (Notice that ψ is sort of a "two-sided" hidden weighted bit function.)

262. (a) If a denotes the function or subfunction f, we can for example let $C(a) = a \oplus 1$ denote $\bar{f}$, assuming that each node occupies an even number of bytes. Then

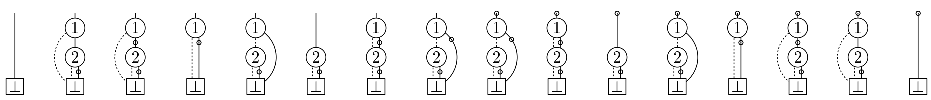
multilinear representations
binary moment diagrams
BMDs
Bryant
Chen
skinny
addition chain
ballot numbers C_{mn}
Semba
Yajima
QDD
Huffman
Moore
Breitbart
hidden weighted bit function

$C(C(a)) = a$, and a link to a denotes a nonnormal function if and only if a is odd; $a \mathbin{\&} -2$ always points to a node, which always represents a normal function.

The LO pointer of every node is even, because a normal function remains normal when we replace any variable by 0. But the HI pointer of any node might be complemented, and an external root pointer to any function of a normalized BDD base might also be complemented. Notice that the $\boxed{\top}$ sink is now impossible.

(b) Uniqueness is obvious because of the relation to truth tables: A bead is either normal (i.e., begins with 0) or the complement of a normal bead.

(c) In diagrams, each complement link is conveniently indicated by a dot:



1 3 3 2 3 2 3 3 3 3 2 3 2 3 3 1

(d) There are $2^{2^m-1} - 2^{2^{m-1}-1}$ normal beads of order m. The worst case, $B^0(f) \le B^0(f_n) = 1 + \sum_{k=0}^{n-1} \min(2^k, 2^{2^{n-k}-1} - 2^{2^{n-k-1}-1}) = (U_{n+1} - 1)/2$, occurs with the functions of answer 110. For the average normalized profile, change $2^{2^{n-k}} - 1$ in (80) to $2^{2^{n-k}} - 2$, and divide the whole formula by 2; again the average case is very close to the worst case. For example, instead of (81) we have

$$(1.0, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0, 127.3, 103.9, 6.0, 1.0, 1.0).$$

(e) We save $\boxed{\top}$, one (6), two (5)s, and three (4)s, leaving 45 normalized nodes.

(f) It's probably best to have subroutines AND, OR, BUTNOT for the case where f and g are known to be normal, together with a subroutine GAND for the general case. The routine GAND(f,g) returns AND(f,g) if f and g are even, BUTNOT$(f, C(g))$ if f is even but g is odd, BUTNOT$(g, C(f))$ if g is even but f is odd, $C(\text{OR}(C(f), C(g)))$ if f and g are odd. The routine AND(f,g) is like (55) except that $r_h \leftarrow \text{GAND}(f_h, g_h)$; only the cases $f = 0$, $g = 0$, and $f = g$ need be tested as "obvious" values.

Notes: Complement links were proposed by S. Akers in 1978, and independently by J. P. Billon in 1987. Although such links are used by all the major BDD packages, they are hard to recommend because the computer programs become much more complicated. The memory saving is usually negligible, and never better than a factor of 2; furthermore, the author's experiments show little gain in running time.

With ZDDs instead of BDDs, a "normal family" of functions is a family that doesn't contain the empty set. Shin-ichi Minato has suggested using $C(a)$ to denote the family $f \oplus \epsilon$, instead of $\bar{f}$, in ZDD work.

263. (a) If $Hx = 0$ and $x \ne 0$, we can't have $\nu x = 1$ or 2 because the columns of H are nonzero and distinct. [R. W. Hamming, *Bell System Tech. J.* **29** (1950), 147–160.]

(b) Let r_k be the rank of the first k columns of H, and s_k the rank of the last k columns. Then $b_k = 2^{r_k + s_{n-k} - r_n}$ for $0 \le k < n$, because this is the number of elements in the intersection of the vector spaces spanned by the first k and last $n - k$ columns. In the Hamming code, $r_k = 1 + \lambda k$ and $s_k = \min(m, 2 + \lambda(k-1))$ for $k > 1$; so we find $B(f) = (n^2 + 5)/2$. [See G. D. Forney, Jr., *IEEE Trans.* **IT-34** (1988), 1184–1187.]

(c) Let $q_k = 1 - p_k$. Maximizing $\prod_{k=1}^{n} p_k^{[x_k = y_k]} q_k^{[x_k \ne y_k]}$ is the same as maximizing $\sum_{k=1}^{n} w_k x_k$, where $w_k = (2y_k - 1)\log(p_k/q_k)$, so we can use Algorithm B.

Notes: Coding theorists, beginning with unpublished work of Forney in 1967, have developed the idea of a code's so-called *trellis*. In the binary case, the trellis is the same as the QDD for f, but with all nodes for the constant subfunction 0 eliminated. (Useful codes have distance > 1; then the trellis is also the BDD for f, but with $\boxed{\bot}$

root pointer
truth tables
bead
Akers
Billon
Knuth
ZDD
normal family
Minato
Hamming
vector spaces
Forney
Coding theorists
trellis
QDD

eliminated.) Forney's original motivation was to show that the decoding algorithm of A. Viterbi [*IEEE Trans.* **IT-13** (1967), 260–269] is optimum for convolutional codes. A few years later, L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv [*IEEE Trans.* **IT-20** (1974), 284–287] extended trellis structure to linear block codes and presented further optimization algorithms. See also the papers of G. B. Horn and F. R. Kschischang [*IEEE Trans.* **IT-42** (1996), 2042–2047]; J. Lafferty and A. Vardy [*IEEE Trans.* **C-48** (1999), 971–986].

264. Procedures that combine the "bottom-up" methods of Algorithm B with "top-down" methods that optimize over predecessors of a node might be more efficient than methods that go strictly in one direction.

INDEX AND GLOSSARY

WHEATLEY

Indexes need not necessarily be dry.

— HENRY B. WHEATLEY, *How to Make an Index* (1902)

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.