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

KNUTH

THE ART OF COMPUTER PROGRAMMING

VOLUME 4 PRE-FASCICLE 7A

A DRAFT OF SECTION 7.2.2.3: CONSTRAINT SATISFACTION

DONALD E. KNUTH *Stanford University*

ADDISON–WESLEY

December 5, 2024

Internet
Stanford GraphBase
downloadable software
MMIX

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

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

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

See also https://www-cs-faculty.stanford.edu/~knuth/programs.html for various experimental programs that I wrote while writing this material (and some data files).

Zeroth printing (revision 4), 04 December 2024

TWAIN
Internet

PREFACE

There is quite a good deal of information in the book.
I regret this very much; but really it could not be helped:
information appears to stew out of me naturally,
like the precious ottar of roses out of the otter.
Sometimes it has seemed to me that I would give worlds
if I could retain my facts; but it cannot be.
The more I calk up the sources, and the tighter I get, the more I leak wisdom.
Therefore, I can only claim indulgence at the hands of the reader, not justification.
— MARK TWAIN, *Roughing It* (1872)

FREEDOM is a wonderful thing. But we can also be thankful for the constraints that give structure to our lives and provide a focus for our creative juices. We experience moments of great satisfaction when challenges have been met. Combinatorial and musical patterns that fit together almost magically can be supremely satisfying.

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

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

To put the material in context, this portion of fascicle 7 previews Section 7.2.2.3 of *The Art of Computer Programming*, entitled "Constraint satisfaction." It will be the first section of Volume 4C. As usual, it covers many topics that are of independent interest and that have close ties to other sections.

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

Please look, for example, at the exercises that I've classed as research problems (rated with difficulty level 46 or higher), namely exercises 37, 48, 66, 81, 84, 96, 106, 119, 138, 147, 231, 474, 476; I've also implicitly mentioned or posed additional unsolved questions in the answers to exercises 36, 63, 90, 93, 94, 98(d), 104(e), 119, 134, 145, 155, 211, 227, 233, 277(b), 281, 296, 306(e, f), 351, 357, 390, 462, 475, 484. Are those intriguing problems still open? Please inform me if you know of a solution to any of them. And of course if no solution is known today but you do make progress on any of them in the future, I hope you'll let me know.

I urgently need your help also with respect to some exercises that I made up as I was preparing this material. I certainly don't like to receive credit for things that have already been published by others, and most of these results are quite natural "fruits" that were just waiting to be "plucked." Therefore please tell me if you know who deserves to be credited, with respect to the ideas found in exercises 27, 28, 29, 30, 31, 32, 34, 35, 36, 37, 39, 46, 47, 49, 50, 51, 52, 53, 54, 55, 56, 57, 59, 60, 61, 62, 68, 75, 88, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 107, 108, 110, 115, 116, 117, 118, 123, 126, 127, 128, 129, 132, 137(c), 138, 143, 146, 155, 158, 164, 165, 166, 169, 173, 187, 189, 190, 200, 205, 212, 216, 217, 222, 223, 226, 241, 248, 255, 256, 257, 258, 265, 277, 280, 284, 285, 286, 287, 288, 290, 291, 306, 328, 351, 359, 371, 379, 383, 385, 387, 388, 390, 392, 399, 400, 401, 402, 403, 404, 407, 408, 410, 411, 412, 413, 421, 422, 423, 438, 439, 441, 444, 446, 447, 452, 453, 454(a), 456, 457, 464, 469, 475, 480, 481, 482, 483, 484, and their answers. Furthermore I've credited exercises 25 and 117 to unpublished work of Ira Gessel and Nikolai Beluhov. Have either of those results ever appeared in print, to your knowledge?

Can anybody help me identify the source of the crystal maze puzzle? (The answer to exercise 13 tells what I know so far.)

* * *

Special thanks are due to Christian Bessière, Víctor Dalmau, Ralph Freese, Daniel Horsley, Peter Jeavons, Ciaran McCreesh, Patrick Prosser, George Sicherman, Christine Solnon, Filip Stappers, Peter Stuckey, Kokichi Sugihara, James Trimble, Udo Wermuth, Ross Willard, and Dmitriy Zhuk for their detailed comments on my early attempts at exposition, as well as to numerous other correspondents who have contributed crucial corrections. And above all, I thank my wife for her constant support and for help with Figs. 100 and 101.

* * *

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:–)

The answers to several of the exercises refer to programs that I wrote while preparing this material. If you want to see a program called FOO, look for FOO on

Gessel
Beluhov
Bessière
Dalmau
Freese
Horsley
Jeavons
McCreesh
Prosser
Sicherman
Solnon
Stappers
Stuckey
Sugihara
Trimble
Wermuth
Willard
Zhuk
Knuth, Jill

the webpage `https://cs.stanford.edu/~knuth/programs.html`. (Many other example programs can also be found there.)

As in Volume 4B, I've posted prototypes of the algorithms presented here on that same webpage. In particular, you can download the programs SSXCC0, SSXCC, SSXCC-BINARY, SSMCC, and XCCDC; those experimental versions of Algorithms C, C^+, B, F, and S were my constant companions while writing the middle portions of Section 7.2.2.3.

Data files for the benchmark examples mentioned in that section can also be found online at

`https://cs.stanford.edu/~knuth/programs/xcc-benchmarks.tgz`
`https://cs.stanford.edu/~knuth/programs/mcc-benchmarks.tgz`

so that interested readers can do their own experiments.

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.
99 Umbruary 2019

P.S.: **A note on notations.** Some formulas in this booklet use the notation 'νx' for the "sideways sum" or "population count" function, as well as the notation 'ρx' for the "ruler" function. Those functions, and other bitwise notations, are discussed extensively in Section 7.1.3 of Volume 4A.

Other formulas use the notation $\langle xyz\rangle$ for the median function, which is discussed extensively in Section 7.1.1.

Hexadecimal constants are preceded by a number sign or hash mark: #123 means $(123)_{16}$.

If you run across other notations that appear strange, please look at the Index to Notations (Appendix B) at the end of Volume 4A or 4B. Volume 4C will, of course, have its own Appendix B some day.

internet
downloadable programs and data–online
Knuth
notation 'νx
sideways sum
population count
notation 'ρx
notation $\langle xyz\rangle$
median function
Hexadecimal constants
TARJAN

The field of combinatorial algorithms is too vast
to cover in a single paper or even in a single book.
— ROBERT ENDRE TARJAN, *SIAM Review* (1978)

A foretaste of Section 7.4.1.2. Section 7.2.2.3 refers forward to Tarjan's algorithm for strong components, which will be discussed at the beginning of Section 7.4.1.2 ("Depth-first search"), according to present plans. That algorithm is copied here for reference. (More details and exposition can be found in prefascicle 12a, on the Internet at `https://cs.stanford.edu/~knuth/fasc12a.ps.gz`.)

In any directed graph g, the relation "u is reachable from v and v is reachable from u" is clearly an equivalence relation. The equivalence classes of this relation are called the *strong components* of g, and a digraph with only one strong component is called *strongly connected*.

Robert Tarjan discovered that the strong components of any given digraph can actually be discovered "on the fly" while we're doing a depth-first search, by doing only a small amount of additional work.

A depth-first search mimics the behavior of a well-organized spelunker who is exploring a new cave. We start at an arbitrary vertex, v. Whenever we're at v, we have access to each of v's outgoing arcs, in some arbitrary order, and we look at them one by one. Two cases arise when we're examining an arc $v \longrightarrow u$:

i) u hasn't been seen before. In this case we say that u is a "child" of v, and that v is u's "parent"; we move from v to u (that is, u becomes the new v).

ii) u has already been seen. We stay at v.

Eventually all arcs from v will have been explored, and we return from v to v's parent. But if v has no parent, we move to any yet-unexplored vertex v — unless we've already seen them all, in which case we're done.

The arcs from parent to child that arise in case (i), together with the vertices of the given digraph g, form an oriented forest, called a *depth-first forest* of g.

In general, there's a conceptually simple way to find the strong components of any digraph: The vertices of an oriented cycle are strongly connected; so they're equivalent, and we can shrink them to a point, a "supervertex." Let's do that repeatedly until the digraph no longer has any oriented cycles. The remaining supervertices are the (shrunken) strong components.

Notice now that every nonempty directed acyclic graph has at least one "sink" vertex, a vertex with out-degree 0. Therefore, by the shrinking mechanism just described, *every nonempty digraph has at least one "sink" strong component*, a strong component without arcs to any other strong components.

Let S be a sink strong component of the digraph g. Since there's no escape from S, we can remove any arcs of g that lead to a vertex of S; after all, arcs into a "black hole" have no influence on g's strong components. Consequently *the strong components of g are S together with the strong components of $g \setminus S$.*

Good — there's a decent chance that we might discover an efficient method, based on the notion of repeatedly identifying and removing sink components until all components have been found. Depth-first search has the convenient property that it learns about g's vertices and arcs one by one; we essentially want to extend it so that it can readily recognize (and remove) sink components.

Recall that all vertices are initially unseen when depth-first search begins. Then they become active, one by one, in preorder of the depth-first forest.

Depth-first search
Internet
strong components–
Tarjan's algorithm–
reachable
equivalence relation
strongly connected
Tarjan
child
oriented forest
depth-first forest
oriented cycle
shrink
supervertex
directed acyclic graph
sink
unseen
active

Furthermore, a vertex remains active until the moment when we know that we've seen all of its outgoing arcs. In other words, every vertex goes through three stages as we perform depth-first search: It's unseen and inactive, then seen and active, then seen and inactive. We're going to extend that process by also removing vertices from the graph and calling them "settled," as soon as we know that they're part of a sink component with respect to unsettled vertices.

seen
settled
sink component
SENT
sentinel
leader
isolated vertex

Thus every vertex will go through *four* stages as we search for strong components: First it's unseen, inactive, unsettled; then it's seen, active, unsettled; then it's seen, inactive, unsettled; finally it's seen, inactive, settled. The standard depth-first search algorithm tells us how to handle the first transitions. The only thing missing is an appropriate way to settle an unsettled vertex.

Let U be the set of all currently seen but unsettled vertices. We can list them

$$u_1,\ u_2,\ \ldots,\ u_t, \quad \text{where} \quad \mathtt{PRE}(u_1) < \mathtt{PRE}(u_2) < \cdots < \mathtt{PRE}(u_t), \qquad (*)$$

according to the order in which we've seen them. Some of those vertices might be active, while others might be inactive. If the active ones are $u_{j_1}, u_{j_2}, \ldots, u_{j_a}$, with $j_1 < j_2 < \cdots < j_a$, the depth-first strategy tells us that $\mathtt{PARENT}(u_{j_1}) = \mathtt{SENT}$, $\mathtt{PARENT}(u_{j_2}) = u_{j_1}$, ..., $\mathtt{PARENT}(u_{j_a}) = u_{j_{a-1}}$, where SENT is a special "sentinel" value that indicates a nonexistent parent.

We've already seen every arc that goes out from the inactive vertices of U. For all we know, however, there may still be lots and lots of yet-unseen arcs, leading from any or all of the active vertices to any vertices whatsoever.

A nice structure does turn out to be present:

Lemma S. *Let $\hat{g}$ be the digraph whose vertices U are listed in $(*)$, and whose arcs are those seen so far from U into U during a depth-first search for the strong components of g. Assume that $t \geq 1$. Also let $S_1, \ldots, S_k$ be the strong components of $\hat{g}$. Then the leftmost element of each S_j, called its "leader" and denoted by u'_j, is an active vertex; and if u''_j denotes S_j's rightmost active vertex, $\hat{g}$ contains the $k-1$ tree arcs*

$$u''_1 \longrightarrow u'_2,\ \ u''_2 \longrightarrow u'_3,\ \ \ldots,\ \ u''_{k-1} \longrightarrow u'_k,$$

which link those components into a path. All other arcs of $\hat{g}$ lie within the individual strong components S_j, which form consecutive intervals of the list $(*)$.

Proof. The lemma holds initially, when $\hat{g}$ is a single isolated active vertex, because an isolated vertex u_1 is a strong component S_1 all by itself. The proof in Section 7.4.1.2 shows that the lemma remains true as the computation proceeds. ▮

Corollary S. *Suppose digraph g has s strong components $\{C_1, \ldots, C_s\}$, and let v_j be the vertex of C_j that is earliest in preorder of g's depth-first forest. Order the components so that $\mathtt{POST}(v_1) < \cdots < \mathtt{POST}(v_s)$. Then the vertices of C_j are*

$$\operatorname{desc}(v_j) \setminus \bigl(\operatorname{desc}(v_1) \cup \cdots \cup \operatorname{desc}(v_{j-1})\bigr), \qquad \text{for } 1 \leq j \leq s,$$

where $\operatorname{desc}(v)$ denotes the descendants of v in the forest. ▮

(Hence the strong components of g are the *connected* components of the digraph obtained by deleting the arcs to $v_1, \ldots, v_s$ from g's depth-first spanning forest.)

The structure guaranteed by Lemma S almost gives us the efficient algorithm that we want. But one ingredient is still missing, namely a quick way to deal with the boundaries between the current candidates S_j for strong components. Tarjan discovered that an extra field called LOW in each vertex is able to do what we need.

The definition of LOW is somewhat tricky and technical: When v is seen but unsettled, $\mathtt{LOW}(v)$ is the smallest preorder rank of an unsettled vertex w for which we've seen a "downpath" from v to w, where a downpath is an oriented path

$$v = v_0 \longrightarrow v_1 \longrightarrow \cdots \longrightarrow v_r = w, \qquad \text{for some } r \geq 0,$$

such that $v_j \longrightarrow v_{j+1}$ "matured" before $v_{j-1} \longrightarrow v_j$ for $0 < j < r$. A tree arc $v \longrightarrow u$ matures when u becomes inactive; a nontree arc matures when we first see it.

Tarjan
LOW
downpath
tree arc
matures
nontree arc
leader
Tarjan
leader
SENT

Theorem T. *When vertex v becomes inactive during a depth-first search, v is the leader of the current sink component if and only if* $\mathtt{LOW}(v) = \mathtt{PRE}(v)$.

Proof. (This fact is Tarjan's "secret sauce.") See exercise 7.4.1.2–50. ▮

OK, we're ready now for the nitty-gritty details of an efficient algorithm. Algorithm T maintains a stack whose top is SINK, linked together in LINK fields, which holds all the inactive vertices that aren't yet settled. It also gives every vertex a REP field for the equivalence class representatives: Vertices v and w are strongly connected if and only if $\mathtt{REP}(v) = \mathtt{REP}(w)$ at termination.

Tarjan noticed in 2021 that we can save time and space by encoding PRE and LOW in the existing LINK and REP fields, because we need only know whether or not $\mathtt{LOW} = \mathtt{PRE}$. While vertex v is active, we shall therefore let

$$\mathtt{LINK}(v) = \begin{cases} \mathtt{SENT}, & \text{if } \mathtt{LOW}(v) = \mathtt{PRE}(v); \\ \Lambda, & \text{if } \mathtt{LOW}(v) \neq \mathtt{PRE}(v). \end{cases} \qquad (**)$$

Moreover, after v has been seen, we shall let

$$\mathtt{REP}(v) = \begin{cases} \mathtt{LOW}(v), & \text{if } v \text{ isn't settled;} \\ \mathtt{SENT} + v', & \text{if } v \text{ is settled with strong component leader } v'. \end{cases} \qquad (***)$$

Algorithm T (*Strong components*). Given a digraph g in SGB format, this algorithm determines the strong components of g. Each vertex is assumed to contain utility fields PARENT, ARC, LINK, and REP, as explained above. Auxiliary vertex pointer variables u, v, w, ROOT, and SINK are used; also an arc pointer a and an integer p. A dummy vertex $\mathtt{SENT} = \mathtt{VERTICES}(g) + \mathtt{N}(g)$ is assumed to be present, with $\mathtt{REP(SENT)} = 0$.

T1. [Initialize.] Set $\mathtt{PARENT}(w) \leftarrow \Lambda$ for $\mathtt{VERTICES}(g) \leq w < \mathtt{SENT}$, until eventually $w = \mathtt{SENT}$. Also set $p \leftarrow 0$ and $\mathtt{SINK} \leftarrow \mathtt{SENT}$.

T2. [Done?] (All vertices reachable from a vertex $v \geq w$ are now settled.) Terminate if $w = \mathtt{VERTICES}(g)$. Otherwise set $w \leftarrow w - 1$, and repeat this step if $\mathtt{PARENT}(w) \neq \Lambda$. Otherwise set $v \leftarrow w$, $\mathtt{PARENT}(v) \leftarrow \mathtt{SENT}$, and $\mathtt{ROOT} \leftarrow v$.

T3. [Begin to explore from v.] Set $a \leftarrow \mathtt{ARCS}(v)$, $p \leftarrow p + 1$, $\mathtt{REP}(v) \leftarrow p$, and $\mathtt{LINK}(v) \leftarrow \mathtt{SENT}$.

T4. [Done with v?] If $a = \Lambda$, go to T7.

T5. [Visit the next arc, $v \longrightarrow u$.] Set $u \leftarrow$ `TIP`(a) and $a \leftarrow$ `NEXT`(a).

T6. [If u is new, move to it.] If `PARENT`$(u) = \Lambda$, set `PARENT`$(u) \leftarrow v$, `ARC`$(v) \leftarrow a$, $v \leftarrow u$, and go to T3. Otherwise, if $u =$ `ROOT` and $p =$ `N`(g) (we're in the last component), while $v \neq$ `ROOT` set `LINK`$(v) \leftarrow$ `SINK`, `SINK` $\leftarrow v$, $v \leftarrow$ `PARENT`(v); then set $u \leftarrow$ `SENT` and go to T8. Otherwise, if `REP`$(u) <$ `REP`(v), set `REP`$(v) \leftarrow$ `REP`(u), `LINK`$(v) \leftarrow \Lambda$ (see $(**)$, $(***)$). Go to T4.

T7. [Finish with v.] Set $u \leftarrow$ `PARENT`(v). If `LINK`$(v) =$ `SENT`, go to T8. (See Theorem T.) Otherwise, if `REP`$(v) <$ `REP`(u), set `REP`$(u) \leftarrow$ `REP`(v) and `LINK`$(u) \leftarrow \Lambda$. Then set `LINK`$(v) \leftarrow$ `SINK`, `SINK` $\leftarrow v$, and go to T9.

T8. [New strong component.] (Now v and its unsettled descendants form a strong component that will be represented by v.) While `REP(SINK)` $\geq$ `REP`(v), set `REP(SINK)` $\leftarrow$ `SENT` $+ v$ and `SINK` $\leftarrow$ `LINK(SINK)`. (See $(***)$.) Finally set `REP`$(v) \leftarrow$ `SENT` $+ v$.

T9. [Tree done?] If $u =$ `SENT`, go back to T2. Otherwise set $v \leftarrow u$, $a \leftarrow$ `ARC`(v), and go to T4. ▌

A foretaste of Section 7.5.1. Section 7.2.2.3 refers forward to the Hopcroft–Karp algorithm, which will be discussed at the beginning of Section 7.5.1 ("Bipartite matching"), according to present plans. That algorithm is copied here for reference. (Further details and exposition can be found in prefascicle 14a, on the Internet at `https://cs.stanford.edu/~knuth/fasc14a.ps.gz`.)

We're given a bipartite graph. The vertices of one part are called "girls" and the vertices of the other part are called "boys," so that we can conveniently use the English language to distinguish the parts. The problem is to find a *maximum matching*, namely a set of disjoint edges that is as large as possible.

Hopcroft and Karp's algorithm constructs dags (directed acyclic graphs) of SAPs (shortest augmenting paths), as explained in that prefascicle. Vertices of the dag are partitioned into sets of girls G_l and boys B_l at level l. The following implementation uses an interesting combination of data structures. First there are "mate tables" to represent the current matching, with `GMATE`$[g]$ for $1 \leq g \leq M$ and `BMATE`$[b]$ for $1 \leq b \leq N$ to indicate the partners of girl g and boy b, or 0 if they're currently free.

The breadth-first construction of a dag is controlled by an array `QUEUE`$[k]$ for $0 \leq k < M$, which records the girls currently present. If f girls are free, they appear in the first f positions of `QUEUE`. There's also a partial inverse, `IQUEUE`$[g]$ for $1 \leq g \leq M$: If $0 \leq k < f$ and `QUEUE`$[k] = g$, then `IQUEUE`$[g] = k$. Yet another array, `MARK`$[b]$ for $1 \leq b \leq N$, equals l if $b \in B_l$; otherwise `MARK`$[b] = 0$. There's also `MARKED`$[t]$, for $0 \leq t < N$; it lists the boys for which `MARK`$[b] \neq 0$.

The algorithm also involves a depth-first process, to remove SAPs after the dag has been built. Those steps use the array `STACK`$[l]$, for $0 \leq l < M$, to remember the boy of B_l who is currently being visited.

The bipartite graph that underlies everything is represented sparsely as a collection of *edge nodes*, each of which contains four fields `GTIP`, `BTIP`, `GNEXT`, `BNEXT`. An edge between girl g and boy b is represented by an edge node e for which `GTIP`$(e) = g$ and `BTIP`$(e) = b$; here $1 \leq e \leq E$, where E is the total

Hopcroft
Karp
Bipartite matching
Internet
matching
dags
data structures
mate tables
breadth-first
depth-first process
sparse graph representation
edge nodes

maximum cardinality matching

number of edges. The first edge involving g, for $1 \le g \le M$, is GLINK[g]; the next one is GNEXT(GLINK[g]); and so on, until 0 terminates the list. The values of GTIP, BTIP, and GNEXT remain fixed throughout the computation.

A similar convention is used to represent the dag, which is constructed dynamically: The first arc from boy b in the dag is BLINK[b], for $1 \le b \le N$, and the next is BNEXT(BLINK[b]), etc. The contents of BLINK and BNEXT are therefore *not* fixed. Every girl g in the dag is the source of exactly one arc, which leads to GMATE[g]. If GMATE[g] $= 0$, that arc leads to $\perp$.

Algorithm H (*Maximum bipartite matching*). Given a bipartite graph with M girls, N boys, and E edges, represented as explained above, this algorithm computes a maximum cardinality matching, which will appear in the GMATE and BMATE arrays. It also uses the auxiliary arrays QUEUE, IQUEUE, MARK, MARKED, and STACK, defined above. The MARK array must be initially zero.

H1. [Prime the pump.] Set GMATE and BMATE to a maximal (not necessarily maximum) matching; also set f to the number of unmatched girls, and list them in the first f slots of QUEUE.

H2. [Start building the dag.] Set $t \leftarrow i \leftarrow l \leftarrow r \leftarrow 0$, $q \leftarrow f$, and $L \leftarrow 0$.

H3. [Begin level $l+1$.] (At this point the girls of G_l are listed in QUEUE[k] for $i \le k < q$, and the dag contains t boys.) Set $q' \leftarrow q$.

H4. [Process a $g \in G_l$.] Go to H10 if $i = q'$. Otherwise set $g \leftarrow$ QUEUE[i], $i \leftarrow i + 1$, and $e \leftarrow$ GLINK[g].

H5. [Let b be a suitor for g.] If $e = 0$, return to H4; otherwise set $b \leftarrow$ BTIP(e).

H6. [Is b new?] If MARK[b] $= 0$, go to H8. Otherwise if MARK[b] $> l$, set BNEXT(e) $\leftarrow$ BLINK[b], BLINK[b] $\leftarrow e$.

H7. [Loop on b.] Set $e \leftarrow$ GNEXT(e) and return to H5.

H8. [Enter b into B_{l+1}.] If $L > 0$ and BMATE[b] $\ne 0$, go to H7. Otherwise set MARK[b] $\leftarrow l+1$, MARKED[t] $\leftarrow b$, $t \leftarrow t+1$, BLINK[b] $\leftarrow e$, BNEXT(e) $\leftarrow 0$.

H9. [Is b free?] If BMATE[b] $\ne 0$, set QUEUE[q] $\leftarrow$ BMATE[b], $q \leftarrow q + 1$. Otherwise if $L = 0$, set $L \leftarrow l + 1$, $r \leftarrow 1$, $q \leftarrow q'$ (we've reached the final level). Otherwise set $r \leftarrow r+1$ (there are r free boys on level L). Go to H7.

H10. [Is the dag complete?] If $q \ne q'$, set $l \leftarrow l+1$ and return to H3. (Otherwise the dag is complete, and the last r elements of MARKED are the free boys in B_L.) Terminate the algorithm if $L = 0$ (there are no augmenting paths).

H11. [Start to find a SAP.] If $r = 0$, set MARK[MARKED[k]] $\leftarrow 0$ for $0 \le k < t$ and return to H2. Otherwise set $b \leftarrow$ MARKED[$t - r$], $r \leftarrow r - 1$, $l \leftarrow L$.

H12. [Enter level l.] Set STACK[l] $\leftarrow b$.

H13. [Advance.] Set $e \leftarrow$ BLINK[b], and go to H15 if $e = 0$. Otherwise set BLINK[b] $\leftarrow$ BNEXT(e), $g \leftarrow$ GTIP(e). If MARK[GMATE[g]] < 0, repeat this step (g has been deleted). Otherwise set $b \leftarrow$ GMATE[g].

H14. [SAP complete?] If $b = 0$ (g is free), go to H16. Otherwise set $l \leftarrow l - 1$ and return to H12.

backtracking
breadth-first
depth-first
certificate of correctness
maximum independent set in bipartite graph

H15. [Resume higher level.] Set $l \leftarrow l+1$. Then go to H11 if $l > L$; otherwise set $b \leftarrow$ `STACK[`l`]` and go back to H13. (This is like "backtracking," except that we never retrace a step because we're destroying the dag as we go.)

H16. [Prepare to augment.] (At this point $l = 1$; $g = g_0$ and `STACK[1]` $= b_1$ in a SAP. The other boys are `STACK[2]`, ..., `STACK[`L`]`.) Set $f \leftarrow f - 1$, $k \leftarrow$ `IQUEUE[`g`]`, $i \leftarrow$ `QUEUE[`f`]`, `QUEUE[`k`]` $\leftarrow i$, and `IQUEUE[`i`]` $\leftarrow k$. (Those operations removed g from the list of free girls.) Set $b \leftarrow$ `STACK[1]`.

H17. [Augment.] Set `MARK[`b`]` $\leftarrow -1$, $g' \leftarrow$ `BMATE[`b`]`, `BMATE[`b`]` $\leftarrow g$, and `GMATE[`g`]` $\leftarrow b$. Then if $g' \neq 0$, set $g \leftarrow g'$, $l \leftarrow l+1$, $b \leftarrow$ `STACK[`l`]`, and repeat this step. Otherwise go back to H11. ▮

This algorithm has many steps, but it's not frighteningly complicated. It essentially consists of two separate-but-cooperating subalgorithms, namely the breadth-first dag construction in H2–H10 and the depth-first dag deconstruction in H11–H17.

Algorithm H comes with an important free bonus: After it has found a supposedly maximum matching, its data structures contain enough information to convince any skeptic that the matching is indeed as large as possible. Indeed, if no girl is free, the matching is perfect and obviously optimum. Otherwise the girls in `QUEUE[`k`]` for $0 \leq k < q$ are adjacent to only t boys in the graph, namely the boys in `MARKED[`k`]` for $0 \leq k < t$. And it's easy to verify that $q = t + f$; hence any matching must leave at least f girls without a partner. Indeed, Algorithm H provides us with a maximum independent set,

$$I = \{g \mid g \text{ is a girl in the final dag}\} \cup \{b \mid b \text{ is a boy not in the final dag}\},$$

which is certified by the maximum matching and vice versa!*

Algorithm H's main claim to fame, however, is that it runs remarkably fast. Give it a graph, and it churns out a maximum matching, lickety-split. The reason is that SAPs are extremely good augmenters:

Theorem H. *Let s be the size of a maximum matching. When $r = 0$ in step H11, the size of the current matching is at least $\frac{L}{L+1}s$.*

Proof. If the current matching has s' edges, we've observed that at least $s - s'$ vertex-disjoint augmenting paths exist. We also know that each of those paths contains at least $L+1$ edges of a maximum matching. So $s \geq (L+1)(s-s')$. ▮

Corollary K. *The running time for Algorithm H to find a maximum matching of size s is $O\big((M+N+E)\sqrt{s}\big)$.*

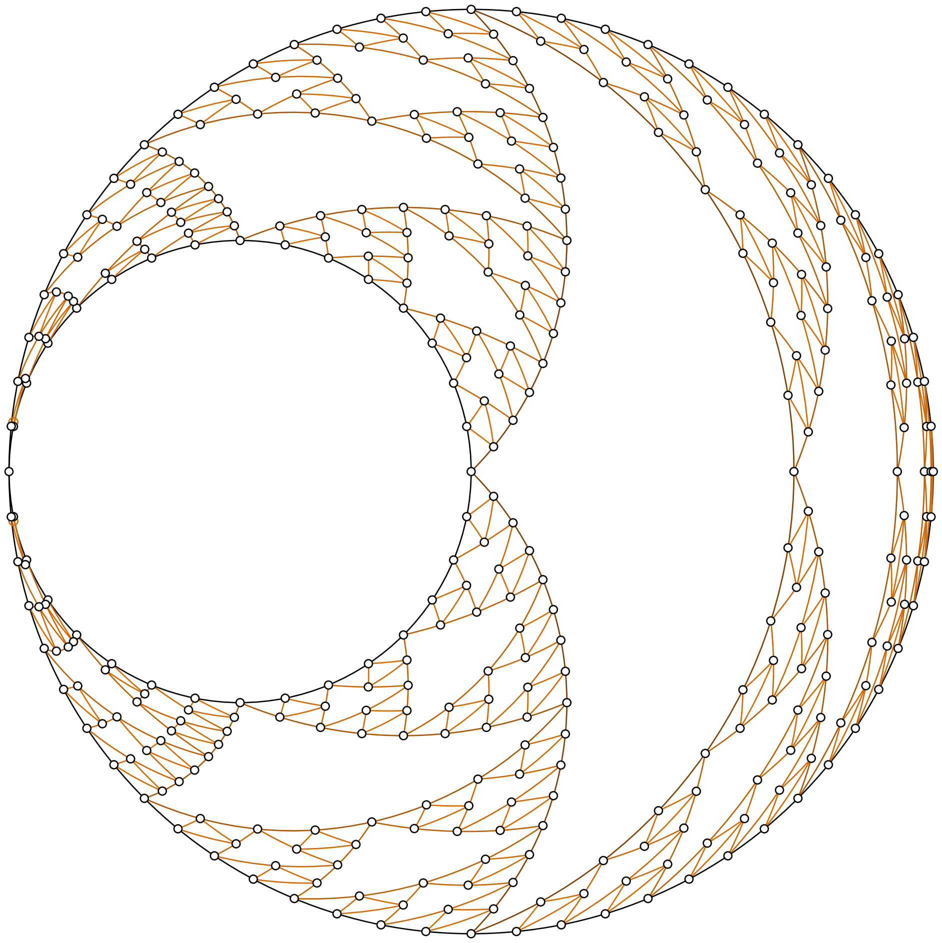
Proof. Every time a dag is constructed, the value of L increases. Each round of construction and deconstruction clearly involves $O(M+N+E)$ steps. If the algorithm hasn't terminated before the value of L exceeds $\sqrt{s}$, a matching of size $\geq \frac{\sqrt{s}-1}{\sqrt{s}}s = s - \sqrt{s}$ has been found, and $\sqrt{s}$ more rounds will complete the task. ▮

* The complement of I is a vertex cover containing C vertices, where C is the size of the matching found. No vertex cover can contain fewer than C vertices; hence I is maximum.

CONTENTS

I always thought Volume 4 was a myth,
like the missing part of the Dead Sea scrolls.
— BILL GASARCH (blog post, 10 January 2008)

pinched gasket



December 5, 2024

His Lady sad to see his sore constraint,
Cride out, Now now Sir knight, shew what ye bee.
— EDMUND SPENSER, *The Faerie Queene* (1590)

The work under our labour grows, Luxurious by restraint.
— JOHN MILTON, *Paradise Lost* (1667)

Liberty exists in proportion to wholesome restraint.
— DANIEL WEBSTER (1847)

It is impossible to be an artist and not care for laws and limits.
Art is limitation; the essence of every picture is the frame.
— GILBERT K. CHESTERTON, *Orthodoxy* (1908)

I surround myself with obstacles.
Whatever diminishes my discomfort diminishes my strength.
The more constraints one imposes, the more one
frees one's self of the chains that shackle the spirit.
— IGOR STRAVINSKY, *Poétique musicale sous forme de six leçons* (1939)

Knight of Holinesse
SPENSER
MILTON
WEBSTER
CHESTERTON
STRAVINSKY
constraint satisfaction problem–
CSP: The constraint sat prob
XCC problems
exact covering with colors
items
options
SAT problems
Boolean satisfiability
satisfiability
literals
clauses
variables
domains
constraints
nogood
SAT as CSP

7.2.2.3. Constraint satisfaction. In Section 7.2.2.1 we solved numerous examples of XCC problems — exact covering with colors — which featured "items" and "options." Then in Section 7.2.2.2 we resolved lots of SAT problems — Boolean satisfiability — which featured "literals" and "clauses." All of these, and more, are instances of a combinatorial challenge that's more general yet, the *constraint satisfaction problem* — often called the CSP for short — which we will see is based on "variables," "domains," and "constraints."

The idea is simple: We're given a finite list of *variables* $(x_1, x_2, \ldots, x_n)$, to which we can assign values that belong to given finite *domains* $(D_1, D_2, \ldots, D_n)$. And we're also given a set of *constraints* $\{R_1, R_2, \ldots, R_m\}$, each of which specifies that a certain subset of the values $(x_1, x_2, \ldots, x_n)$ must be mutually compatible. Some combinations of values are "good"; the others are "nogood."

For example, let $n = 5$, and suppose that each domain is a set of letters:

$$D_1 = \{\mathtt{B},\mathtt{S}\},\ \ D_2 = \{\mathtt{C},\mathtt{L}\},\ \ D_3 = \{\mathtt{A},\mathtt{I},\mathtt{U}\},\ \ D_4 = \{\mathtt{E},\mathtt{O}\},\ \ D_5 = \{\mathtt{D},\mathtt{N}\}. \tag{1}$$

Thus there are $2\times2\times3\times2\times2 = 48$ possible settings of $x_1x_2x_3x_4x_5$, from BCAED to SLUON. Let's also impose three constraints:

$$\begin{aligned} R_1(x_1,x_3,x_5) &= \text{`}x_1x_3x_5 \in \{\mathtt{BAN},\mathtt{BUD},\mathtt{SIN}\}\text{'};\\ R_2(x_1,x_4) &= \text{`}x_1x_4 \in \{\mathtt{BE},\mathtt{SE},\mathtt{SO}\}\text{'};\\ R_3(x_2,x_4,x_5) &= \text{`}x_2x_4x_5 \in \{\mathtt{COD},\mathtt{CON},\mathtt{LED}\}\text{'}. \end{aligned} \tag{2}$$

This CSP has two solutions, easily found by hand (see exercise 1).

Every SAT problem is obviously a CSP in which all the domains are $\{0,1\}$. For example, problem $F = \{1\bar{2}, 23, \bar{1}\bar{3}, \bar{1}\bar{2}3\}$ in 7.2.2.2–(3) has four constraints,

$$x_1x_2 \in \{00,10,11\};\quad x_2x_3 \in \{01,10,11\};\quad x_1x_3 \in \{00,01,10\};$$
$$x_1x_2x_3 \in \{000,001,010,011,100,101,111\}. \tag{3}$$

Conversely, every CSP can be formalized as an equivalent SAT problem, by using several SAT variables to represent each CSP variable x whose domain size d exceeds 2. For example, if the domain is $\{0,1,\ldots,d-1\}$, Section 7.2.2.2 discussed the "log encoding," with $l = \lceil \lg d \rceil$ Boolean variables meaning that $x = (x_{l-1}\ldots x_1 x_0)_2$. There's also the "direct encoding," with d variables $x_k = [x=k]$, as well as the "order encoding," which has $d-1$ variables $x^j = [x \ge j]$. We also discussed a variety of ways to represent arbitrary constraints, in the form of one or more clauses involving such Boolean variables. Each of those encodings has its own virtues and weaknesses, depending on the application.

Every XCC problem can, similarly, be regarded as a CSP. One way is to have a variable x_i for every primary item i, with domain D_i equal to the set of options that contain i. The constraints are that x_i and x_j cannot be options that conflict: If $x_i = o_i$ and $x_j = o_j$, where $o_i \ne o_j$, then o_i and o_j cannot have a common primary item, nor can they have a common secondary item that's colored differently in o_i and o_j. Conversely, exercise 7.2.2.1–100 presented one way to encode any CSP as an XCC problem.

Thus XCC, SAT, and CSP can each be reduced to the other two.

We've already learned how to construct excellent XCC solvers and excellent SAT solvers, so we might be tempted to stop there, regarding CSP as a problem that's already been well solved. But we shall see that careful consideration of the CSP not only clarifies XCC and SAT, it also teaches us important new methods.

Related models. Many groups of researchers have independently adopted conceptual frameworks that are identical to or very similar to the notions of variables, domains, and constraints. For example, a theory of *relational structures* has been developed as part of the branch of mathematics called "model theory," which spawned "universal algebra." A relational structure is a set D together with a set $\{R_1, R_2, \ldots\}$ of relations or "predicates" defined on the elements of D. Each relation R_i depends on k elements, for some $k = k_i$, and it defines the k-tuples of elements for which that predicate is true. [See P. M. Cohn, *Universal Algebra* (1965), Chapter V.]

Let's be a little more precise. The *Cartesian product* of sets $(D_1,\ldots,D_n)$, denoted by $D_1 \times \cdots \times D_n$, is the set of all n-tuples $(x_1,\ldots,x_n)$ such that $x_i \in D_i$ for $1 \le i \le n$. Thus, $D_1 \times \cdots \times D_n$ is the set of all solutions to a CSP with domains $(D_1,\ldots,D_n)$, in the case when there are no constraints. An n-tuple such as $(x_1,\ldots,x_n)$ is often written simply as $x_1 \ldots x_n$, when commas aren't necessary. We also write $D \times \cdots \times D = D^n$ when the n domains are all identical.

A k-ary *relation* on sets $(D_1,\ldots,D_k)$ is a subset of $D_1 \times \cdots \times D_k$. We write either $R(x_1,\ldots,x_k)$ or $x_1 \ldots x_k \in R$ when we want to say that the k-tuple $(x_1,\ldots,x_k)$ satisfies relation R. The relation is called *binary* when $k=2$, *ternary* when $k=3$, *quaternary* when $k=4$, and so on; it's *unary* when $k=1$. (Strictly speaking, there also are *nullary* relations; see exercise 5.)

The simplest nontrivial relational structures arise where there's just a single binary relation. In fact, this case is so simple, we hardly ever think of it as a "relation structure" at all: We call it a *directed graph*. Indeed, we know well

CSP as SAT
log encoding
Boolean variables
direct encoding
order encoding
XCC as CSP
primary item
options
secondary item
CSP as XCC
relational structures
model theory
universal algebra
predicates
Cohn
Cartesian product
tuples
commas
relation
binary
ternary
quaternary
unary
nullary
directed graph

that a directed graph is a set V of *vertices*, together with a set $A \subseteq V \times V$ of *arcs*; and that's exactly what it means to be a relational structure with a single binary relation. This case is so common, we usually use the special notation $u \longrightarrow v$, instead of writing $A(u,v)$ or $uv \in A$.

Furthermore, if the lone binary relation is symmetric (meaning that $u \longrightarrow v$ implies $v \longrightarrow u$) and irreflexive (meaning that $v \not\longrightarrow v$), we usually call it E instead of A; and we write $u \mathbin{—\!—} v$ instead of writing $E(u,v)$ or $uv \in E$. In such cases, of course, we have an ordinary (undirected) *graph*, and E is its set of *edges*.

Now let's consider *two* graphs, $G = (V,E)$ and $G' = (V',E')$. Suppose we attach a label $h(v)$ to every vertex $v \in V$, where $h(v)$ belongs to V'. This mapping $h : V \to V'$ is called a *homomorphism* if $E(u,v)$ implies $E'(h(u),h(v))$; in other words, it's a homomorphism if we have

$$h(u) \mathbin{—\!—} h(v) \text{ in } G' \qquad \text{whenever } u \mathbin{—\!—} v \text{ in } G. \tag{4}$$

For example, if G' is the complete graph K_d on vertices $V' = \{1,2,\ldots,d\}$, we have $j \mathbin{—\!—} k$ in G' if and only if $j \neq k$. So *h is a homomorphism from G to K_d if and only if it's a way to color the vertices of G properly with d colors.*

Going the other way, suppose G (not G') is the complete graph K_d. It's easy to see that *h is a homomorphism from K_d to G' if and only if the vertices $\{h(1),\ldots,h(d)\}$ form a d-clique in G'.*

Things get even more interesting when there's more than one relation. If $S = (D,R_1,\ldots,R_t)$ and $S' = (D',R'_1,\ldots,R'_t)$ are relational structures, we say that S and S' are *similar* if R_i and R'_i both have the same "arity," for $1 \le i \le t$. (In other words, R_i and R'_i are both k_i-ary.) In such cases we define a homomorphism h from S to S' to be a mapping from D to D' such that

$$R_i(x_1,\ldots,x_{k_i}) \quad \text{implies} \quad R'_i(h(x_1),\ldots,h(x_{k_i})), \quad \text{for } 1 \le i \le t. \tag{5}$$

For example, consider the augmented graph structure $G^{\neq} = (V,E,\neq)$ whose relations include the nonequality relation '$\neq$' as well as the ordinary edge relation E. A homomorphism from $G^{\neq}$ to $G'^{\neq}$ now has *two* properties:

$$h(u) \mathbin{—\!—} h(v) \text{ in } G' \text{ whenever } u \mathbin{—\!—} v \text{ in } G; \quad h(u) \neq h(v) \text{ whenever } u \neq v. \tag{6}$$

Consequently G is *embedded* in G': The vertices $\{h(v) \mid v \in V\}$ and edges $\{h(u) \mathbin{—\!—} h(v) \mid u \mathbin{—\!—} v \text{ in } G\}$ form a subgraph of G' that's essentially a copy of G. If, for instance, G is the n-cycle C_n, h proves that G' contains an n-cycle.

Sometimes a k-ary relation is *constant*, meaning that it is satisfied by only a single k-tuple. One interesting example is the structure $S = (V,A,\{ab\})$, where (V,A) is a digraph with special vertices a and b. Then a homomorphism h from S to the relational structure $S' = (\{0,1\},=,\neq)$ will tell us that $u \longrightarrow v$ implies $h(u) = h(v)$, and also that $h(a) \neq h(b)$. Hence every vertex v reachable from a will have $h(v) = h(a)$, and we can conclude that b is unreachable. Conversely, if b isn't reachable from a, such a homomorphism can easily be found.

The evident versatility of homomorphisms has led Peter Jeavons to define the *general combinatorial problem* (GCP) as follows: "Given a pair of similar relational structures S and S', is there a homomorphism from S to S'?" [See *Theo-*

vertices
arcs
symmetric
irreflexive
graph
edges
homomorphism
complete graph
clique
similar
arity
nonequality relation
disequality, see nonequality
embedded
subgraph isomorphism, see embedded graphs
subgraph
copy
constant
reachable
Jeavons
general combinatorial problem
GCP

retical Computer Science **200** (1998), 185–204; see also T. Feder and M. Y. Vardi, *SICOMP* **28** (1998), 57–104.] Exercises 9 and 10 provide further examples.

In particular, Jeavons observed that the CSP is indeed a special case of the GCP. To cast (1) and (2) in this framework, for example, we let

$$S = (\{1,2,3,4,5\},\{1\},\{2\},\{3\},\{4\},\{5\},\{135\},\{14\},\{245\}); \qquad (7)$$

$$S' = (\{\mathtt{A},\ldots,\mathtt{Z}\}, D_1, D_2, D_3, D_4, D_5, R_1, R_2, R_3); \qquad (8)$$

here D_1 through D_5 are the domains in (1), and R_1 through R_3 are the tuples in (2). The general idea is to have only *constant* relations in S, and to put the domains and constraints into S'. A homomorphism h from S to S' will then give us the values $(h(1),\ldots,h(5)) = (x_1,\ldots,x_5)$ that simultaneously belong to the domains and satisfy the constraints.

Conversely, every GCP is readily seen to be a CSP. (See exercise 11.)

The CSP framework is also intimately connected with the theory of *relational databases*. Individual facts in such a database are sets of tuples, involving the values of variables called "attributes." For example, we might have five attributes called 'location', 'employee', 'manager', 'job', 'language', and three relations:

Departments			Layout		Personnel		
location	manager	language	location	job	employee	job	language
basement	Alice	norsk	basement	test	Chris	code	deutsch
basement	Udo	deutsch	solarium	test	Chris	code	norsk
solarium	Iris	norsk	solarium	code	Logan	test	deutsch

What combinations of (location, employee, manager, job, language) exist in this peculiar institution? They correspond precisely to the solutions to the CSP in (1) and (2)! Database theorists call this the *natural join* of the three relations. [See E. F. Codd, *CACM* **13** (1970), 377–387; H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database Systems: The Complete Book* (Prentice–Hall, 2002).]

***Statistical mechanics.** Similar ideas arise also when physicists conceive of the universe as a gigantic collection of discrete particles, each of which has its own "spin." If there are N particles, the overall state is then an N-tuple $\Sigma = \sigma_1 \ldots \sigma_N$ called a *configuration*, where σ_j is the jth spin. Different particles can have different kinds of quantized spins, belonging to a given finite space of possible values, exactly analogous to the domains in a CSP.

Every configuration Σ has an associated *energy* $E(\Sigma)$, which is usually the sum of contributions from particles that interact locally. For example, the "one-dimensional Ising model," formulated by W. Lenz and analyzed by his student E. Ising [*Zeitschrift für Physik* **31** (1925), 253–258], has the energy function

$$E(\Sigma) = -\sum_{j=1}^{N-1} \sigma_j\sigma_{j+1} - B\sum_{j=1}^{N} \sigma_j, \qquad (9)$$

where each spin σ_j is ± 1, and where the constant B represents the strength of an external magnetic field. If $\sigma_{j-1} = \sigma_{j+1} = -\sigma_j$ and $B\sigma_j < 2$, particle j will tend to change its spin to match its neighbors, because that would reduce the energy.

Feder
Vardi
Jeavons
relational databases
attributes
natural join
join
Codd
Garcia-Molina
Ullman
Widom
Statistical mechanics
physics–
spin
configuration
energy
Ising model
Lenz
Ising

Any set of k-ary relations between particles can be used to define energy functions. So, in particular, we can cast the CSP of (1) and (2) into this mold, obtaining configurations in $D_1 \times \cdots \times D_5$ whose energy function is

$$E(\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5) = -[R_1(\sigma_1,\sigma_3,\sigma_5)] - [R_2(\sigma_1,\sigma_4)] - [R_3(\sigma_2,\sigma_4,\sigma_5)]. \qquad (10)$$

Here are the 48 possibilities, together with their associated energy levels:

Σ	$E(\Sigma)$	Σ	$E(\Sigma)$	Σ	$E(\Sigma)$	Σ	$E(\Sigma)$	Σ	$E(\Sigma)$	Σ	$E(\Sigma)$
BCAED	-1	BCUED	-2	BLIED	-2	SCAED	-1	SCUED	-1	SLIED	-2
BCAEN	-2	BCUEN	-1	BLIEN	-1	SCAEN	-1	SCUEN	-1	SLIEN	-2
BCAOD	-1	BCUOD	-2	BLIOD	0	SCAOD	-2	SCUOD	-2	SLIOD	-1
BCAON	-2	BCUON	-1	BLION	0	SCAON	-2	SCUON	-2	SLION	-2
BCIED	-1	BLAED	-2	BLUED	-3	SCIED	-1	SLAED	-2	SLUED	-2
BCIEN	-1	BLAEN	-2	BLUEN	-1	SCIEN	-2	SLAEN	-1	SLUEN	-1
BCIOD	-1	BLAOD	0	BLUOD	-1	SCIOD	-2	SLAOD	-1	SLUOD	-1
BCION	-1	BLAON	-1	BLUON	0	SCION	-3	SLAON	-1	SLUON	-1

To analyze such models, physicists essentially calculate the generating function $G(z) = \sum z^{E(\Sigma)}$, summed over all configurations Σ. In our case, for example, $G(z) = 2z^{-3} + 18z^{-2} + 24z^{-1} + 4$. But because physicists understand physics, they do this in an idiosyncratic way by setting $z = e^{-\beta}$, where β is the reciprocal of the "temperature." In other words, they calculate $\sum e^{-\beta E(\Sigma)}$, which they call the *partition function*; and they usually denote that sum by $Z(\beta)$.

Since the partition function is always a sum of positive terms, physicists consider the ratio $e^{-\beta E(\Sigma)}/Z(\beta)$ to be the *probability* of configuration Σ. [Such probability distributions were introduced in the 19th century by Ludwig Boltzmann; see, for example, the *Sitzungsberichte der Mathematisch-Naturwissenschaftlichen Classe der Kaiserlichen Akademie der Wissenschaften* **76** (Wien, 1877), 373–435.]

At high temperatures, β is near 0; hence all configurations are almost equally likely. But at low temperatures, β approaches ∞; then only the configurations with smallest possible energy, the so-called "ground states," are significant, because they are exponentially more probable than any other state. In our 48-state example, each of the configurations with energy -3 occurs with probability $\frac{1}{48} + \frac{13}{384}\beta + O(\beta^2)$ when $\beta \to 0$, but probability $\frac{1}{2} - \frac{9}{2}e^{-\beta} + O(e^{-2\beta})$ when $\beta \to \infty$.

Thus, in general, the solutions to a satisfiable CSP correspond to the ground states of the associated physical problem. And when the CSP is *unsatisfiable*, the ground states satisfy as many of the constraints as possible.

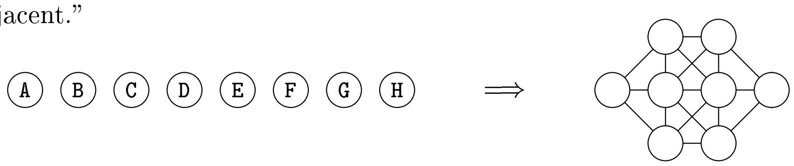
Considerations such as these account for the fact that physicists have contributed significantly to the understanding of random satisfiability problems, in particular by introducing Algorithm 7.2.2.2S. Further discussion of statistical mechanics is, of course, beyond the scope of a book on computer programming; but readers hungry for more may consult *Information, Physics, and Computation* by Marc Mézard and Andrea Montanari (Oxford University Press, 2009).

The takeaway message from all these examples is obvious: There has to be something good about the CSP notions of variables, domains, and constraints, when we want to model real-world problems, because so many people have independently come up with essentially the same framework.

generating function
temperature
partition function
probability
Boltzmann
ground states
MAXSAT
random satisfiability
Mézard
Montanari

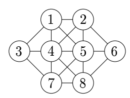
hello world
crystal maze puzzle–
global
all-different

A simple example. To warm up, let's look at a little puzzle that appeared on a British TV show called *The Crystal Maze* in 1994. The task is simple — but you've got only two minutes to do it: "Place eight large disks, marked with the letters A through H, onto the eight circles shown; consecutive letters can't be adjacent."



(11)

We're actually facing two challenges here, namely (i) solve the puzzle; and (ii) express it as a constraint satisfaction problem, so that a computer can solve it for us. We'll tackle (ii), so as not to spoil the fun of (i). And we'll allow ourselves ten minutes, say, to accomplish goal (ii).



What are appropriate variables, domains, and constraints? We'd better label the vertices of the graph, so that we can readily describe what we want to define. One approach, based on the labeling shown, is to have eight variables $\{x_1, x_2, \ldots, x_8\}$, one for each vertex, each with domain $\{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{H}\}$. Then there are seventeen constraints, one for each edge of the graph; for example, the constraint for edge $1 — 2$ is

$$x_1x_2 \in \{\mathtt{AC,AD,AE,AF,AG,AH,BD,BE,BF,BG,BH,CA,CE,CF,CG,CH,DA,DB,DF,DG,DH,EA,}\\ \mathtt{EB,EC,EG,EH,FA,FB,FC,FD,FH,GA,GB,GC,GD,GE,HA,HB,HC,HD,HE,HF}\}, \quad (12)$$

and the same relation is used for all of the other edges. It can be written much more succinctly, if we assume that the letters are represented by integer codes:

$$|x_1 - x_2| > 1. \quad (13)$$

OK, that took three minutes. Are we done? Well, no, actually; the seventeen constraints we've specified do not obviously rule out the possibility that $x_1 = x_8$. We're not allowed to put a disk on two different circles.

We could add eleven further constraints, namely $x_i \neq x_j$ for each of the yet-unconstrained pairs. But seasoned CSP solvers generally prefer to append a single *global* constraint instead, involving all of the variables at once:

$$x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6,\ x_7,\ x_8 \text{ are all different.} \quad (14)$$

Indeed, special methods have been devised for the "all-different" constraint, because it arises in so many different problems. With (14), we've satisfied (ii).

Five minutes to go. Is there a better way? Another possibility is to let the variables be $\{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{H}\}$, one for each *disk*, each with domain $\{1, 2, \ldots, 8\}$. Then only *seven* constraints are needed, one for each pair of consecutive letters; e.g.,

$$\mathtt{AB} \in \{16,17,18,23,27,28,32,35,36,38,46,53,61,63,64,67,71,72,76,81,82,83\}. \quad (15)$$

And each of these constraints has only 22 tuples, compared to 42 in (12). It's a win! Of course we also need the global all-different constraint. (See exercise 13.)

If we only had more time, we could have discovered a completely different way to model problem (11) as a CSP, such as the approach in exercise 16.

car sequencing problem
CSPLIB
benchmarks
Gent
Walsh
reflection
mirror images
symmetry breaking

Automating automobiles. We've already seen dozens and dozens of significant examples of constraint-based problems when we studied exact covering and SAT. But we certainly haven't exhausted all of the major applications, and several problems on our yet-unexamined list have been associated historically with the CSP. One of them, known as the *car sequencing problem*, is especially appropriate for us to study next, not only because its initials are "CSP" but also because it is problem 001 in CSPLIB — a noteworthy collection of benchmarks that was launched by I. P. Gent and T. Walsh in 1999 (see *LNCS* **1713** (1999), 480–481).

Consider the portion of an automobile assembly line where optional features are being installed on newly made vehicles. Some of the cars will be made with moonroofs; some will have heated seats; and so on. The assembly line is divided into work areas, one for each special feature. Work area w has space for q_w cars, where q_w is the number of time slots needed to install feature w as the conveyor belt moves the cars along. If at most p_w/q_w of the cars need that feature, p_w installers are on duty, one of whom will commence work when a car enters the area and walk with it until the installation is done. The car sequencing problem is the task of arranging a given set of cars into a sequence so that *no subsequence of q_w consecutive cars will include more than p_w that need feature w.*

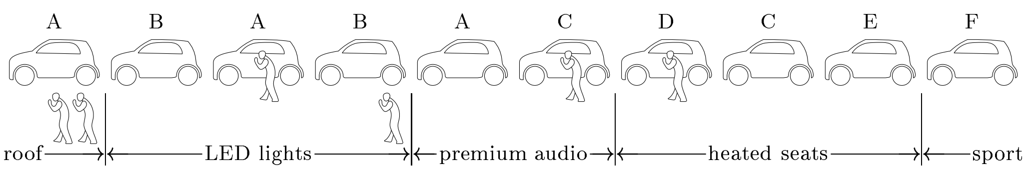


Fig. 100. Cars of models A, B, ... enter this assembly line at the far right, receiving optional features when they're in an appropriate work area. If this sequence has specifications (16), the final car (F) will be delayed in the LED area, because three cars in a row want that feature. The car sequencing problem tries to avoid such delays.

For example, there might be six models using the following subsets of five features:

Model	A	B	C	D	E	F	w	p_w	q_w
moonroof?	☐	☑	☐	☑	☑	☐	0	2	5
LED lights?	☑	☐	☑	☐	☑	☑	1	2	3
premium audio?	☐	☑	☑	☐	☐	☑	2	1	2
heated seats?	☐	☐	☐	☑	☐	☑	3	1	3
sport suspension?	☐	☐	☐	☐	☑	☐	4	1	5

(16)

Suppose ten cars of models $\{A, A, A, B, B, C, C, D, E, F\}$ are to be made. The sequence ABABACDCEF is almost correct, but it fails on the final car (see Fig. 100). Can you find a delay-free sequence? Notice that the left-right reflection of any solution is also a solution; we can rule out mirror images by requiring that model F, say, appears among the first five cars. Exercise 17 has the (unique) answer.

The car sequencing problem has boundary effects at the left and right that make it somewhat unrealistic. (Industrial assembly lines don't really start out empty every day!) Still, it's a nice clean problem, instructive to chew on.

One way to formulate the car sequencing problem in terms of variables, domains, and constraints is to have one variable x_i for every "time slot" in the assembly line sequence. The domain of each x_i is the set of model types, for $0 \le i < t$, where t is the total number of cars to be produced. We can also introduce t inverse variables, one for each vehicle, telling which slot it occupies; those variables have the domain $\{0, 1, \ldots, t-1\}$.

Our example of the 10 cars in Fig. 100 and (16) would therefore have 10 variables $\{x_0, \ldots, x_9\}$ with the 6-element domain $\{\mathrm{A}, \ldots, \mathrm{F}\}$, plus 10 variables $\{a_0, a_1, a_2, b_0, b_1, c_0, c_1, d, e, f\}$ with the 10-element domain $\{0, \ldots, 9\}$. These variables are related to each other by so-called "channelling constraints": For example, we can't have $a_1 = j$ unless $x_j = \mathrm{A}$; and in general the slot occupied by each vehicle must have the corresponding model type. We also constrain $a_0 < a_1 < a_2$, $b_0 < b_1$, and $c_0 < c_1$, so that vehicles of the same type are properly ordered in the overall sequence. (Notice that the number of ways to satisfy the stated constraints between these 20 variables is exactly $10!/(3!\,2!\,2!\,1!\,1!\,1!) = 151200$, which is the number of permutations of the multiset $\{\mathrm{A}, \mathrm{A}, \mathrm{A}, \mathrm{B}, \mathrm{B}, \mathrm{C}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\}$. We could cut that number in half by requiring $f < 5$; see exercise 18.)

We also need constraints to rule out bad situations, like the subsequence $x_7x_8x_9 = \mathrm{CEF}$ that delays the lineup in Fig. 100. For this purpose it's convenient to introduce Boolean variables f_{iw} for $0 \le i < t$ and $0 \le w < m$, where m is the number of optional features and $f_{iw} = 1$ if and only if the car in slot i has feature w. There are channelling constraints between x_i and f_{iw}; for example, $x_i = \mathrm{B}$ implies that $f_{i0}f_{i1}f_{i2}f_{i3}f_{i4} = 10100$. The assembly-line constraints are then

$$f_{iw} + f_{(i+1)w} + \cdots + f_{(i+q_w-1)w} \le p_w, \quad \text{for } 0 \le i \le t - q_w \text{ and } 0 \le w < m. \tag{17}$$

For example, $x_7x_8x_9 = \mathrm{CEF}$ causes $f_{71}f_{81}f_{91} = 111$, violating $f_{71} + f_{81} + f_{91} \le 2$.

OK, it looks like we're done. Given any car sequencing problem with t cars and m features, we've now defined $t(2+m)$ variables, and devised sufficient constraints to characterize all the solutions. It turns out, however, that we could actually find those solutions much faster by adding *additional* constraints: If r_w is the total number of cars that will receive feature w, we must also have

$$f_{0w} + f_{1w} + \cdots + f_{(t-lq_w-1)w} \ge r_w - lp_w, \quad \text{for } 0 < l < \lceil r_w/p_w \rceil \text{ and } 0 \le w < m. \tag{18}$$

The reason is that the final lq_w cars in the sequence cannot account for more than lp_w of the total. (In our example, $r_1 = 7$; hence (18) gives f_{01} when $l = 3$; the first car therefore cannot be of type B or D.) The constraints in (18) are redundant, yet a computer might not be able to think of them, and they can significantly reduce the size of the search tree. (See exercise 21.)

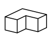
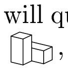
Of course the car sequencing problem can also be formulated as a CSP in many other ways, which will suggest themselves as we gain further experience.

Historical notes: Successful experiments with the car sequencing problem were first carried out by M. Dincbas, H. Simonis, and P. Van Hentenryck [*ECAI* **8** (1988), 290–295]. They were able to solve randomly generated problems with $t = 200$, $m = 5$, $(p_0/q_0, \ldots, p_4/q_4) = (1/2, 2/3, 1/3, 2/5, 1/5)$, and with overall utilization $r_w \approx .9tp_w/q_w$, by introducing the redundant constraints (18).

variables
domains
slot
inverse variables
channelling constraints
permutations of the multiset
multiset
Boolean variables
redundant
Historical notes
Dincbas
Simonis
Van Hentenryck

An international competition was held in 2005, based on actual industrial data. It included additional constraints, such as the colors of paint to be used and the initial contents of the assembly line, and it inspired many creative solutions. [See C. Solnon, V. D. Cung, A. Nguyen, and C. Artigues, *EJOR* **191** (2008), 912–927.] The winning programs were based on local search methods analogous to WalkSAT, using "greedy" heuristics.

Line labeling in computer vision. Speaking of history, let's turn now to some fascinating aspects of computer vision that influenced much of the early work on constraint processing. When a camera photographs a scene, it makes a two-dimensional image of three-dimensional reality; interesting problems arise when we try to reconstruct the reality from the image.

We'll work with an extremely simplified yet powerful model, as the original researchers did: Our "reality" will be a world of special polyhedral objects, where exactly *three faces* meet at each of the vertices. For example, an ordinary cube or tetrahedron or  will qualify. But an octahedron will not, nor will an Egyptian-style pyramid, nor , because a vertex where four faces meet isn't allowed. These three-faced concepts can be generalized, of course, but it's helpful to start with a thorough understanding of the comparatively simple trihedral world.

More precisely, the 3D objects we shall deal with have no curved surfaces. They are defined by vertices, edges, and faces, where the vertices are "corners" at which edges and faces come together. All of the faces are "flat," meaning that their points all lie on some plane. Each face is bounded by an exterior polygon, possibly with one or more interior polygons delimiting "holes" in the face. Each edge runs between two vertices and is part of the (infinite) line where the planes of two adjacent faces meet; it's a segment of the polygonal boundaries of those faces. And significantly, *each vertex is the endpoint of exactly three edges.* We shall call such an object a *three-valent polyhedral object*, or 3VP for short. (See Fig. 101.)

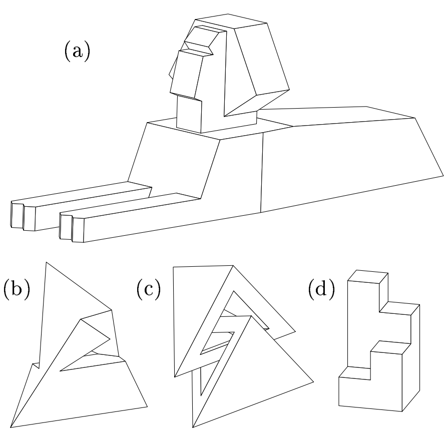
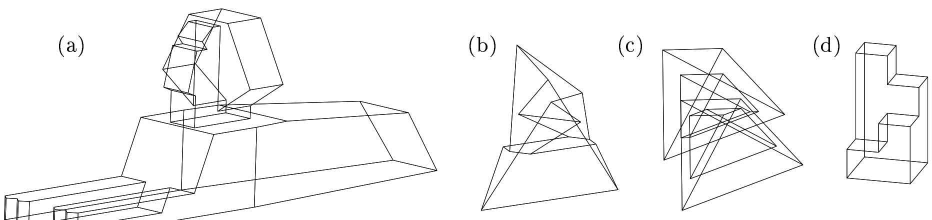


Fig. 101. Examples of 3VPs (three-valent polyhedra): (a) A stylized sphinx. [68 vertices, 102 edges, 38 faces.] (b) The Szilassi polyhedron, defined in exercise 26. Each of its seven faces is adjacent to all of the other six(!). [14 vertices, 21 edges, 7 faces.] (c) A clasp formed from two identical, interlocked objects, each of which is a tetrahedron from which a large triangular wedge has been hollowed out. [20 vertices, 30 edges, 14 faces.] (d) The histoscape for the matrix $\begin{pmatrix} 4 & 3 \\ 1 & 2 \end{pmatrix}$, as defined in exercise 27. [20 vertices, 30 edges, 12 faces.] Many of the vertices, edges, and faces of these examples are invisible because they lie behind the parts that we *can* see.

The two-dimensional images shown here make sense to us, somehow, although significant depth information has been lost. In some mysterious way we've learned to rely on visual cues in order to understand what's really present.

competition
contest
real-world data
Solnon
Cung
Nguyen
Artigues
WalkSAT
greedy
computer vision
vision
photographs
scene
faces
octahedron
pyramid
three-faced
trihedral world
3D objects
vertices
edges
faces
three-valent polyhedral object
Szilassi polyhedron
histoscape

Huffman
Clowes
half edge
valley folds
origami
HC picture
junctions



(all of these 3D images are orthographic projections)

Fig. 102. If the objects in Fig. 101 were transparent, except for the edges, none of the edges would have been hidden. Each edge is a segment of a straight line, on the boundary between two adjacent faces. Exactly three of them meet at each vertex of a 3VP.

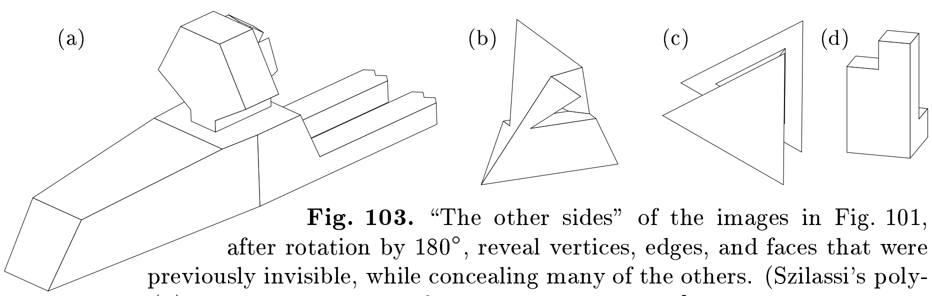


Fig. 103. "The other sides" of the images in Fig. 101, after rotation by 180°, reveal vertices, edges, and faces that were previously invisible, while concealing many of the others. (Szilassi's polyhedron (b) looks the same as before, because it has 180° rotational symmetry: The horizontal face is symmetrical, but the other three were visible only from behind.)

What are those visual cues? Working independently, D. A. Huffman and M. B. Clowes were able to decipher them successfully, in a pair of influential papers that were published at almost the same time [*Machine Intelligence* **6** (1971), 295–323; *Artificial Intelligence* **2** (1971), 79–116]. Given a 2D image that represents a 3VP X in a 3D scene, their first key idea was to classify each line segment by giving it one of four labels, according to its context:

- a *convex* edge (+), where points between the adjacent faces belong to X;
- a *concave* edge (−), where points between adjacent faces aren't part of X;
- a *half* edge (> or <), where only one of its adjacent faces can be seen.

(A half edge in the 2D image is actually a convex edge in X itself. But one of the two faces joined by this edge is invisible, because that face lies behind what we can see.) The label of a half edge is chosen so that the visible adjacent face appears to our *right* as we walk toward the point of the arrow.

For example, Fig. 104 is a marked-up version of Fig. 101, with all lines properly labeled. Convex edges are identified by tick marks, suggesting + signs. Concave edges are shown as dashed lines, like the "valley folds" in standard origami diagrams. The half edges are decorated with arrows in the proper directions. Notice that the outer boundary in each case is a polygon that consists entirely of half edges, traversed clockwise.

Let's say that an *HC picture* is a list of distinct 2D points $j = (x, y)$, called "junctions," together with lines $j \mathrel{—} j'$ between designated junctions, for which (i) every junction has degree 2 or 3; (ii) two lines intersect only at junctions;

planar
Huffman
Clowes
orthographically
general position
projection

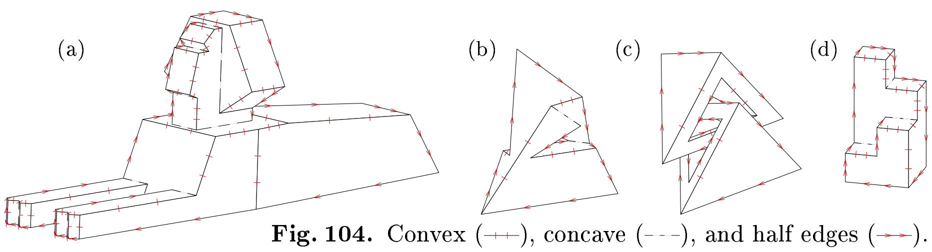


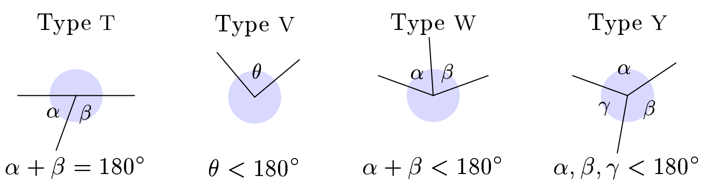
Fig. 104. Convex (—+—), concave (- - -), and half edges (—>—).

(iii) the two lines at a junction of degree 2 aren't collinear. Property (ii) means that the associated graph is planar. Property (iii) means that we can "see" all the junctions just by looking at the lines. (The "HC" in this definition stands for Huffman and Clowes.)

Given any 3VP X, suppose we project its vertices $v = (x, y, z)$ and edges v —— v' orthographically onto the (x, y) plane, eliminating hidden points and lines by assuming that (x, y, z) is in front of (x, y, z') whenever $z < z'$. We shall also assume that X is in *general position*, meaning that a slight rotation of X won't change the number of lines we see or the ways they relate to each other. (This assumption rules out exceptional cases that might occur accidentally, but with probability zero; exercise 40 has a formal definition.)

The resulting projection is always an HC picture, to which labels might be attached. For example, Figs. 101 and 103 are HC pictures, and Fig. 104 is a labeled HC picture. Every visible vertex of X appears as a junction in the HC picture. Furthermore, additional junctions are often present at the left of half edges, as artifacts of the projection process: We see them wherever an edge of X is partly hidden, but they aren't really intrinsic to X itself. (One such junction is below the middle of Fig. 104(d); Fig. 104(c) has 15 of them.)

The junctions of an HC picture can be classified into four types, based on their degrees and the angles between their neighboring lines:



(19)

(Type T junctions are the artifacts of projection, mentioned above.)

And now we get to the punch line, noticed independently by Huffman and Clowes: *When the lines of an HC picture are labeled with* + *or* − *or* > *or* <, *in order to distinguish between convex edges, concave edges, and half edges, only a small number of cases are actually possible, for each type of junction.* In fact,

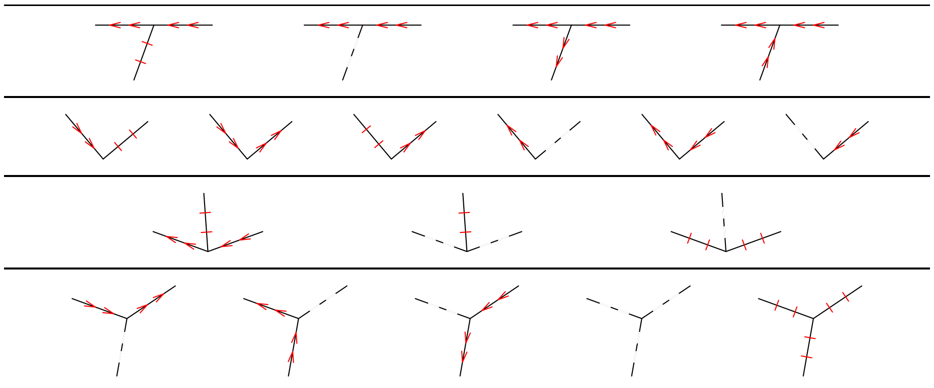
- A T junction can be labeled in only four ways (not $4^3 = 64$);
- A V junction can be labeled in only six ways (not $4^2 = 16$);
- A W junction can be labeled in only three ways (not $4^3 = 64$);
- A Y junction can be labeled in only five ways (not $4^3 = 64$).

That's part of the reason why we're able to perceive depth rather easily.

HC network
line labeling problem

Table 1

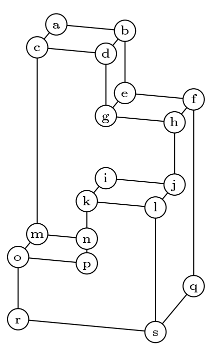
LEGAL LABELS FOR EACH JUNCTION TYPE



Exercise 41 works out the complete list of possibilities, exhibited in Table 1. And there's also more good news, a second punch line: Every line appears in two junctions, but has only one label; hence it's constrained at both ends!

Let's convert these geometric concepts to a purely combinatorial problem, by abstracting away the coordinates and considering only the underlying graph. We shall say that an *HC network* is a list of named junctions, where each junction is either $T(l,m,r)$, $V(l,r)$, $W(l,m,r)$, or $Y(a,b,c)$; here l, m, r, a, b, and c are the names of other junctions, and junction j' appears in the definition of j if and only if j appears in the definition of j'.

For example, here's the HC network that corresponds to Fig. 101(d):



$$\begin{array}{ll}
\mathrm{a}=V(\mathrm{b},\mathrm{c}); & \\
\mathrm{b}=W(\mathrm{e},\mathrm{d},\mathrm{a}); & \mathrm{k}=W(\mathrm{i},\mathrm{l},\mathrm{n});\\
\mathrm{c}=W(\mathrm{a},\mathrm{d},\mathrm{m}); & \mathrm{l}=Y(\mathrm{j},\mathrm{s},\mathrm{k});\\
\mathrm{d}=Y(\mathrm{b},\mathrm{g},\mathrm{c}); & \mathrm{m}=Y(\mathrm{c},\mathrm{n},\mathrm{o});\\
\mathrm{e}=Y(\mathrm{b},\mathrm{f},\mathrm{g}); & \mathrm{n}=T(\mathrm{k},\mathrm{m},\mathrm{p});\\
\mathrm{f}=W(\mathrm{q},\mathrm{h},\mathrm{e}); & \mathrm{o}=W(\mathrm{m},\mathrm{p},\mathrm{r});\\
\mathrm{g}=W(\mathrm{d},\mathrm{e},\mathrm{h}); & \mathrm{p}=V(\mathrm{o},\mathrm{n});\\
\mathrm{h}=Y(\mathrm{f},\mathrm{j},\mathrm{g}); & \mathrm{q}=V(\mathrm{s},\mathrm{f});\\
\mathrm{i}=V(\mathrm{j},\mathrm{k}); & \mathrm{r}=V(\mathrm{o},\mathrm{s});\\
\mathrm{j}=W(\mathrm{l},\mathrm{i},\mathrm{h}); & \mathrm{s}=W(\mathrm{r},\mathrm{l},\mathrm{q}).
\end{array} \tag{20}$$

(Every HC picture has a unique HC network, except that the parameters of Y junctions can be permuted cyclically. For example, we could have written '$\mathrm{d}=Y(\mathrm{g},\mathrm{c},\mathrm{b})$' or '$\mathrm{d}=Y(\mathrm{c},\mathrm{b},\mathrm{g})$' instead of '$\mathrm{d}=Y(\mathrm{b},\mathrm{g},\mathrm{c})$' in (20); and there also are three equivalent ways to define each of the other Y junctions $\{\mathrm{e},\mathrm{h},\mathrm{l},\mathrm{m}\}$. But '$\mathrm{d}=Y(\mathrm{b},\mathrm{c},\mathrm{g})$' would be incorrect, because it doesn't match the HC picture. The branches of a Y must be listed in clockwise order.)

Given an HC network, the *line labeling problem* is to classify each of the lines between adjacent junctions as either convex (+), concave (−), or a properly

tuples
boundary cycle
standard

oriented half edge ($\texttt{<}$ or $\texttt{>}$), in such a way that every junction conforms to one of the patterns in Table 1; a half edge ab that points from a to b is labeled $\texttt{>}$. This is, of course, a constraint satisfaction problem: The variables are the lines; the domains are the symbols $\{\texttt{+}, \texttt{-}, \texttt{<}, \texttt{>}\}$; and the constraints are given by Table 1.

For example, the line labeling problem for (20) has the 26 variables

$$\begin{aligned}&\{\mathrm{ab}, \mathrm{ac}, \mathrm{bd}, \mathrm{be}, \mathrm{cd}, \mathrm{cm}, \mathrm{dg}, \mathrm{ef}, \mathrm{eg}, \mathrm{fh}, \mathrm{fq}, \mathrm{gh}, \mathrm{hj},\\ &\qquad\qquad \mathrm{ij}, \mathrm{ik}, \mathrm{jl}, \mathrm{kl}, \mathrm{kn}, \mathrm{ls}, \mathrm{mn}, \mathrm{mo}, \mathrm{np}, \mathrm{op}, \mathrm{or}, \mathrm{qs}, \mathrm{rs}\}\end{aligned} \tag{21}$$

and 19 constraints defined by the following sets of tuples:

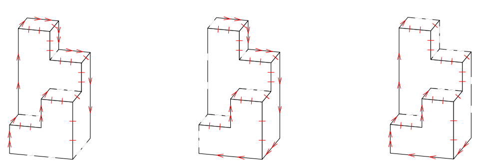
$$\begin{array}{rl@{\qquad}rl}
(\mathrm{ab}, \mathrm{ac}) \in & \{\texttt{<+}, \texttt{<>}, \texttt{+>}, \texttt{>-}, \texttt{><}, \texttt{-<}\}; & (\mathrm{ik}, \mathrm{kl}, \mathrm{kn}) \in & \{\texttt{<+<}, \texttt{-+-}, \texttt{+-+}\};\\
(\mathrm{be}, \mathrm{bd}, \mathrm{ab}) \in & \{\texttt{>+>}, \texttt{-+-}, \texttt{+-+}\}; & (\mathrm{jl}, \mathrm{ls}, \mathrm{kl}) \in & \{\texttt{<->}, \texttt{-<<}, \texttt{>>-}, \texttt{---}, \texttt{+++}\};\\
(\mathrm{ac}, \mathrm{cd}, \mathrm{cm}) \in & \{\texttt{<+<}, \texttt{-+-}, \texttt{+-+}\}; & (\mathrm{cm}, \mathrm{mn}, \mathrm{mo}) \in & \{\texttt{<-<}, \texttt{-<>}, \texttt{>>-}, \texttt{---}, \texttt{+++}\};\\
(\mathrm{bd}, \mathrm{dg}, \mathrm{cd}) \in & \{\texttt{<->}, \texttt{-<<}, \texttt{>>-}, \texttt{---}, \texttt{+++}\}; & (\mathrm{kn}, \mathrm{np}) \in & \{\texttt{<<}\};\\
(\mathrm{be}, \mathrm{ef}, \mathrm{eg}) \in & \{\texttt{<-<}, \texttt{-<>}, \texttt{>>-}, \texttt{---}, \texttt{+++}\}; & (\mathrm{mo}, \mathrm{op}, \mathrm{or}) \in & \{\texttt{<+<}, \texttt{-+-}, \texttt{+-+}\};\\
(\mathrm{fq}, \mathrm{fh}, \mathrm{ef}) \in & \{\texttt{>+>}, \texttt{-+-}, \texttt{+-+}\}; & (\mathrm{op}, \mathrm{np}) \in & \{\texttt{>+}, \texttt{><}, \texttt{+<}, \texttt{<-}, \texttt{<>}, \texttt{->}\};\\
(\mathrm{dg}, \mathrm{eg}, \mathrm{gh}) \in & \{\texttt{<+<}, \texttt{-+-}, \texttt{+-+}\}; & (\mathrm{qs}, \mathrm{fq}) \in & \{\texttt{<+}, \texttt{<<}, \texttt{+<}, \texttt{>-}, \texttt{>>}, \texttt{->}\};\\
(\mathrm{fh}, \mathrm{hj}, \mathrm{gh}) \in & \{\texttt{<->}, \texttt{-<<}, \texttt{>>-}, \texttt{---}, \texttt{+++}\}; & (\mathrm{or}, \mathrm{rs}) \in & \{\texttt{>+}, \texttt{>>}, \texttt{+>}, \texttt{<-}, \texttt{<<}, \texttt{-<}\};\\
(\mathrm{ij}, \mathrm{ik}) \in & \{\texttt{<+}, \texttt{<>}, \texttt{+>}, \texttt{>-}, \texttt{><}, \texttt{-<}\}; & (\mathrm{rs}, \mathrm{ls}, \mathrm{qs}) \in & \{\texttt{<+>}, \texttt{-+-}, \texttt{+-+}\}.\\
(\mathrm{jl}, \mathrm{ij}, \mathrm{hj}) \in & \{\texttt{>+>}, \texttt{-+-}, \texttt{+-+}\}; & &
\end{array} \tag{22}$$

(Here ‘$\texttt{<+}$’ stands for the ordered pair $(\texttt{<}, \texttt{+})$; ‘$\texttt{>+>}$’ stands for $(\texttt{>}, \texttt{+}, \texttt{>})$; and so on.)

Notice that the constraint for junction b was not written ‘$(\mathrm{be}, \mathrm{bd}, \mathrm{ba}) \in \{\texttt{>+<}, \texttt{-+-}, \texttt{+-+}\}$’, because ‘ba’ isn’t one of the variables: The line between junctions b and a is represented by ‘ab’ in (21). We could have had 52 variables $\{\mathrm{ab}, \mathrm{ac}, \mathrm{ba}, \mathrm{bc}, \ldots, \mathrm{sr}\}$ instead of 26, by introducing 26 further constraints such as $(\mathrm{ab}, \mathrm{ba}) \in \{\texttt{++}, \texttt{--}, \texttt{<>}, \texttt{><}\}$. But that would have wasted time and space.

Notice also that the constraint for junction n was not written ‘$(\mathrm{kn}, \mathrm{mn}, \mathrm{np}) \in \{\texttt{<+<}, \texttt{<-<}, \texttt{<<<}, \texttt{<><}\}$’. The simpler and more direct statement in (22) is more efficient, and in fact it’s the best way to understand the top row of Table 1.

The CSP in (22) is readily expressed as an XCC problem (see exercise 44), and it turns out to have just four solutions. The labeled picture in Fig. 104(d) represents the histoscape “floating in air”; the other three solutions



(23)

represent it “attached to the ground,” or “attached to a wall” at the left or back.

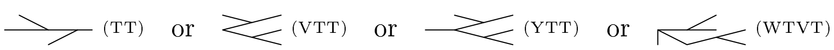
Every connected HC picture has a unique *boundary cycle*, consisting of the junctions that touch the “outside” region, in clockwise order. For example, the boundary cycle of (20) is (abefqsromc). A line labeling is called *standard* if every line between consecutive junctions of the boundary cycle has been labeled as a half edge pointing clockwise. That makes sense, because it means that the object lies entirely inside the boundary — unattached to any unbounded background environment. All four of the labelings in Fig. 104 are standard.

The sphinx of Fig. 101(a) has only two standard labelings, in spite of its numerous junctions and lines. The other possibility, besides Fig. 104(a), simply changes two of the labels so that the head isn't necessarily attached to the body.

The Szilassi polyhedron, Fig. 101(b), likewise has exactly two standard labelings. (See exercise 45.) But Fig. 101(c) is far more ambiguous: It has 256 standard labelings. Indeed, three of its lines are completely unconstrained, because they're the stems between two T junctions.

A surprising thing happens when we ask for *all* valid labelings of Fig. 101, standard or not: The possibilities for the interior lines — the lines *not* between adjacent junctions of the boundary cycle — remain the same! More precisely, the number of ways to satisfy the constraints only at the boundary junctions turns out to be (720, 3, 6, 4), for Figs. 101(a), (b), (c), (d), respectively, while the total number of valid labelings is $(720\cdot 2, 3\cdot 2, 6\cdot 256, 4\cdot 1)$. In other words, all of the consistent boundary labelings are mutually interchangeable; hence the boundary can essentially be "factored out." When this happens we say that the HC picture has a *free boundary.* Not every picture has a free boundary, but exceptions seem to be rare in practice. Exercises 49–56 explore this curious phenomenon.

It's not difficult to construct HC pictures that *cannot* be labeled. For example, any picture that contains a subpicture of the forms



(24)

will fail because each T junction forces two labels. Other impossible subpictures

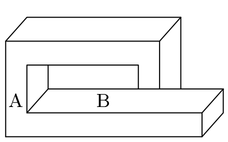
(25)

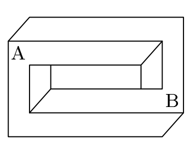
involve only one T; and exercise 57 has a small T-less example. The Swedish artist Oscar Reutersvärd has devised many amusing unlabelable pictures such as


(26)

that fool our eyes when plausible side patterns are contradictory in the middle.

On the other hand, some HC pictures can be labeled perfectly, yet they don't correspond to any actual 3VP. Consider, for example, the pictures



or


, (27)

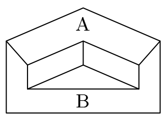
which look locally right although they're globally wrong. They "fail to compute" because each of them has two plane regions ('A' and 'B') that intersect in two *different* lines, contradicting a well-known principle of geometry.

A somewhat subtle distinction arises here, noted by Huffman in his original paper of 1971: There are locally consistent pictures that are globally inconsistent

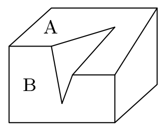
sphinx
Szilassi polyhedron
free boundary
Reutersvärd
intersection of planes+
Huffman

strongly realizable
weakly realizable
impossible
Huffman
truncated tetrahedron
rounding error
Penrose triangle
Penrose
Penrose

by virtue of the two-planes-determine-one-line principle, such as

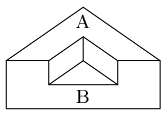


and

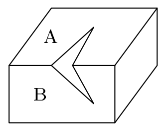


; (28)

yet certain globally *consistent* pictures have exactly the same HC networks:



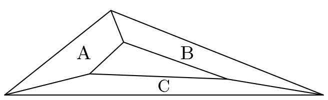
and



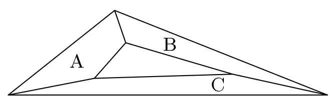
. (29)

Let's say that an HC picture H is *strongly realizable* if H is the projection of at least one 3VP X in general position. It is *weakly realizable* if there's an HC picture H' with the same HC network as H for which H' is strongly realizable. It is *impossible* if it's not weakly realizable. Thus, the pictures in (29) are strongly realizable; the pictures in (28) are weakly realizable; the picture in (26) is impossible. (The picture in (26) is not only impossible, it can't even be labeled.)

Huffman observed that a truncated tetrahedron gives another instructive example: Consider

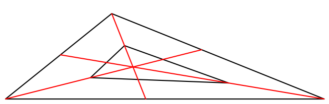


versus

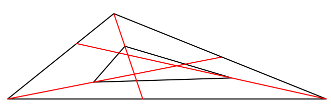


. (30)

The left picture is strongly realizable, but the right picture is not! In this case three planes are involved ('A', 'B', 'C'); three of the lines show the intersections of planes AB, BC, and CA. Those three planes always intersect in a single point, ABC, because no two of them are parallel. The relevant lines at the left of (30) do indeed share an invisible common point; but the lines at the right do not:



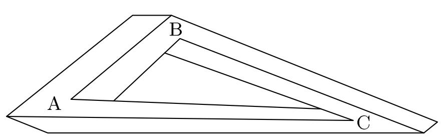
versus



.

Thus we see that the notion of strong realizability is quite delicate — not at all robust: A tiny rounding error in one of the (x, y) coordinates can change a strongly realizable picture into one that can be realized only weakly.

The most famous impossible HC picture is probably the "Penrose triangle"

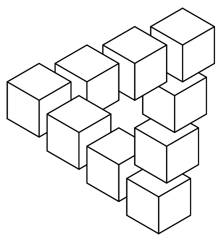


(31)

introduced by L. S. Penrose and R. Penrose in the *British Journal of Psychology* **49** (1958), 31–33. (Their version was slightly different: It was equilateral, and it included a few spurious "crack" lines.) Huffman's argument about nonconcurrent lines AB, BC, CA proves that (31) isn't even weakly realizable; and exercise 58 gives another proof of impossibility.

Reutersvärd
fun
HUFFMAN
Graph labeling
graceful labeling
complement

Oscar Reutersvärd, who is now known as the "father" of impossible pictures, discovered a paradoxical pattern akin to the Penrose triangle already in 1934:



(32)

This HC picture appears to be made of nine separate boxes that overlap in an impossible fashion. Surprisingly, however, it actually turns out to be strongly realizable! (See exercise 59.)

In fact, the theory of realizable objects is still far from complete, even when restricted to the 3VP world, and many fascinating problems remain to be solved.

> *I plead guilty to the charge that I deal with pictures of impossible objects because it is fun. It is, and that is reason enough. However, in addition to this I believe that much can be learned in the study of any language by asking* 'Is *that a nonsense sentence?' and* 'Why *is that a nonsense sentence?'.*
>
> — D. A. HUFFMAN (1971)

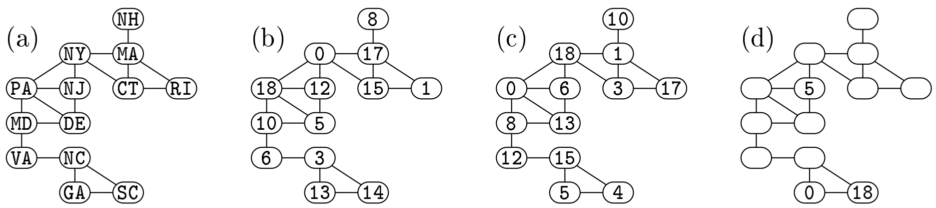
Graph labeling. Let's turn now to a completely different but equally fascinating way to attach labels to the vertices and edges of a graph. Our new goal is to give an identifying number to each vertex while simultaneously identifying each edge.

Consider, for example, Fig. 105(a), which is a graph of the 13 colonies that combined to form the original United States of America in 1776. Two vertices are adjacent if the corresponding colonies have a common boundary. Figure 105(b) shows that each colony can be represented by a cleverly chosen number, so that every edge is identified uniquely by the difference between the numbers of its endpoints:

$$\begin{array}{rrrrrr} 14-13=1 & 10-6=4 & 12-5=7 & 13-3=10 & 18-5=13 & 17-1=16 \\ 17-15=2 & 10-5=5 & 18-10=8 & 14-3=11 & 15-1=14 & 17-0=17\,. \\ 6-3=3 & 18-12=6 & 17-8=9 & 12-0=12 & 15-0=15 & 18-0=18 \end{array} \qquad (33)$$

Numberings with this property are called "graceful." Formally speaking, if G is a graph with m edges, a *graceful labeling* of G is a function that assigns an integer $l(v)$ to each vertex v, in the range $0 \le l(v) \le m$, with the property that no two vertices have the same value of $l(v)$, and no two edges have the same value of $|l(v) - l(w)|$. We say that $l(v)$ is the label of vertex v, and $|l(v) - l(w)|$ is the label of edge $v \mathbin{—} w$. Notice that $|l(v) - l(w)|$ is always positive, and it's at most $|m-0| = m$; therefore there's exactly one edge labeled d, for each d in $\{1, \ldots, m\}$.

Every graceful labeling has a "complement," obtained by setting $l(v) \leftarrow m - l(v)$ for all v. (See Fig. 105(c).) Complementation doesn't change the label of any edge. A labeling and its complement are considered to be essentially identical.



symmetry
automorphism
permutation
13-colonies graph
essentially different
puzzle
sudoku
data structure
isolated vertices

Fig. 105. (a) A famous graph G, which has 13 vertices and 18 edges. (b) One of the many graceful labelings of G. (c) The same labeling as (b), but complemented. (d) A puzzle: Complete this labeling to make it graceful. (The solution is unique.)

Every symmetry of a graph also preserves gracefulness. In other words, if α is an automorphism (a permutation of the vertices for which $v \longrightarrow w$ implies $v\alpha \longrightarrow w\alpha$), and if l is a graceful labeling, then the labeling $l'(v) = l(v\alpha)$ is also graceful. For example, Fig. 105(a) is symmetrical if we swap GA $\leftrightarrow$ SC; hence we could also swap the labels 13 $\leftrightarrow$ 14 in Fig. 105(b) and/or the labels 5 $\leftrightarrow$ 4 in Fig. 105(c). In this way every graceful labeling of the 13-colonies graph yields a set of four labelings that are mutually equivalent. (See exercise 68.)

That graph actually has hundreds of thousands of graceful labelings: 641952 altogether! Dividing by 4 gives us 160488 that are essentially different. They can be found quickly, using for example the XCC model of exercise 69. Each of the 18 edges can be the "longest," namely the edge that's labeled 18. That edge connects NY to PA, as it does in Fig. 105(b, c), in 22782 of those 160488 solutions; and it connects NY to MA in even more of them (24896). On the other hand only 24 of the 160488 have the longest edge between GA and SC, as in Fig. 105(d). (The latter labeling has been left as a puzzle; it's roughly as difficult as a "hard" sudoku.)

A nice data structure can be used to represent a gracefully labeled graph inside a computer, using a few arrays of size $m + 1$. First, by including isolated vertices if necessary, we can assume that the vertices are named $0, 1, \ldots, m$, and that $l(v) = v$ for $0 \le v \le m$. (In other words, a vertex's label is also its name.) Then, if edge d connects vertices v and $v + d$, we set $\mathtt{LO}[d] \leftarrow v$. Consequently *two arbitrary vertices v and w with $v < w$ are adjacent if and only if* $\mathtt{LO}[w - v] = v$. With three further arrays, FIRST, NEXTL, and NEXTH, we can also visit all neighbors w of any given vertex v using a simple loop:

$$\text{Set } w \leftarrow \mathtt{FIRST}[v]. \text{ While } w \ge 0, \text{ set } w \leftarrow \begin{cases} \mathtt{NEXTL}[v - w], & \text{if } w < v; \\ \mathtt{NEXTH}[w - v], & \text{if } w > v. \end{cases} \qquad (34)$$

For example, the arrays might look like this in the case of Fig. 105(b):

$l =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LO[l] $=$	—	13	15	3	6	5	12	5	10	8	3	3	0	5	1	0	1	0	0
FIRST[l] $=$	12	15	−1	6	−1	10	3	−1	17	−1	6	−1	18	14	13	17	−1	15	12
NEXTL[l] $=$	—	3	8	10	5	18	10	0	5	1	−1	−1	−1	0	0	−1	0	−1	−1
NEXTH[l] $=$	—	3	1	13	−1	12	5	18	−1	−1	14	−1	15	−1	17	17	−1	18	−1
NAME[l] $=$	NY	RI	—	NC	—	DE	VA	—	NH	—	MD	—	NJ	GA	SC	CT	—	MA	PA

(35)

(The NAME array shown here gives an optional *external* name for printouts. Entries marked '—' in these example arrays are unused.)

When a graph has at least one graceful labeling it's called "graceful"; in that sense, the 13-colonies graph can be considered quite graceful indeed. Not all graphs are graceful, of course. For example, a disconnected graph with more than $m+1$ vertices can't possibly be graceful; there aren't enough labels to go around. And one can easily check that C_5, the 5-cycle, has no graceful labeling.

C_5
contiguous USA graph
USA graph
randomized algorithm
miracle
π
Rokicki

The 13-colonies graph is merely an induced subgraph of a much larger graph, which we've been exploring in lots of examples in previous sections. The contiguous USA graph, introduced in 7–(17) and last seen in exercises 7.2.2.2–35 and 37, has 49 vertices and 107 edges. Could that graph possibly be graceful?

The answer is yes; and exercise 102 discusses a randomized algorithm that is able to label it gracefully without a great deal of work. In fact, an inspired use of that algorithm has revealed what can only be described as a *graceful miracle*: A solution can actually be achieved by stipulating that the 15 states on the western and northern borders, from California to Maine, should be labeled respectively with the numbers 31, 41, 59, ..., 83, 27 — the first 30 digits of π!! This has to be seen to be believed (see Fig. 106).

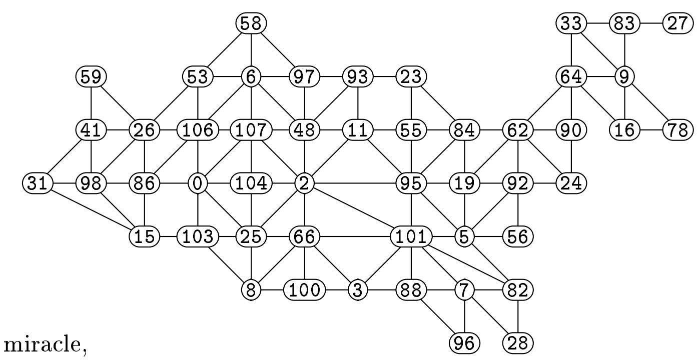


Fig. 106. A graceful miracle, found by Tomas G. Rokicki in October 2020.

	$98-86=12$	$33-9=24$	$92-56=36$	$64-16=48$	$86-26=60$	$98-26=72$	$95-11=84$	$101-5=96$
$107-106=1$	$101-88=13$	$25-0=25$	$48-11=37$	$97-48=49$	$84-23=61$	$92-19=73$	$88-3=85$	$100-3=97$
$64-62=2$	$19-5=14$	$90-64=26$	$62-24=38$	$83-33=50$	$78-16=62$	$83-9=74$	$86-0=86$	$101-3=98$
$107-104=3$	$41-26=15$	$53-26=27$	$97-58=39$	$56-5=51$	$66-3=63$	$82-7=75$	$92-5=87$	$101-2=99$
$97-93=4$	$31-15=16$	$90-62=28$	$95-55=40$	$58-6=52$	$66-2=64$	$95-19=76$	$103-15=88$	$106-6=100$
$58-53=5$	$25-8=17$	$84-55=29$	$66-25=41$	$106-53=53$	$84-19=65$	$82-5=77$	$96-7=89$	$107-6=101$
$101-95=6$	$59-41=18$	$92-62=30$	$48-6=42$	$82-28=54$	$90-24=66$	$103-25=78$	$95-5=90$	$104-2=102$
$16-9=7$	$101-82=19$	$64-33=31$	$62-19=43$	$64-9=55$	$98-31=67$	$104-25=79$	$97-6=91$	$103-0=103$
$96-88=8$	$106-86=20$	$55-23=32$	$55-11=44$	$83-27=56$	$92-24=68$	$106-26=80$	$100-8=92$	$104-0=104$
$11-2=9$	$28-7=21$	$59-26=33$	$93-48=45$	$98-41=57$	$78-9=69$	$88-7=81$	$95-2=93$	$107-2=105$
$41-31=10$	$84-62=22$	$100-66=34$	$48-2=46$	$66-8=58$	$93-23=70$	$93-11=82$	$101-7=94$	$106-0=106$
$95-84=11$	$25-2=23$	$101-66=35$	$53-6=47$	$107-48=59$	$86-15=71$	$98-15=83$	$103-8=95$	$107-0=107$

The problem of labeling a given graph G of size m gracefully can be formalized as a CSP in many ways. For example, we can render the definition directly, by saying that the variables of the CSP are the vertices and edges of G; the domain of each vertex is $\{0,\ldots,m\}$ and the domain of each edge is $\{1,\ldots,m\}$; the constraints are that $l(e)=|l(v)-l(w)|$ when e is the edge $v \longrightarrow w$; furthermore the vertex labels should all be different and the edge labels should all be different.

That direct model lets us solve small problems, of course. But experience shows that it doesn't scale up well. A much better method can be based on the LO and NAME arrays of the data structure in (35), where we take the attitude that vertex and edge labels are already given; our job is to attach them to the graph! More precisely, there's a variable for each vertex label in $\{0,\ldots,m\}$; and they all have the domain $V \cup \{?\}$, meaning that each label l should be assigned a NAME[l], which is either a vertex of G or undefined. The defined labels should all be different. Furthermore, there's a variable for each edge label in $\{1,\ldots,m\}$; and its value LO[l] has the domain $\{0,\ldots,m-l\}$. The constraint is that

$$\texttt{NAME[LO[}l\texttt{]]} \mathbin{—\!\!—} \texttt{NAME[LO[}l\texttt{]}+l\texttt{]} \quad \text{is an edge of } G, \qquad \text{for } 1 \le l \le m. \tag{36}$$

Let's call this the "reverse model."

The reverse model has a big advantage, because LO[l] has a very small domain when l is large. Indeed, LO[m] must be 0; and LO[$m-1$] must be either 0 or 1. We can in fact assume that LO[$m-1$] $= 0$, because complementation changes LO[$m-1$] to $1-$ LO[$m-1$]. (See exercise 70.)

For example, the reverse model makes it easy to discover all of the graceful labelings when G is the complete graph K_n. In this case there are $m = \binom{n}{2}$ edges; and the constraint (36) is satisfied if and only if NAME[LO[l]] and NAME[LO[l]$+l$] are both defined, meaning that LO[l] and LO[l]$+l$ are both among the n "real" vertices that belong to K_n.

If $n = 1$, we're done: K_1 is graceful, with vertex 0.

Otherwise $m > 0$ and LO[m] $= 0$. Hence 0 and m are real vertices, and we're done if $n = 2$.

Otherwise $m > 1$, and we may assume that LO[$m-1$] $= 0$ as stated above. That means $m-1$ is also real. So if $n = 3$, we know that the three real vertices are $\{0,2,3\}$; hence LO[2] $= 0$ and LO[1] $= 2$. That settles K_3.

If $m > 2$, edge $m-2$ is always either $0 \mathbin{—\!\!—} (m-2)$ or $1 \mathbin{—\!\!—} (m-1)$ or $2 \mathbin{—\!\!—} m$, and each case gives us a new real vertex. Consequently the four vertices when $n = 4$ are either $\{0,4,5,6\}$, $\{0,1,5,6\}$, or $\{0,2,5,6\}$. Only the third alternative allows us to define LO[3] without introducing a fifth real vertex. That settles K_4.

Finally, if $n > 4$, we get stuck (see exercise 71). So we've discovered that *K_n has a unique graceful labeling when $n \le 4$, but K_n is ungraceful when $n \ge 5$.*

The star graph $K_{1,n}$ is another instructive example. It consists of a central vertex that's joined to each of n other vertices; so it has lots of symmetry, like K_n, but it has only $m = n$ edges.

We might as well assume that $n > 1$, because $K_{1,1} = K_2$. So we know that LO[n] $= 0$, and also LO[$n-1$] $= 0$. But that means 0 must be the central vertex, because no other vertex has more than one neighbor. Consequently LO[$n-2$] $= 0$, LO[$n-3$] $= 0$, and so on; *$K_{1,n}$ has a unique graceful labeling.*

That was easy. But what happens if G is the path graph P_n? A graceful labeling of P_n is called a *graceful permutation*, because P_n has $m = n-1$ edges, and the sequence $p_0p_1\ldots p_{n-1}$ of labels on the path is a permutation of $\{0,1,\ldots,n-1\}$. The permutation $p_0p_1\ldots p_{n-1}$ is graceful if and only if

$$|p_0-p_1|\,|p_1-p_2|\,\ldots\,|p_{n-2}-p_{n-1}| \ \text{ is a permutation of } \{1,\ldots,n-1\}. \tag{37}$$

domain
reverse model
complementation
unique graceful labeling
star graph
unique graceful labeling
path graph
graceful permutation

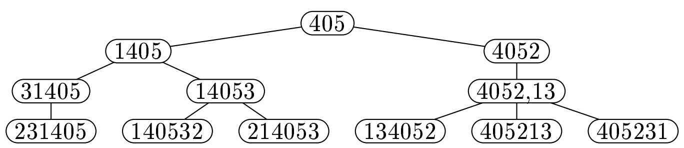
We know that P_3 has a unique graceful labeling, because $P_3 = K_{1,2}$. That fact can be confusing, because the six permutations $p_0p_1p_2$ of $\{0,1,2\}$ are

$$012,\ 021,\ 102,\ 120,\ 201,\ 210$$

and *four* of them satisfy (37)! Everything becomes clear, however, once we realize that the permutations $p_0p_1\ldots p_{n-1}$, $p_{n-1}\ldots p_1p_0$, $(n{-}1{-}p_0)\ldots(n{-}1{-}p_{n-1})$, and $(n{-}1{-}p_{n-1})\ldots(n{-}1{-}p_0)$ are considered to be essentially the *same*, because each of them is obtainable from the others by reversal and complementation. Similarly, the graceful labelings of P_4 and P_5 reduce to 1203 and either 21304 or 30421, which each represent four permutations. There are four times as many graceful permutations as there are ways to label P_n gracefully, when $n > 2$.

unique graceful labeling
reversal
complementation
data structure
ZDD
KP graphs
cliques

Let's take a look at P_6. We can assume that edges 5 and 4 will be $0 \text{---} 5$ and $0 \text{---} 4$, which we can abbreviate to 05 and 04, respectively. Thus $p_0p_1\ldots p_5$ will contain the substring 405 or 504, and we can assume that it's 405. Edge 3 must be 03 or 14 or 25; but 03 is impossible because 0 already has two neighbors. Two cases remain, 1405 and 4052. The tree of possibilities is, in fact,



(38)

as we choose edge 3, edge 2, then edge 1, leading to six solutions altogether.

Notice that this procedure chooses the values of `LO[5]`, `LO[4]`, `LO[3]`, ... sequentially. But it does *not* choose *any* values for the `NAME` array until the very last step. For instance, at one point in (38) we know that 4052 and 13, or their reflections, should be substrings of the final permutation; but we don't commit ourselves prematurely to exactly where those substrings will appear. Exercise 72 discusses a convenient data structure for dealing with such partial permutations.

The number of graceful permutations grows exponentially with n. For example, P_{41} can be labeled gracefully in 258,002,411,935,989,500 ways! Exercise 73 explains how a ZDD with fewer than 25 million nodes can represent them all.

Some dazzling patterns arise when we consider "KP graphs" of the form $K_n \,\square\, P_r$, which consist of $r > 1$ cliques in a row, each of size $n > 2$. For example, here are two of the many graceful labelings of $K_4 \,\square\, P_{10}$ and $K_5 \,\square\, P_7$:

$$\begin{pmatrix} 0 & 96 & 4 & 93 & 5 & 90 & 11 & 88 & 22 & 84 \\ 1 & 3 & 13 & 65 & 89 & 14 & 62 & 25 & 81 & 58 \\ 91 & 9 & 87 & 7 & 77 & 50 & 18 & 72 & 51 & 69 \\ 95 & 28 & 73 & 12 & 55 & 17 & 82 & 33 & 68 & 27 \end{pmatrix}; \quad \begin{pmatrix} 10 & 56 & 99 & 0 & 100 & 13 & 93 \\ 33 & 66 & 7 & 77 & 12 & 87 & 59 \\ 81 & 95 & 1 & 41 & 3 & 94 & 8 \\ 86 & 2 & 97 & 15 & 70 & 26 & 71 \\ 89 & 6 & 79 & 52 & 69 & 45 & 24 \end{pmatrix}. \qquad (39)$$

Each of the 10 columns on the left has six differences; in the first column they are $\{|0-1|, |0-91|, |0-95|, |1-91|, |1-95|, |91-95|\} = \{1, 91, 95, 90, 94, 4\}$. And each row also has nine differences between adjacent columns; in the first row they are $\{|0-96|, |96-4|, \ldots, |22-84|\} = \{96, 92, 89, 88, 85, 79, 77, 66, 62\}$. Those $60+36$ differences are all distinct! And so are the $70+30$ differences on the right!!

symmetries
automorphisms
complementation
break this symmetry
canonical form

In general $K_n \,\square\, P_r$ has rn vertices and $m = r\binom{n}{2} + (r-1)n$ edges; that's exactly n^2 edges when $r = 2$. It has $2n!$ symmetries (aka automorphisms), because we can permute the rows of the matrix and/or reflect it left↔right.

Every graceful labeling of a KP graph can be represented as an $n \times r$ matrix (x_{ij}), for $1 \le i \le n$ and $1 \le j \le r$, as in (39). When complementation is taken into account, every suitable matrix is therefore equivalent to a set of $4n!$ such solutions. The usual way to break this symmetry, in order to generate only inequivalent solutions, is to add additional constraints so that the matrix is in a "canonical form." For example, we can insist as above that 0 is adjacent to $m-1$, or that 0 and $m-1$ occupy the same column, and also that

$$x_{11} < x_{21} < \cdots < x_{n1}; \quad x_{11} < x_{1r}. \tag{40}$$

(See exercise 76.) The matrices in (39) are canonical in this sense. Constraints like (40), which significantly prune the search tree, are supposedly helpful.

But in this case a far more efficient approach is possible, based on the label-oriented philosophy suggested by the reverse model and exemplified by the way we've already handled K_n and P_r. Figure 107 illustrates the smallest KP graph:

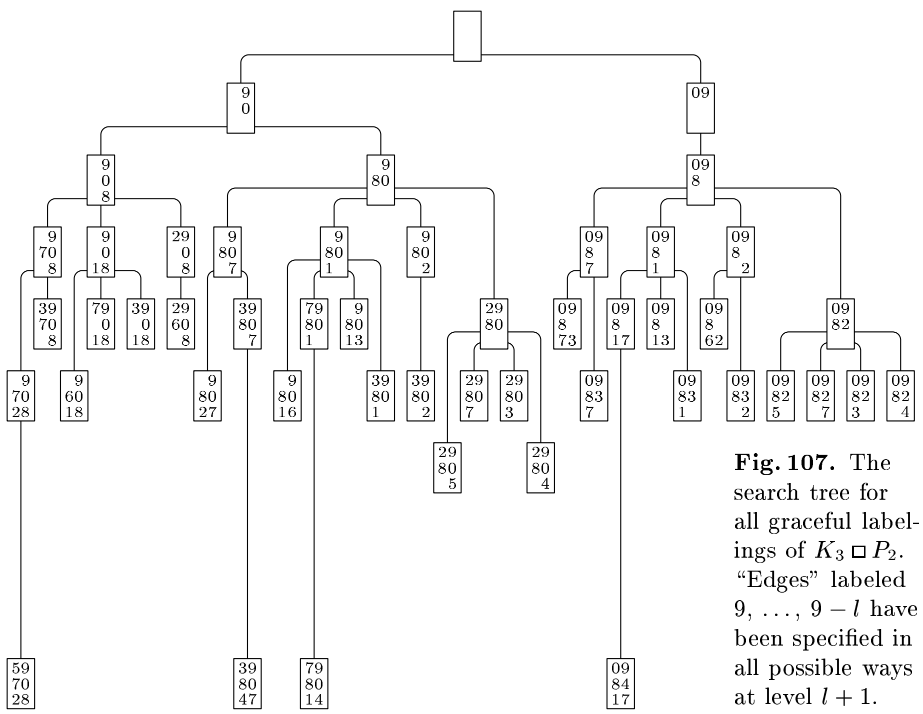


Fig. 107. The search tree for all graceful labelings of $K_3 \,\square\, P_2$. "Edges" labeled $9, \ldots, 9-l$ have been specified in all possible ways at level $l+1$.

This problem has four solutions, which appear at the bottom of the tree (level 9). The key idea here is that we construct a "home-grown" canonical representation on the fly, by filling the 3×2 matrix with the labels of vertices that we've chosen to be the endpoints of edges m, $m-1$, $m-2$, $\ldots$. Sometimes the placement of a single new vertex will create more than one necessary edge (see exercise 77).

proof, computer-generated
unique graceful labeling
KC graphs
wraparound edges
parity
Bosák
cycle graphs
C_5
C_6

Search trees analogous to Fig. 107 can be constructed for all $n > 2$, and it turns out that the trees for $n = 3, 4, 5, \ldots$ have respectively 49, 446, 2094, 5545, 8103, 8825, 8907, 8910, 8910, 8910, 8910, … nodes. Also, the number of solutions for those n turns out to be respectively 4, 15, 1, 0, 0, 0, 0, 0, 0, 0, 0, … .

Hmmm — guess what? The algorithm runs through precisely the *same* calculations for all $n \geq 10$, except that the number m of edges keeps getting larger and larger. It never is able to get past row 10 of its partially filled matrix. This amounts to a computer-generated proof that *the graphs $K_n \,\square\, P_2$ are ungraceful for all $n > 5$.* (See exercise 79.) Furthermore, the maximum running time over all n, which is also the time needed to generate that proof, is only 1.6 megamems.

Of course the graphs $K_n \,\square\, P_3$ can be analyzed too, by filling $n \times 3$ matrices in a similar way. The calculations are harder, yet the running time is still quite reasonable: Only (700 Kμ, 80 Mμ, 3.6 Gμ, 60 Gμ, 360 Gμ) are needed for $n = (3, 4, 5, 6, 7)$ to show that they have respectively (284, 704, 101, 1, 0) graceful labelings. Furthermore, 1.9 Tμ suffice to prove that *$K_n \,\square\, P_3$ is ungraceful for all $n > 6$*, by constructing a tree of 5,463,149,994 nodes.

Fig. 108. Some graceful gems: The unique labelings of $K_5 \,\square\, P_2$ and $K_6 \,\square\, P_3$. Also a (less rare) $K_6 \,\square\, P_4$ and $K_5 \,\square\, C_5$.

$$\begin{pmatrix} 0 & 24 \\ 6 & 22 \\ 7 & 19 \\ 21 & 11 \\ 25 & 2 \end{pmatrix} \quad \begin{pmatrix} 0 & 56 & 1 \\ 5 & 36 & 9 \\ 12 & 6 & 52 \\ 33 & 55 & 26 \\ 44 & 2 & 49 \\ 57 & 20 & 11 \end{pmatrix} \quad \begin{pmatrix} 0 & 78 & 4 & 76 \\ 16 & 37 & 67 & 25 \\ 40 & 69 & 17 & 53 \\ 62 & 3 & 72 & 70 \\ 73 & 2 & 60 & 6 \\ 77 & 51 & 7 & 45 \end{pmatrix} \quad \begin{pmatrix} 0 & 62 & 6 & 64 & 75 \\ 3 & 18 & 69 & 10 & 33 \\ 41 & 70 & 23 & 59 & 20 \\ 73 & 9 & 43 & 24 & 51 \\ 74 & 2 & 71 & 14 & 8 \end{pmatrix}$$

There's another intriguing family of graphs, the "KC graphs" $K_n \,\square\, C_r$ for $n > 2$ and $r > 2$, which add wraparound edges to the KP graphs. These graphs have even more symmetry: Every vertex has degree $n+1$, so there are rn vertices and $m = r(n+1)n/2$ edges. An example appears at the right of Fig. 108, where one can check that the 50 column differences $|x_{ij} - x_{kj}|$ together with the 25 row differences $|x_{ij} - x_{i((j-1) \bmod r)}|$ are precisely $\{1, 2, \ldots, 75\}$.

A new phenomenon now appears. Experiments show that $K_3 \,\square\, C_r$ is *ungraceful* whenever r is odd; yet the number of graceful labelings for the even values $r = 4, 6, \ldots$ grows very rapidly: 3809, 41928684, … . There's a very simple mathematical reason for failure in the odd-r case:

Lemma O. *In any graceful labeling of a graph with $4k+1$ or $4k+2$ edges, the number of vertices with an odd degree and an odd label is always odd.*

Proof. We have $\sum_{u \text{---} v} \bigl|l(u) - l(v)\bigr| = 1 + 2 + \cdots + m = \binom{m+1}{2}$ when there are m edges; and a given vertex v appears exactly $\deg(v)$ times in this sum. Working modulo 2, we also have $\bigl|l(u) - l(v)\bigr| \equiv l(u) + l(v)$. Therefore $\sum_v \deg(v) l(v) \equiv \binom{m+1}{2}$. But $\binom{m+1}{2} \equiv 1$ when $m = 4k+1$ or $m = 4k+2$. ▌

Corollary E (J. Bosák). *If all vertices of a graceful graph have even degree, the graph has $4k$ or $4k+3$ edges for some integer k.* ▌

In particular, $K_3 \,\square\, C_r$ is ungraceful when r is odd, because it has $6r$ edges. Furthermore, the simple cycle graphs C_5, C_6, C_9, C_{10}, C_{13}, … can't be graceful.

The reverse model tells us another basic fact about gracefulness in general:

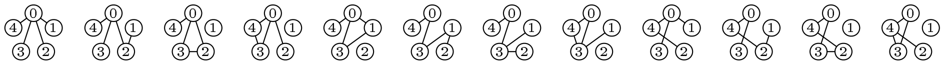
Theorem S. *There are exactly $m!$ graceful labelings with m edges.*

Proof. There are exactly $k+1$ ways to make $0 \le \texttt{LO}[m-k] \le k$, for $0 \le k < m$. ▮

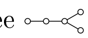
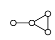
More precisely, if we insist that $\texttt{LO}[m-1] = 0$ in order to rule out complementary solutions, *there are exactly $m!/2$ essentially distinct graceful labelings with m edges, for all $m \ge 2$.* [D. A. Sheppard, *Discr. Math.* **15** (1976), 379–388.]

Here, for example, are the $4!/2 = 12$ labelings when $m = 4$:

0000 0001 0002 0003 0010 0011 0012 0013 0020 0021 0022 0023



Each instance is accompanied by its four-digit LO string, LO[4]LO[3]LO[2]LO[1]. There are $m+1$ vertices in general, namely $\{0, 1, \ldots, m\}$; but some of them may be isolated — not participating in any edge. We can think of each isolated vertex in two ways: It's either present in the graph, representing its label; or it's absent, representing an unused label.

One of the nice things about this listing of $m!/2$ labelings is that symmetry is automatically handled as it should be. A highly symmetrical graph will appear only as often as it has truly distinct labelings, because labelings that differ only because of an automorphism are seen just once. For example, we observed earlier that $K_{1,4}$ has a unique graceful labeling, while P_5 has two; sure enough, we obtain $K_{1,4}$ only in case 0000, but P_5 in cases 0011 and 0021. Notice that C_4 also has a unique labeling (case 0022). The tree , which is often called the "fork," has three distinct labelings (cases 0001, 0012, 0020). The "paw" , otherwise known as $K_1 — (K_1 \oplus K_2)$, has the most (cases 0002, 0003, 0010, 0013, 0023).

We can see gracefulness in action by looking at all $m!/2$ cases, when m isn't too large, and we're immediately faced with a host of interesting unsolved questions: How many of those cases yield graphs that are connected? planar? bipartite? triangle-free? When we omit the isolated vertices, how many of the resulting graphs are connected? cubic? And so on. (See exercises 91–97.)

In particular, how many of those graceful labelings yield a *free tree* on the vertices $\{0, 1, \ldots, m\}$? Equivalently, how many of those $m!/2$ sets of m edges have no cycles? In such cases no vertex is isolated. (See Theorem 2.3.4.1A.) The free trees shown above when $m = 4$ are 0000, 0001, 0011, 0012, 0020, and 0021.

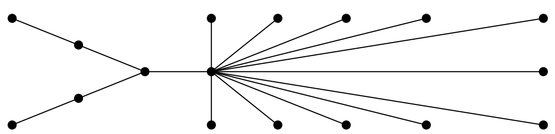
Experimentation now reveals a striking phenomenon: *The number of graceful labelings of free trees grows superexponentially, as m increases, while the number of free trees grows only exponentially.* (There are nice ways to compute both numbers, without explicitly generating labelings or trees; see exercise 105 and 2.3.4.4–(9). Furthermore, according to R. Otter in *Annals of Mathematics* (2) **49** (1948), 583–599, the number of free trees with n vertices is proportional to $\alpha^n/n^{5/2}$, where $\alpha \approx 2.955765$.) For example, when $m = 30$, there are 902,745,276,529,593,126,158,482,120 essentially different labelings, but only 40,330,829,030 free trees with 31 vertices. That's an average of more than 2×10^{16} labelings per tree!

Sheppard
isolated vertex
symmetry
unique graceful labeling
fork
paw
free tree
superexponentially
Otter

Kotzig
GTC
Rosa
Anick
caterpillars
diameter 4
Graceful digraphs
digraphs, graceful
set inclusion
Boolean lattice

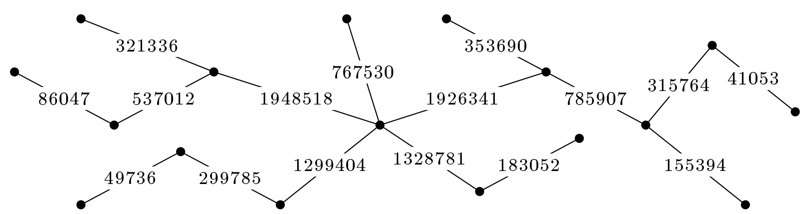
Anton Kotzig conjectured in 1965 that every tree is graceful, and his conjecture soon became famous, even infamous — because nobody could figure out how to prove it, yet all other questions about trees have generally been fairly easy to resolve. Indeed, there are hundreds of people for whom the initials GTC now mean only one thing: Not Green Templeton College, not Girls' Training Corps, not GPU Technology Conference, but Graceful Tree Conjecture.

The GTC is almost certainly true. For example, Alexander Rosa, who invented the concept of graceful graphs while completing his dissertation under Kotzig's direction, proved it already in 1965 for all trees of at most 16 vertices, and for many infinite families of trees. A careful study of the case $m = 16$ by David Anick [*Discrete Applied Mathematics* **198** (2016), 65–81] showed that only a handful of the 48629 free trees with 17 vertices have fewer than 50 labelings; and those few turned out to be obviously graceful, because they all are "caterpillars" (see exercise 115) except for this one of diameter 4:



(41)

At the other extreme, the champion tree has 10,399,350 different labelings. Here it is, with each edge showing the number of times it can be the edge of length 16:



(42)

(Long edges seem to prefer vertices of high degree.) Anick's analysis suggests strongly that all trees of larger sizes will also be easy to label.

***Graceful digraphs.** There's also a nice way to define the concept of a graceful *directed* graph. Suppose D is a simple, loopfree digraph with m arcs. As before we want to assign distinct integers $l(v)$ to its vertices, with $0 \le l(v) \le m$. But now we say that each directed arc $v \longrightarrow w$ implicitly receives the label $\bigl(l(w)-l(v)\bigr) \bmod (m+1)$, respecting the orientation of the arc; and D is *graceful* if those arc labels are distinct. It follows that gracefulness gives us exactly one arc labeled k, for each k between 1 and m.

For example, Fig. 109 shows a digraph that represents set inclusion in a 3-element universe, together with several of its graceful labelings. We can check labeling (b) for gracefulness, just as we did in (33) for the undirected graph in Fig. 105, but this time using the operator $y \ominus x = (y - x) \bmod 13$:

$$\begin{array}{llllll} 1\ominus 0=1 & 9\ominus 6=3 & 8\ominus 3=5 & 7\ominus 0=7 & 3\ominus 7=9 & 4\ominus 6=11 \\ 3\ominus 1=2 & 8\ominus 4=4 & 6\ominus 0=6 & 9\ominus 1=8 & 4\ominus 7=10 & 8\ominus 9=12 \end{array}. \qquad (43)$$

Symmetries
antiautomorphisms
antisymmetries
self-converse

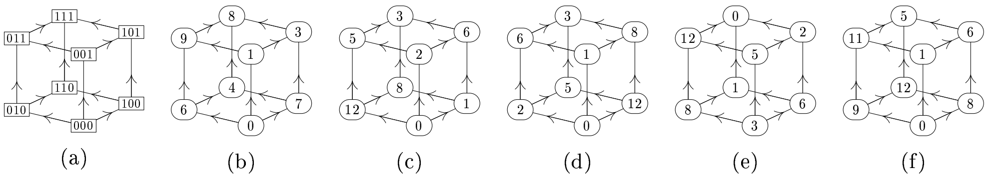


Fig. 109. This directed graph (a) can be gracefully labeled in many ways, some of which are readily derivable from each other. For example, (c) arises from (b) when every vertex label is doubled, modulo 13. (We work mod 13 in this digraph because it has 12 arcs.) Can you see how (d), (e), and (f) were obtained from the others?

Let $q = m + 1$. Cyclic labels mod q are much more versatile mathematically than the absolute-difference labels that we considered before, because (for example) we can add a constant to every vertex label without changing the implied label of any arc. This means we can arbitrarily choose any vertex v and look only for labelings with $l(v) = 0$, when we're trying to decide whether or not a given digraph is graceful. Any graceful labeling with $l(v) = b$ yields one with $l(v) = 0$ after b is subtracted from each label.

Furthermore, when q is a prime number as it is in Fig. 109, we can arbitrarily choose any *two* vertices v and w, and look only for labelings with $l(v) = 0$ and $l(w) = 1$: Given any labeling with $l(v) = 0$, we can multiply all the vertex labels by the number a for which $a \cdot l(w) \equiv 1$ (modulo q). This operation preserves gracefulness, because it implicitly multiplies every arc label by a (modulo q). For example, multiplying Fig. 109(b) by 2 changes the label of vertex 100 from 7 to 1.

Symmetries of the digraph give us yet another way to derive one labeling from another, just as the symmetry `GA` $\leftrightarrow$ `SC` did in Fig. 105. For example, labeling (d) arises from (c) when the label currently assigned to vertex $x_1x_2x_3$ is moved to vertex $x_2x_3x_1$, for each binary vector $x_1x_2x_3$.

Digraphs also bring a new notion into the picture, because they can have *antiautomorphisms* (antisymmetries), which are permutations α of the vertices for which $v \longrightarrow w$ implies $v\alpha \longleftarrow w\alpha$. In general, every digraph D has a *converse* D^T whose arcs all go the other way. A digraph is *self-converse* if and only if it has an antiautomorphism. For example, the mapping $x_1x_2x_3\alpha = \bar{x}_1\bar{x}_2\bar{x}_3$ is an antiautomorphism of the digraph in Fig. 109; hence the labeling in (e), obtained from (d) when each $l(v)$ is replaced by $l(v\alpha)$, gracefully negates each arc label.

Two labelings of a digraph are regarded as essentially the same if we can get one from the other by (i) subtracting b mod q, or (ii) multiplying by a mod q when a is relatively prime to q, or (iii) using an automorphism or antiautomorphism to permute the vertex labels, or (iv) using any combination of transformations (i), (ii), (iii). In this sense, 156 different labelings are essentially equivalent to Fig. 109(b) — including Fig. 109(f). (See exercises 121 and 122.)

Exercise 123 explains how to find all graceful labelings of a given digraph D, by finding representatives of each of its equivalence classes. The first step is to solve an appropriate CSP, using methods adapted from those that work for undirected graphs. Some instructive case studies appear in exercises 124 and 127.

data structures
digraph representation
compact representation

We saw above in (35) that any graceful graph can be represented conveniently within a computer by a set of five compact arrays. Directed graphs turn out to be even *more* attractive in this respect, because only four arrays suffice; a single array `NEXT` replaces the former `NEXTL` and `NEXTH`. For example, here's a compact representation of Fig. 109(a) that corresponds to Fig. 109(b):

$l =$	0	1	2	3	4	5	6	7	8	9	10	11	12
`LO`$[l] =$	—	0	1	6	4	3	0	0	1	7	7	6	9
`FIRST`$[l] =$	7	9	−1	8	8	−1	4	4	−1	8	−1	−1	−1
`NEXT`$[l] =$	—	−1	−1	−1	−1	−1	1	6	3	−1	3	9	−1
`NAME`$[l] =$	000	001	—	101	110	—	010	100	111	011	—	—	—

(44)

As before, the general idea is to include isolated vertices if necessary so that the vertices of the graceful digraph D are $\{0,1,\ldots,m\}$, the same as their labels. The `NAME` array connects these internal numbers with D's external representation, if those vertex names are needed for communication with users.

The `LO` array is crucial. For $1 \le l \le m$, we have `LO`$[l] = v$ if and only if the arc labeled l goes from v to $(v+l) \bmod q$, where $q = m+1$. Consequently it's easy to test whether or not $v \longrightarrow w$ is an arc of D, given v and w, by inspecting a single element of the `LO` array: *That arc is present if and only if* `LO`$[(w-v) \bmod q] = v$.

The `FIRST` and `NEXT` arrays are set up so that we can easily visit every successor of a given vertex v, using the following efficient algorithm:

$$\begin{array}{l}\text{Set } w \leftarrow \texttt{FIRST}[v];\\ \text{while } w \ge 0, \text{ visit } w, \text{ then set } w \leftarrow \texttt{NEXT}[(w-v) \bmod q].\end{array} \qquad (45)$$

Exercise 125 explains one way to derive `FIRST` and `NEXT` from `LO`.

Every array `LO` with $0 \le$ `LO`$[l] \le m$ for $1 \le l \le m$ defines a graceful digraph with m arcs on the vertices $\{0,\ldots,m\}$. Thus the total number of m-arc graceful labelings is exactly $(m+1)^m$. That's much larger than the $m!$ graceful labelings with m *edges* (see Theorem S); exercise 129 shows, however, that we can decrease it by a factor of approximately $2m^2$ when equivalent labelings are lumped together. Thus the complete set of graceful digraphs can be explored without difficulty when m isn't too large.

Digraphs often do turn out to be graceful; for example, 844161 of the 1540944 nonisomorphic digraphs on six vertices can be labeled successfully. But of course there are many exceptions—including half of the "most basic" ones:

Theorem H. *The oriented path $P_n^{\rightarrow}$ and the oriented cycle $C_n^{\rightarrow}$ are both graceful when n is even, but they're both ungraceful when n is odd.*

Proof. The arcs are $v_0 \longrightarrow v_1 \longrightarrow \cdots \longrightarrow v_m$, where $m = n-1$ for $P_n^{\rightarrow}$ and $m = n$ (and $v_m = v_0$) for $C_n^{\rightarrow}$. Suitable labels exist when n is even (see exercise 130).

But there's an unsurmountable problem when n is odd, because the sum (modulo q) of all arc labels, $\bigl(l(v_1)-l(v_0)\bigr) \bmod q + \cdots + \bigl(l(v_m)-l(v_{m-1})\bigr) \bmod q$, is congruent to $l(v_m)-l(v_0)$. This sum should *not* be congruent to zero in the case of the path, but it *should* be congruent to zero in the cycle.

In a graceful digraph the sum of all the arc labels must be $1 + 2 + \cdots + m$, which is $q(q-1)/2$. Hence it's congruent to 0 when q is odd, and it's an odd multiple of $q/2$ when q is even. Contradiction. ▮

An undirected graph is called *digraceful* if there's at least one way to convert it to a graceful digraph by orienting each of its edges. There are 2^m possible orientations of m edges, so this gives us lots of flexibility.

A graceful graph is obviously digraceful as well, because we can orient each edge towards its endpoint whose label is largest. Furthermore, the ungraceful graphs C_{4n+2} are digraceful, because $\vec{C}_{4n+2}$ is graceful by Theorem H. On the other hand, exercise 135 proves that the graphs C_{4n+1} are *not* digraceful.

Is the complete graph K_n digraceful? This is probably the most interesting unsolved question about digracefulness, because every orientation of K_n is called a *tournament.* Graceful tournaments have been studied in other disguises, and they are known to exist for $n = 1$, 2, 3, 4, 5, and 9. (See exercise 137.)

There is, however, a much nicer and more natural way to regard an undirected graph G as a digraph, namely to treat it as the symmetric digraph $G^{\leftrightarrow}$, in which every edge $u \mathrel{—} v$ has been replaced by two arcs $u \longrightarrow v$ and $v \longrightarrow u$. Indeed, as discussed just before 7–(26), G and $G^{\leftrightarrow}$ have essentially the same properties, so we represent them both in the same way inside a computer.

If G has m edges, $G^{\leftrightarrow}$ has $2m$ arcs. Thus the vertex labels of $G^{\leftrightarrow}$ should be chosen modulo $q = 2m + 1$. The labels of $u \longrightarrow v$ and $v \longrightarrow u$ are then negatives of each other, modulo q; and there are just m possibilities, namely $\{\pm1, \pm2, \ldots, \pm m\}$. Consequently we define the label of edge $u \mathrel{—} v$ in $G^{\leftrightarrow}$ to be

$$\begin{aligned} d_L\bigl(l(u), l(v)\bigr) &= \min\bigl(\bigl(l(u) - l(v)\bigr) \bmod q, \bigl(l(v) - l(u)\bigr) \bmod q\bigr) \\ &= \min\bigl(\bigl|l(u) - l(v)\bigr|, q - \bigl|l(u) - l(v)\bigr|\bigr). \end{aligned} \tag{46}$$

(This is the *Lee distance* between the points $l(u)$ and $l(v)$ on a q-cycle; see exercise 7.2.1.1–18.) And now a pleasant thing happens: When we draw K_{2m+1} with its vertices in a circle, it has exactly $2m+1$ edges of Lee distance 1, exactly $2m+1$ edges of Lee distance 2, …, and exactly $2m+1$ edges of Lee distance m. Therefore if $G^{\leftrightarrow}$ is a graceful digraph with m edges, we can pack $2m+1$ copies of G perfectly into K_{2m+1}. (Figure 110 illustrates the case $m = 5$.)

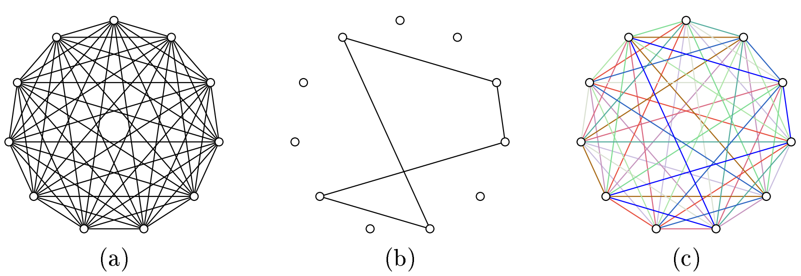


Fig. 110. K_{11} has eleven edges of distance 1, …, and eleven of distance 5. A 5-cycle can be drawn with one edge of each distance. Hence eleven 5-cycles exactly cover K_{11}. "Eleven people can form eleven rings of five, where everybody meets everybody else."

digraceful
orientations
tournament
symmetric digraph
representation of graphs and digraphs
Lee distance

rainbow graceful
cycle, rainbow graceful
Graceful Tree Conjecture
Montgomery
Pokrovskiy
Sudakov
Gallian
embedding
subgraph
image

Let's say that a graph G with m edges is *rainbow graceful* if the corresponding digraph $G^{\leftrightarrow}$ is graceful. This means that we can assign a label $l(v)$ to each vertex v, with $-m \le l(v) \le m$, in such a way that the edge labels $d_L\bigl(l(u), l(v)\bigr)$ defined in (46) are distinct for all m edges $u — v$.

A graceful graph is automatically rainbow graceful, because $d_L\bigl(l(u), l(v)\bigr) = \bigl|l(u) - l(v)\bigr|$ when $l(u)$ and $l(v)$ are nonnegative. Furthermore Fig. 110(b) shows that C_5 is rainbow graceful, although it is neither graceful nor digraceful. In fact — see exercise 139 — there's an astonishingly simple way to prove that *every cycle C_n is rainbow graceful*, for $n \ge 3$, because of the elegant labeling

$$l(k) = (-1)^{k+[2k<n]}k, \qquad \text{for } 1 \le k \le n. \tag{47}$$

A great many graphs are in fact known to be rainbow graceful, and more are being discovered every day. For example, according to the systematic study in exercise 142, every graph on at most 6 vertices is rainbow graceful, except for $K_6 \setminus K_2$ (the 14-edge graph obtained by deleting one of the edges of K_6).

We've seen that graphs with lots of edges are often impossible to label gracefully, because so many labels have to avoid interfering with each other. Yet rainbow labeling is different, because the complete graphs K_5 and K_6 — which have the *maximum* number of edges — do turn out to be labelable! In fact, exercise 145 shows that *K_{n+1} is rainbow graceful whenever n is prime or a power of a prime*. It's remarkable, but true, that K_8, K_9, K_{10}, and K_{12} are rainbow graceful. (On the other hand, K_7, K_{11}, and K_{13} are not.)

The first major steps towards proving the Graceful Tree Conjecture were taken by R. Montgomery, A. Pokrovskiy, and B. Sudakov, who developed new methods in order to prove an asymptotic form of a weaker conjecture:

Theorem M. *All sufficiently large trees are rainbow graceful.*

Proof. See *Geometric and Functional Analysis* **31** (2021), 663–720. ▮

Numerous unresolved questions about gracefulness remain under active investigation, because the number of interesting graphs and digraphs is essentially boundless. Joseph A. Gallian has been actively maintaining a dynamic survey of what is currently known. His annual reports [*Electronic Journal of Combinatorics*, #DS6] began in 1998 with a 46-page review containing 306 references; its 25th edition (2022) had 623 pages (with a 19-page index) and 3295 references.

Graph embedding. Graph G is said* to be *embedded* in graph H if it is isomorphic to a subgraph of H. Informally, this means that H contains a "copy" of G. Formally, it means that there's a function f from the vertices of G to the vertices of H such that two conditions are satisfied:

i) if $v \ne w$ then $f(v) \ne f(w)$;

ii) if $v — w$ in G then $f(v) — f(w)$ in H.

When that happens, we say that "H contains G," and the set of all vertices $\{f(v) \mid v \text{ is a vertex of } G\}$ is called the *image* of G in H.

* People also talk about a graph "embedded in a surface"; that's an entirely different topic.

strict embedding
isometric embedding
shortest distance
`WORDS`(1000)
Stanford GraphBase
five-letter words
clique
snake path: an induced path
induced
Cartesian product of graphs

Embeddings actually come in three flavors. An ordinary vanilla-flavored embedding simply satisfies (i) and (ii); but a stronger version, called a *strict* embedding, also satisfies a third condition:

iii) if $v \mathrel{-\!\!\!\!/\!\!\!-} w$ in G then $f(v) \mathrel{-\!\!\!\!/\!\!\!-} f(w)$ in H.

Stronger yet is an *isometric embedding*, which satisfies even more:

iv) $d_G(v,w) = d_H(f(v), f(w))$, where d denotes the shortest distance.

Notice that condition (iv) by itself implies (i), (ii), and (iii).

For example, suppose G is the five-cycle C_5, and suppose H is `WORDS`(1000), the Stanford GraphBase graph that represents the thousand most common five-letter words of English. One of the zillions of five-cycles in H is

`share` — `spare` — `stare` — `store` — `shore` — `share`. (48)

Formally we could say that the vertices of G are $\{0,1,2,3,4\}$, and that G's edges are v — $((v+1) \bmod 5)$ for $0 \le v < 5$; then $f(0) =$ `share`, ..., $f(4) =$ `shore`. But such formalities are needlessly complicated when we're talking about graphs as simple as C_5; the embedding is immediately clear just from (48).

Example (48) is not a *strict* embedding of C_5, because we have `share` — `stare` in H but $0 \mathrel{-\!\!\!\!/\!\!\!-} 2$ in G. We could in fact have come up with a five-cycle such as

`share` — `shape` — `shade` — `shake` — `shame` — `share`, (49)

in which all five words are mutually adjacent in H; but that seems like cheating, because *any* graph is trivially isomorphic to a subgraph of a complete graph. (This graph `WORDS`(1000) actually contains the 8-clique {`right`, `might`, `night`, `light`, `sight`, `fight`, `tight`, `eight`}; hence it contains a copy of *every* G with up to eight vertices!) The essence of a five-cycle is present in (48), at least partly, but it has been drowned out in (49). A strict embedding retains the full structure, because (ii) and (iii) say that G appears as an *induced* subgraph of H.

There's no way to embed C_5 *strictly* into `WORDS`(1000), because `WORDS`(1000) is a subgraph of $K_{26} \,\square\, K_{26} \,\square\, K_{26} \,\square\, K_{26} \,\square\, K_{26}$; and that graph has no induced C_5 (see exercise 151(f)). Thus a weak embedding like (48) is the best we can get.

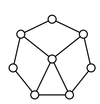
Surprisingly, however, there *is* a strict embedding of the next odd cycle, C_7:

`likes` — `lakes` — `cakes` — `caves`
— `waves` — `wives` — `lives` — `likes`. (50)

This one even turns out to be isometric, in the target graph `WORDS`(1000).

But — surprise, surprise — the induced cycle (50) is *not* isometric in the larger graph `WORDS`(5757) — because that graph contains the somewhat unusual word `laves`. The distance from `lakes` to `waves` in the larger graph is therefore 2, not 3; and the same is true for the distance from `caves` to `lives`.

Notice that if we add the word `laves` to (50), we get an isometric embedding of the graph


(51)

into $K_{26} \,\square\, K_{26} \,\square\, K_{26} \,\square\, K_{26} \,\square\, K_{26}$.

Evidently isometric embeddings are somewhat tricky. Some of their basic properties are explored in exercises 152–159, but we shall concentrate on embeddings of the other two kinds.

Given a pattern graph G and a target graph H, the problem of visiting all embeddings of the pattern in the target is called the *subgraph isomorphism problem* (SIP), and the problem of visiting all of the *strict* embeddings is called the *induced subgraph isomorphism problem* (ISIP). These should be distinguished from the *graph isomorphism problem* (GIP), which is to test whether or not G and H are essentially the same. The GIP is obviously equivalent to testing SIP or ISIP in both directions; but it's much simpler, and it can be attacked by many methods that don't work for the SIP or ISIP. We'll study the GIP in Section 7.2.3.

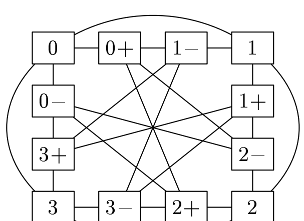
Let's write $G \subseteq H$ if the SIP for pattern G and target H is solvable, and $G \sqsubseteq H$ if the ISIP is solvable. (This is a slight abuse of notation; the relation $G \subseteq H$ really means that $G \cong H'$ for some $H' \subseteq H$, and $G \sqsubseteq H$ really means that $G \cong H \mid U$ for some vertices U of H. But we think of the embedded graph as actually present inside its host.)

The SIP is easily seen to be a CSP, with variables, domains, and constraints: The variables are the vertices of G, the domains are the vertices of H, and the constraints are conditions (i) and (ii). Indeed, we've already noted this characterization of embedding in (6) above. The SIP is, in essence, the CSP that's constrained to be a homomorphism of a given binary relation, together with the all-different constraint.

To fix the ideas, it will be helpful to consider an "organic" example. Figure 111 shows the principal interconnections of a typical human brain, together with two of the subgraphs obtained when only the strongest links are considered.*

Clearly `BRAIN83`(250) is embedded in `BRAIN83`; but a moment's thought shows that it would be pointless to use a subgraph-isomorphism test to verify that fact: The big graph is so rich and twisted, almost *any* not-too-big graph can probably be found within it, in zillions of ways. The interesting question is rather whether a smaller graph with nice structure can be found within `BRAIN83`(250).

Consider, for example, the attractive 4-regular graph called Chvátal's graph. We looked at it long ago in Figure 2(f), near the beginning of Chapter 7; here it is again, with convenient names given to the vertices:



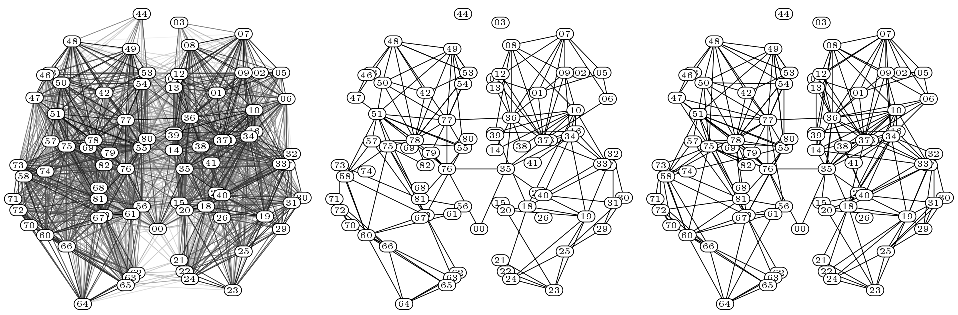
(52)

Can this graph be embedded in the somewhat sparse graph `BRAIN83`(250)?

* See `https://cs.stanford.edu/~knuth/brain83.html` for complete details about this graph, which was constructed from data compiled and simplified by Alain Goriely.

subgraph isomorphism problem
SIP
induced subgraph isomorphism problem
ISIP
graph isomorphism problem
GIP
CSP
homomorphism
all-different constraint
brain graph
connectome, see brain graph
Goriely
internet
BRAIN83+
4-regular graph
Chvátal's graph

exact cover problem
Human Connectome Project



BRAIN83, 1654 edges (a) BRAIN83(250), 170 edges (b) BRAIN83(300), 242 edges (c)

Fig. 111. The graph BRAIN83, based on hundreds of high-resolution brain scans performed by the Human Connectome Project, shows the "wiring diagram" of a healthy human brain. The full graph (a) has 83 vertices (representing the major regions of interest) and 1654 edges (representing channels between them). Vertex 00 is the brain stem; vertices 01–41 form the right brain; and vertices 42–82 form the left brain, with $v+41$ on the left corresponding to v on the right.

Each edge is labeled with an integer $l \geq 0$, which is a logarithmic measure of its importance: The strength of an interconnection is proportional to $e^{-l/1000}$. (However, l is depicted linearly here, with a line that's shaded $l/1350$ of the way from black to white.) The subgraph BRAIN83(250) in (b), which retains only the edges with $l \leq 250$, illustrates some of the strongest interconnections. For example, vertices 77 and 36 are the left and right caudate nuclei, and they are connected by an edge with $l = 33$.

One way to decide this is to set it up as an exact cover problem, following the lead of exercise 7.2.2.1–77, which considered the special case where G and H have the same number of vertices. In general, let there be a primary item v for each vertex v of G, and a secondary item V for each vertex V of H. Let there also be secondary items $e \cdot E$ for every edge e of G and every *non*-edge E of H. The exact cover problem then has one option for each pair (v, V), representing the potential mapping $v \mapsto V$, namely

$$\text{`}v\ \ V\ \ \bigcup\bigl\{e \cdot E \bigm| e = (u \mathrel{-\!\!-} v) \text{ and } E = (U \mathrel{-\!\!\!/\!\!\!-} V) \text{ for some } u \text{ and } U\bigr\}\text{'}. \qquad (53)$$

The solutions to this exact cover problem are precisely the embeddings we want, because (i) every vertex v of G is paired with a distinct vertex V of H; and (ii) we cannot pair u with U and v with V in cases where $u \mathrel{-\!\!-} v$ and $U \mathrel{-\!\!\!/\!\!\!-} V$.

For example, when G is Chvátal's graph (52) and H is BRAIN83(250), G has 12 vertices and 24 edges; H has 68 non-isolated vertices, with $\binom{68}{2} - 170 = 2108$ nonedges between them. Our exact cover problem therefore has 12 primary items, $68 + 24 \cdot 2108 = 50660$ secondary items, and $12 \cdot 68 = 816$ options.

The options are long: Graph H has 65 nonedges involving vertex 00, so every option that pairs v with 00 contains $2 + 4 \cdot 65 = 262$ items. The 816 options therefore have more than 200,000 entries altogether, and Algorithm 7.2.2.1X takes 6 gigamems just to input them before getting started! But then it needs only 2 gigamems to solve the problem — and the result is *no solutions* (no embeddings).

automorphisms
essentially different solutions
United States

We can discover the lack of solutions by being a little smarter when we set up the problem. In the first place, there's no point in considering 00 as a potential target vertex, because that vertex has degree 2; every target vertex will clearly have degree 4 in the image of G, so it must have degree ≥ 4 in H. We can therefore eliminate from BRAIN83(250) not only the 15 isolated vertices, but also the 28 vertices of degrees 1, 2, and 3.

Furthermore, after those vertices go away, other vertices no longer have degree ≥ 4. If we keep pruning vertices of low valency from the graph, we finally reach a graph $\widehat{H}$ that has only 22 vertices, all of degree ≥ 4. This reduced target graph has only $\binom{22}{2} - 63 = 168$ nonedges; and the reduced exact cover problem has fewer than 17000 entries in its 264 options. Algorithm 7.2.2.1X needs just 40 megamems to input them, and 90 megamems to prove them unsolvable.

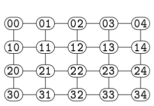
In fact, exercise 173 shows that there's a sneaky way to see that $G \not\subseteq \widehat{H}$ without even running the algorithm.

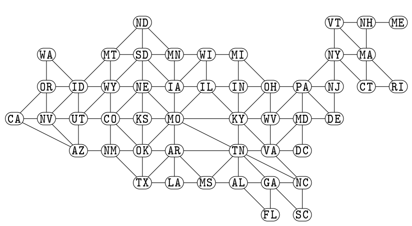
OK; BRAIN83(250) is too sparse to contain Chvátal's graph G. But what about BRAIN83(300)? That graph H has 70 nonisolated vertices, and we can prune it down to a min-degree-4 graph $\widehat{H}$ with only 58 vertices and 211 edges. Now we get an exact cover problem that Algorithm 7.2.2.1X can input in 3 Gμ and solve in 8 Gμ; there are 72 solutions. Therefore G is indeed embeddable into the graph of Fig. 111(c), in 72 ways. (That fact has little or no biological significance, of course; but somehow it's comforting to know that we all have Chvátal's graph rather firmly embedded in our brains.)

All 72 solutions turn out, in fact, to lie entirely within the left brain. But the right brain will contain (52) too, if we add a few more edges of the full graph.

It's significant that 72 is a multiple of 8, because Chvátal's graph has 8 automorphisms (see exercise 7–44). If G is any graph with exactly r automorphisms, the number of functions f that embed G into H is always a multiple of r, because we obtain r distinct embedding functions $f(v\alpha)$ when α ranges over all the automorphisms. Thus there really are only 9 essentially *different* ways to embed (52) into BRAIN83(300). One of them takes $0 \mapsto 48$, $0{+} \mapsto 49$, $1{-} \mapsto 51$, $1 \mapsto 47$, $1{+} \mapsto 77$, $2{-} \mapsto 53$, $2 \mapsto 78$, $2{+} \mapsto 55$, $3{-} \mapsto 58$, $3 \mapsto 75$, $3{+} \mapsto 50$, $0{-} \mapsto 54$; it's essentially the same as the embedding $1 \mapsto 48$, $1{+} \mapsto 49$, $2{-} \mapsto 51$, $2 \mapsto 47$, $\ldots$, $1{-} \mapsto 54$, and to six others. (This solution does not belong to BRAIN83(298), because the edge $48 \mathrel{-\!\!-} 54$ has the label $l = 299$. There are $2 \cdot 8$ embeddings into BRAIN83(293), but none into BRAIN83(292).)

That was fun. Let's try another example, this time with a smaller target so that we can see more closely what is going on. Here's a question about the United States that has perhaps never been asked before:

Is 

$\subseteq$ 

? (54)

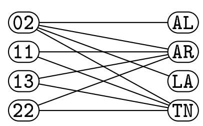
On the left is $P_4 \,\square\, P_5$, a 4×5 grid. On the right is the 49-vertex, 107-edge graph of the continental USA that we saw most recently in Fig. 106. At first glance, smallish grids are visible within the right-hand graph, but a 4×5 seems unlikely.

There are, in fact, three different ways to solve the embedding problem of (54) — that is, $4 \cdot 3$ actual embedding functions, because the grid has four automorphisms. The reader is encouraged to find at least one of them now, by hand, before turning the page to peek at the answer.

Meanwhile let's look at how a computer might attack this problem intelligently. Call the graphs G and H. In the first place, the six interior vertices of G have degree 4; so their domains cannot include any of the 15 states {CA, CT, DC, DE, FL, LA, ME, MI, ND, NH, NJ, RI, SC, VT, WA} of smaller degree.

We can shrink the domains even further by looking at the degrees of neighbors. For example, the neighbors of 11 in G have degrees $\{3,3,4,4\}$, while the neighbors of GA in H have degrees $\{2,2,4,4,8\}$. Therefore no embedding of G into H can map 11 $\mapsto$ GA. (See exercise 178.) In a similar way we can remove AL, GA, MA, NC, OR from the domains of 11, 12, 13, 21, 22, and 23. Furthermore the neighbors of NY in H have degrees $\{3,3,3,5,6\}$; this doesn't rule out 11 $\mapsto$ NY, but it does show that we can't map 12 $\mapsto$ NY or 22 $\mapsto$ NY. That leaves just 28 possibilities in the initial domains of G's "middle" vertices 12 and 22.

An even closer look shows that we can't take 12 $\mapsto$ MS. For if we did, there would be a matching of size 4 in the bipartite graph



. (55)

The left part here shows the neighbors of 12; they must each match a vertex in their domain that also happens to be a neighbor of MS. There's no such matching. Similar analyses rule out the mappings 11 $\mapsto$ OR, 02 $\mapsto$ MA, and so on. This technique for domain reduction was introduced by C. Solnon [*Artificial Intelligence* **174** (2010), 850–864], who called it LAD filtering (for "Locally All Different").

We now begin to form a search tree, with 27 possibilities to try for the image of 12. The first of these, alphabetically, is AZ, so let's tentatively map 12 $\mapsto$ AZ. This means we remove AZ from every other domain, and restrict the domains of 02, 11, 13, and 22 to neighbors of AZ. Hmm; we soon reach an impasse, because 21 has no place to go: It must map to a neighbor of the domains of 11 and 22, namely a neighbor of {NM, NV, UT}; but LAD filtering proves that impossible.

The next thing to try is 12 $\mapsto$ AR. This option is somewhat more plausible; LAD filtering whittles the domains down quite a bit, but not too far. They are

$$\begin{pmatrix} \mathtt{i} & \mathtt{e} & \mathtt{d} & \mathtt{e} & \mathtt{i} \\ \mathtt{h} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{h} \\ \mathtt{g} & \mathtt{c} & \mathtt{b} & \mathtt{c} & \mathtt{g} \\ \mathtt{j} & \mathtt{g} & \mathtt{f} & \mathtt{g} & \mathtt{j} \end{pmatrix}; \quad \begin{array}{ll} \mathtt{a} = \{\mathtt{AR}\}, & \mathtt{d} = \mathtt{b} \cup \{\mathtt{LA},\mathtt{MS},\mathtt{TX}\}, \\ \mathtt{b} = \{\mathtt{MO},\mathtt{OK},\mathtt{TN}\}, & \mathtt{e} = \mathtt{b} \cup \mathtt{c} \cup \{\mathtt{AL},\mathtt{MS},\mathtt{NM},\mathtt{TX}\}, \\ \mathtt{c} = \{\mathtt{KS},\mathtt{KY},\mathtt{MO}\}, & \mathtt{f} = \mathtt{b} \cup \mathtt{c} \cup \{\mathtt{CO},\mathtt{IA},\mathtt{IL},\mathtt{NE},\mathtt{VA}\}, \\ \mathtt{g} = \mathtt{f} \cup \{\mathtt{IN},\mathtt{WV}\}, & \mathtt{i} = \mathtt{e} \cup \mathtt{g} \cup \mathtt{h} \cup \{\mathtt{GA}\}, \\ \mathtt{h} = \mathtt{f} \cup \{\mathtt{NC},\mathtt{NM}\}, & \mathtt{j} = \mathtt{g} \cup \mathtt{h} \cup \{\mathtt{MD},\mathtt{OH},\mathtt{SD},\mathtt{WI},\mathtt{WY}\}. \end{array} \tag{56}$$

grid
continental USA
automorphisms
initial domains
maximum bipartite matching
matching
bipartite graph
Solnon
LAD filtering–

GAD filtering
all-different
Sudoku

For example, the domain of 11, 13, and 22 is {MO, OK, TN}; the domain of 02 is {LA, MO, MS, OK, TN, TX}; and the domain of 32 has 10 elements.

At this point we turn to a complementary technique, known as GAD filtering (for "Globally All Different"). The idea is again to solve a bipartite matching problem; but our goal this time is to match *every* pattern vertex with some element of its current domain. (Because if no such matching exists, the current domains are too small and we must backtrack.)

The domains in (56) readily yield such a matching. For example, here's one:

$$\begin{pmatrix} \mathtt{VA} & \mathtt{NM} & \mathtt{TX} & \mathtt{MS} & \mathtt{AL} \\ \mathtt{NC} & \mathtt{TN} & \mathtt{AR} & \mathtt{OK} & \mathtt{CO} \\ \mathtt{IA} & \mathtt{KY} & \mathtt{MO} & \mathtt{KS} & \mathtt{WV} \\ \mathtt{SD} & \mathtt{NE} & \mathtt{IL} & \mathtt{IN} & \mathtt{WY} \end{pmatrix}. \qquad (57)$$

Of course this doesn't solve our subgraph isomorphism problem — Virginia is nowhere near New Mexico, and there are many other faults. But VA does belong to the current domain of 00, according to (56), and NM does belong to the domain of 01. The advantage of (57) is that the theory of bipartite matching gives us an efficient way to trim off all the "excess fat" from the domains of variables that are required to be all-different. Indeed, the algorithm of exercise 185 uses (57) to reduce (56) substantially, so that only the following domains are left:

$$\begin{pmatrix} \mathtt{i} & \mathtt{e} & \mathtt{d} & \mathtt{e} & \mathtt{i} \\ \mathtt{h} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{h} \\ \mathtt{g} & \mathtt{c} & \mathtt{b} & \mathtt{c} & \mathtt{g} \\ \mathtt{j} & \mathtt{g} & \mathtt{f} & \mathtt{g} & \mathtt{j} \end{pmatrix}; \quad \begin{array}{ll} \mathtt{a} = \{\mathtt{AR}\}, & \mathtt{d} = \{\mathtt{LA},\mathtt{MS},\mathtt{TX}\}, \\ \mathtt{b} = \{\mathtt{MO},\mathtt{OK},\mathtt{TN}\}, & \mathtt{e} = \{\mathtt{AL},\mathtt{MS},\mathtt{NM},\mathtt{TX}\}, \\ \mathtt{c} = \{\mathtt{KS},\mathtt{KY}\}, & \mathtt{f} = \{\mathtt{CO},\mathtt{IA},\mathtt{IL},\mathtt{NE},\mathtt{VA}\}, \\ \mathtt{g} = \mathtt{f} \cup \{\mathtt{IN},\mathtt{WV}\}, & \mathtt{i} = \mathtt{e} \cup \mathtt{g} \cup \mathtt{h} \cup \{\mathtt{GA}\}, \\ \mathtt{h} = \mathtt{f} \cup \{\mathtt{NC},\mathtt{NM}\}, & \mathtt{j} = \mathtt{g} \cup \mathtt{h} \cup \{\mathtt{MD},\mathtt{OH},\mathtt{SD},\mathtt{WI},\mathtt{WY}\}. \end{array} \qquad (58)$$

Notice, for example, that (56) had MO in 19 of the 20 domains; the only exception was 'a', the domain of the pattern vertex 12 that we've tentatively mapped to AR. But in (58), MO belongs only to 'b', which is the domain of pattern vertices 11, 13, and 22.

Sudoku experts will see why MO can be dropped from 16 of the 19 domains where it was formerly present: Any all-different assignment using (56) must map {11, 13, 22} into {MO, OK, TN}. Hence those three values can't be used elsewhere.

Similarly, we now know that 21 and 23 can't be mapped to MO; so they must map to {KS, KY}. We can therefore eliminate KS and KY from all domains but c.

GAD filtering, which reduces (56) to (58), is not specific to the subgraph isomorphism problem; it applies to *any* CSP with an all-different constraint. No further reduction from (58) is possible, from that global standpoint.

But the smaller domains in (58) now let us make further progress on our SIP, (54), by going back to LAD filtering, because the local bipartite graphs have gotten significantly smaller. Indeed, exercise 179 shows that a contradiction soon arises. Thus we learn that the tentative mapping 12 $\mapsto$ AR is impossible.

So we try 12 $\mapsto$ CO next. LAD filtering is now able to remove 300 elements from the other 19 domains; that's good, yet it's significantly fewer than the 379

GAD filtering
Supplemental
McCreesh
Prosser
Trimble
Chvátal

LAD deletions that we had in the previous case. So our new LAD-consistent domains are not as constrained as those in (56) above:

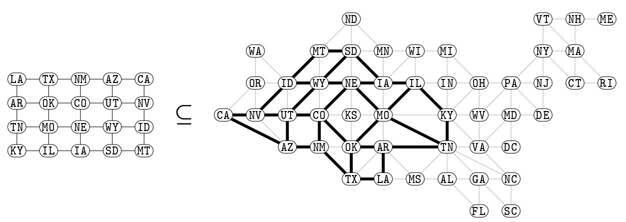
$$\begin{pmatrix} \mathtt{i} & \mathtt{e} & \mathtt{d} & \mathtt{e} & \mathtt{i} \\ \mathtt{h} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{h} \\ \mathtt{g} & \mathtt{c} & \mathtt{b} & \mathtt{c} & \mathtt{g} \\ \mathtt{j} & \mathtt{g} & \mathtt{f} & \mathtt{g} & \mathtt{j} \end{pmatrix}; \quad \begin{array}{ll} \mathtt{a} = \{\mathtt{CO}\}, & \mathtt{c} = \mathtt{x} \cup \{\mathtt{AZ}, \mathtt{ID}, \mathtt{MO}, \mathtt{SD}, \mathtt{TX}\}, \\ \mathtt{b} = \mathtt{d}, & \mathtt{d} = \mathtt{x} \cup \{\mathtt{KS}, \mathtt{NM}, \mathtt{UT}\}, \\ \mathtt{f} = \mathtt{h}, & \mathtt{e} = \mathtt{c} \cup \{\mathtt{KS}\}, \\ \mathtt{g} = \mathtt{i}, & \mathtt{h} = \mathtt{e} \cup \{\mathtt{AR}, \mathtt{IA}, \mathtt{MT}, \mathtt{NV}, \mathtt{UT}\}, \\ \mathtt{x} = \{\mathtt{NE}, \mathtt{OK}, \mathtt{WY}\}, & \mathtt{i} = \mathtt{h} \setminus \{\mathtt{AZ}\} \cup \mathtt{y} \cup \{\mathtt{IL}, \mathtt{LA}, \mathtt{MN}, \mathtt{ND}, \mathtt{TN}\}, \\ \mathtt{y} = \{\mathtt{CA}, \mathtt{NM}, \mathtt{OR}\}, & \mathtt{j} = \mathtt{i} \cup \{\mathtt{AZ}, \mathtt{KY}, \mathtt{MS}, \mathtt{WA}, \mathtt{WI}\}. \end{array} \tag{59}$$

In this situation GAD filtering makes no change. So we need to branch again; let's try $\mathtt{11} \mapsto \mathtt{OK}$. Hurray! LAD filtering now reduces most of the domains to singletons:

$$\begin{pmatrix} \{\mathtt{LA}\} & \{\mathtt{TX}\} & \{\mathtt{NM}\} & \{\mathtt{AZ}\} & \{\mathtt{CA}, \mathtt{NV}\} \\ \{\mathtt{AR}\} & \{\mathtt{OK}\} & \{\mathtt{CO}\} & \{\mathtt{UT}\} & \{\mathtt{ID}, \mathtt{NV}\} \\ \{\mathtt{TN}\} & \{\mathtt{MO}\} & \{\mathtt{NE}\} & \{\mathtt{WY}\} & \{\mathtt{ID}, \mathtt{MT}\} \\ \{\mathtt{KY}\} & \{\mathtt{IL}\} & \{\mathtt{IA}\} & \{\mathtt{SD}\} & \{\mathtt{MT}, \mathtt{ND}\} \end{pmatrix}. \tag{60}$$

So we're almost done. Branching on $\mathtt{04} \mapsto \mathtt{CA}$ gives us Fig. 112; and the other branch gives a second solution (see exercise 180).

Fig. 112. One of the three ways to embed $P_4 \,\square\, P_5$ into the graph USA.



***Supplemental labels and graphs.** We've now seen how to solve problem (54), using a mixture of LAD and GAD filtering to keep the backtrack tree reasonably small. And there's another important technique that we could also have used, based on the fact that subgraph isomorphism is quite a strong property. [See C. McCreesh and P. Prosser, *LNCS* **9255** (2015), 295–312; C. McCreesh, P. Prosser, and J. Trimble, *LNCS* **12150** (2020), 316–324.] Notice, for example, that one subgraph isomorphism always implies another:

$$\text{If } G \subseteq H, \text{ then } G^{\leq 2} \subseteq H^{\leq 2}, \text{ with the same embedding.} \tag{61}$$

Here $G^{\leq 2}$ denotes the graph whose vertices are the same as those of G, but whose edges $u \longrightarrow v$ exist if and only if there's a path of length ≤ 2 between u and v in G. If the function f embeds G into H, and if there's such a path in G, then there's clearly also a path of length ≤ 2 between $f(u)$ and $f(v)$ in H.

With (61) we can improve on what we did before. For example, suppose G is Chvátal's graph (52). Then $G^{\leq 2} = K_{12}$ and every vertex has degree 11, since the diameter is 2. But if H is BRAIN83(300), its vertices 30, 70, and 71 have degree only 9 in $H^{\leq 2}$. Therefore we can omit those three vertices from all domains, and it turns out that the SIP computation will take only 83% as long as before.

supplemental label
motif
initial domains
monotone
supplemental edge label
cycle C_k
LAD filtering
supplemental pair label

We didn't actually need the full strength of (61) in this particular case; all we used was the *degrees* of vertices in $G^{\leq 2}$ and $H^{\leq 2}$. In general, a *supplemental label* for a vertex is any function d_G for which the following property holds:

$$\text{If } G \subseteq H \text{ via embedding function } f \text{, then } d_G(v) \leq d_H(f(v)) \text{ for all } v \in G. \qquad (62)$$

The degree of v in $G^{\leq 2}$ is just one example of a supplemental label.

Suppose S is an arbitrary graph, with a designated vertex s, and let $d_G^S(v)$ be the number of embeddings of S into G that map v to s. Then d_G^S is a supplemental label, because those embeddings of S into G will also be embeddings of S into the image $f(G)$ within H. We can think of S as a local "motif."

If we can somehow discover a motif S that occurs frequently in the pattern G but less often in the target H, the labels d_G^S and d_H^S will help reduce the size of initial domains when we try to embed G into H. (See also exercise 178.)

Supplemental labels can be combined in numerous ways. If d_G and d'_G are any two supplemental labels, so are $\min(d_G, d'_G)$, $\max(d_G, d'_G)$, and $\alpha d_G + \beta d'_G$ whenever $\alpha, \beta \geq 0$; indeed, so is *any* monotone combination of d_G and d'_G.

Furthermore, supplemental labels can be derived for edges as well as vertices. A supplemental edge label is a function ℓ_G for which we can prove the following:

$$\begin{aligned}&\text{If } G \subseteq H \text{ via embedding function } f,\\ &\qquad \text{then } \ell_G(u,v) \leq \ell_H(f(u), f(v)) \text{ whenever } u \mathbin{—} v \text{ in } G.\end{aligned} \qquad (63)$$

(It's possible to have $\ell_G^S(u,v) \neq \ell_G^S(v,u)$.) For example, let S be a motif graph in which two adjacent vertices, $s \mathbin{—} t$, have been designated; and let $\ell_G^S(u,v)$ be the number of ways we can embed S into G with $u \mapsto s$ and $v \mapsto t$. Then ℓ_G^S is a supplemental edge label, by the same reasoning we used for d_G^S above. And supplemental edge labels can be combined monotonically as before. Notice that, when S is the cycle C_k, ℓ_G^S is the number of k-cycles in G that contain a given edge.

A well-chosen supplemental edge label can significantly enhance LAD filtering. Let's go back to the USA problem of (54) and label each edge $u \mathbin{—} v$ by the number $\ell_G(u,v)$ of 4-cycles that it supports. Then ℓ_G equals 2 on every internal edge of $G = P_4 \square P_5$; and ℓ_H has interesting diversity on the edges of $H = \texttt{USA}$. We can now, for example, prove that $\texttt{11} \mapsto \texttt{NY}$ is impossible: The neighbors of `11` are `01`, `10`, `12`, and `21`, all linked by edges with $\ell_G = 2$; the neighbors of `NY` are `CT`, `MA`, `NJ`, `PA`, `VT`, whose ℓ_H labels are respectively 2, 2, 1, 1, 2. LAD filtering rules this out, because the bipartite problem requires the four pattern vertices to match only three target vertices $\{\texttt{CT}, \texttt{MA}, \texttt{VT}\}$. Similar reasoning shows that $\texttt{11} \not\mapsto \texttt{AZ}$, `NM`, `WI`, and 17 other targets that non-supplemental arguments had previously ruled out. The same pruning applies also, of course, to the domain of `12`.

More generally, a *supplemental pair label* ℓ_G satisfies a stronger condition:

$$\begin{aligned}&\text{If } G \subseteq H \text{ via embedding function } f,\\ &\qquad \text{then } \ell_G(u,v) \leq \ell_H(f(u), f(v)) \text{ for all vertices } u \text{ and } v \text{ in } G.\end{aligned} \qquad (64)$$

One way to get such a function is to designate two *non*-adjacent vertices s and t in a motif graph, and to define ℓ_G^S just as we did above. A supplemental pair label obtained in this way might turn out to be nonzero when $u \mathbin{—} v$.

Finally there's an even more powerful notion, a *supplemental graph*, which is a (possibly directed) graph on the same vertices but usually with a different adjacency relation. Suppose the following statement is true:

$$\text{If } G \subseteq H, \text{ then } G^{\Sigma} \subseteq H^{\Sigma}, \text{ with the same embedding.} \qquad (65)$$

Then we say that G^{Σ} and H^{Σ} are a pair of supplemental graphs. (We began this discussion with such a pair, in (61).)

For example, if ℓ_G is a supplemental pair label, we get a supplemental graph by letting $u \longrightarrow v$ if and only if $\ell_G(u,v) \geq k$, for any threshold k. (And we conventionally write u —— v if and only if we have both $u \longrightarrow v$ and $v \longrightarrow u$.) Let's say that $G^{S,k}$ is the supplemental graph we obtain in this way from the supplemental pair label ℓ_G^S. (Examples can be found in exercises 196 and 198.) The union and intersection of supplemental graphs is a supplemental graph.

And once we have a supplemental graph, we can use it to define *further* supplemental labels and graphs, based on *its* motifs!

We're clearly faced here with an embarrassment of riches. Innumerable supplemental labels and graphs can potentially be computed, perhaps turning a huge search tree into a mere shrub. On the other hand, supplemental data based on motifs that don't occur anywhere in the pattern is totally useless. A delicate balancing act is required when solving a SIP, and indeed when solving *any* CSP: It's great to reduce the number of search nodes by a factor of 10, but not when the computation time per node increases by a factor of 100, and not when there aren't extremely many nodes in the first place.

Thus a well-engineered SIP solver does its best to concentrate on supplemental data that justifies the time and space needed to compute it. We can judiciously relax our standards of LAD and GAD filtering, if our data structures allow us to do a pretty-good-but-incomplete job at high speed, as long as we don't change the set of solutions. Maximum bipartite matching problems are solved quickly by the Hopcroft–Karp algorithm (Algorithm 7.5.1H on page x); but the existence of a suitably large matching can often be ruled out even more quickly by rudimentary tests. (See exercises 201–204.)

When C. Solnon surveyed the state of the art of SIP solving [*LNCS* **11510** (2019), 1–13], she observed that it's wise to feed your problem first to a comparatively simple solver that polishes off easy instances quickly. You can solve more problems in a given amount of time if you start in that way, but switch to heavier artillery if that solver doesn't finish in, say, 0.1 seconds.

Some SIP problems are extremely difficult indeed. So we can expect continued progress towards methods that ameliorate their solution — perhaps by understanding more about how to find fruitful motifs in a given pattern and target.

Special cases of subgraph isomorphism. The general SIP has many special cases that are well known by other names. For example, when the pattern graph is a path or a cycle having the same number of vertices as the target graph, the problem is to find a Hamiltonian path or Hamiltonian cycle. Special techniques apply to that problem, and we shall discuss them at length in Section 7.2.2.4. Similarly, when the pattern graph is a clique, the special methods discussed in

supplemental graph
LAD
GAD
Maximum bipartite matching
Hopcroft–Karp algorithm
Solnon
Hamiltonian
clique

Section 7.2.2.5 become available. And when the pattern graph is the same as the target graph, the solutions to the SIP are the automorphisms of that graph.

An n-vertex graph G is three-colorable if and only if $G \subseteq K_{n,n,n}$. It has bandwidth $\le k$ if and only if $G \subseteq P_n^k$, where P_n^k is the graph on $\{0, 1, \ldots, n-1\}$ with $u \mathrel{—\!\!—} v$ if and only if $|u - v| \le k$.

The special case when both pattern and target are free trees is perhaps the nicest of all, for in that case the SIP can be solved with a beautiful algorithm published by David W. Matula in 1978. His algorithm (see exercise 213) has a running time of $O(m^{1.5}n)$ in the worst case, when the pattern size is m and the target size is n; and its running time in practice is typically of order mn.

The fact that subtree isomorphism can be handled so efficiently might lead us to suspect that "subdag isomorphism" — when both pattern and target are directed acyclic graphs — might also be fairly easy. All such hopes are dashed, however, by the simple construction in exercise 168, which shows that *every* SIP can be regarded as a special case of subdag isomorphism.

The special case of trees cannot even be extended to forests: If the pattern graph G consists of disconnected trees, the problem of deciding whether or not $G \subseteq H$ turns out to be NP-hard, even when H is a free tree and G has an extremely simple form. (See exercise 163.)

On the other hand, if the pattern G is simply a collection of disjoint edges, $P_2 \oplus \cdots \oplus P_2$, an embedding of G is the same thing as a *matching*, and again we can test $G \subseteq H$ efficiently. The Hopcroft–Karp algorithm does this well when H is bipartite, and other methods work for *arbitrary* H (see Section 7.5.5).

Solving a CSP. So far we've been looking at lots of different kinds of constraint satisfaction problems; and an endless variety of further applications beckons. But it's time now to think systematically about general approaches that we might take when we're faced with a new CSP.

In the first place, we can always basically start from scratch, and write a standalone program that's specifically tailored to whatever special problem we have in mind. In fact, Algorithm 7.2.2B, the basic backtrack algorithm, is still the method of choice for sufficiently simple tasks,* as well as for comparatively unstructured tasks like those in exercises 7.2.2–71 and 79. The CSP framework of variables, domains, and constraints has also suggested *refinements* of backtracking, such as backmarking (see exercise 306).

In the second place, we can formulate any CSP as an XCC problem — exact covering with colors — and use the versatile methods of Section 7.2.2.1. Exercise 4 is a simple example of this general principle, and further examples can be found in exercises 44 (line labeling) and 69 (graceful labeling). Similarly, exercise 20 solves the car sequencing problem as an MCC, using Algorithm 7.2.2.1M for *non*exact covering. The notions of items and options often turn out to be more directly related to a problem than the notions of variables, domains, and constraints; for example, we saw in (53) that subgraph isomorphism

* The author still finds himself turning back to that algorithm about once a month, since customizations of 7.2.2B continue to be useful and fun, even after 60 years of experience!

automorphisms
three-colorable
bandwidth
free trees
trees
Matula
subtree isomorphism
subdag isomorphism
directed acyclic graphs
forests
NP-hard
matching
Hopcroft
Karp
bipartite
author
backtracking
backmarking
XCC
line labeling
graceful labeling
car sequencing problem
MCC
subgraph isomorphism

is conveniently expressed as an XC problem — exact covering *without* colors. Another instructive example is the "rainbow path problem" in the answer to exercise 212.

In the third place, we can formulate any CSP as a satisfiability problem, and use the extremely well-developed SAT solvers discussed in Section 7.2.2.2. This approach is often the way to go, especially if we want to find only one solution instead of the complete set, and we'll soon examine it in greater detail.

In the fourth place, we can choose from many well-designed computer programs that have been developed specifically for problems that conform explicitly to the CSP model. The task of designing a complete, general-purpose CSP solver is beyond the scope of this book; however, we shall study several of the important techniques that have been devised for such systems. A large community of researchers in constraint processing has developed new methods that enhance what we've already seen in Sections 7.2.2.1 and 7.2.2.2.

XC problem
rainbow path problem
satisfiability
SAT solvers
CSP solver
satisfiability problem
ternary domain
balanced
clauses
at least one
at most one
direct encoding
binary notation
log encoding

Translating CSP to SAT. The most obvious difference between the satisfiability problem that we considered in Section 7.2.2.2 and the more general CSP is the fact that satisfiability is based on *Boolean* variables, while the variables of a CSP usually have domains with *more* than two values. Large domains can, however, be represented with small domains, if we increase the number of variables.

Let's look first at the simplest non-binary case, where all CSP variables have the ternary domain $\{0,1,2\}$. (We could consider the "balanced" domain $\{-1,0,+1\}$ instead; and indeed, $\{-1,0,+1\}$ is the domain of choice in many applications. But all ternary domains are essentially equivalent to $\{0,1,2\}$; and we'll soon be studying domains $\{0,1,\ldots,d-1\}$ for $d>3$.)

One natural way to represent a ternary variable v SATwise is to encode it as three binary variables, $\{v_0,v_1,v_2\}$, where $v_j=[v=j]$. The three possible triplets $v_0v_1v_2$ are then $\{100,010,001\}$; and the other five triplets, $\{000,\ 011,\ 101,\ 110,\ 111\}$ can be excluded by introducing four clauses into our SAT problem:

$$(v_0 \vee v_1 \vee v_2); \tag{66}$$

$$(\bar v_0 \vee \bar v_1) \ \wedge\ (\bar v_0 \vee \bar v_2) \ \wedge\ (\bar v_1 \vee \bar v_2). \tag{67}$$

Clause (66) says that v has *at least one* value, namely that $v_0+v_1+v_2\ge 1$; clauses (67) say that v has *at most one* value, namely that $v_0+v_1+v_2\le 1$. We've often seen this so-called *direct encoding* before, for instance in Eq. 7.2.2.2–(13).

A closer look shows that v_0 is really unnecessary here, because the three allowable pairs $v_1v_2=\{00,10,01\}$ are distinct. In fact, if we read those pairs in the opposite order, v_2v_1, we get 00, 01, and 10, which are the values 0, 1, and 2 in binary notation! When v_0 is dropped, we need only one constraint to ensure uniqueness of v's value,

$$(\bar v_1 \vee \bar v_2), \tag{68}$$

instead of the four in (66) and (67). This method is called the *log encoding*, because it generalizes to a representation of d values with only $\lceil \lg d\rceil$ binary variables. (At least $\lceil \lg d\rceil$ of them are needed, to distinguish between d cases.)

Table 2

ENCODING '$u \neq v$' WITH TERNARY DOMAINS

Name	Clauses for u	Clauses for v	Clauses for u and v
Direct	$(u_0 \vee u_1 \vee u_2)$ $(\bar{u}_0 \vee \bar{u}_1)$ $(\bar{u}_0 \vee \bar{u}_2)$ $(\bar{u}_1 \vee \bar{u}_2)$	$(v_0 \vee v_1 \vee v_2)$ $(\bar{v}_0 \vee \bar{v}_1)$ $(\bar{v}_0 \vee \bar{v}_2)$ $(\bar{v}_1 \vee \bar{v}_2)$	$(\bar{u}_0 \vee \bar{v}_0)$ $(\bar{u}_1 \vee \bar{v}_1)$ $(\bar{u}_2 \vee \bar{v}_2)$
Multivalued	$(u_0 \vee u_1 \vee u_2)$	$(v_0 \vee v_1 \vee v_2)$	$(\bar{u}_0 \vee \bar{v}_0)$ $(\bar{u}_1 \vee \bar{v}_1)$ $(\bar{u}_2 \vee \bar{v}_2)$
Log	$(\bar{u}_1 \vee \bar{u}_2)$	$(\bar{v}_1 \vee \bar{v}_2)$	$(u_2 \vee u_1 \vee v_2 \vee v_1)$ $(\bar{u}_1 \vee \bar{v}_1)$ $(\bar{u}_2 \vee \bar{v}_2)$
Binary			$(u_2 \vee u_1 \vee v_2 \vee v_1)$ $(u_2 \vee \bar{u}_1 \vee v_2 \vee \bar{v}_1)$ $(\bar{u}_2 \vee u_1 \vee \bar{v}_2 \vee v_1)$ $(u_2 \vee u_1 \vee \bar{v}_2 \vee \bar{v}_1)$ $(\bar{u}_2 \vee \bar{u}_1 \vee v_2 \vee v_1)$ $(\bar{u}_2 \vee \bar{u}_1 \vee \bar{v}_2 \vee \bar{v}_1)$
Support	$(u_0 \vee u_1 \vee u_2)$ $(\bar{u}_0 \vee \bar{u}_1)$ $(\bar{u}_0 \vee \bar{u}_2)$ $(\bar{u}_1 \vee \bar{u}_2)$	$(v_0 \vee v_1 \vee v_2)$ $(\bar{v}_0 \vee \bar{v}_1)$ $(\bar{v}_0 \vee \bar{v}_2)$ $(\bar{v}_1 \vee \bar{v}_2)$	$(\bar{u}_0 \vee v_1 \vee v_2)$ $(\bar{u}_1 \vee v_0 \vee v_2)$ $(\bar{u}_2 \vee v_0 \vee v_1)$ $(u_0 \vee u_1 \vee \bar{v}_2)$ $(u_0 \vee u_2 \vee \bar{v}_1)$ $(u_1 \vee u_2 \vee \bar{v}_0)$
Weakened	$(u_0 \vee u_1 \vee u_2)$	$(v_0 \vee v_1 \vee v_2)$	$(\bar{u}_0 \vee u_1 \vee u_2 \vee \bar{v}_0 \vee v_1 \vee v_2)$ $(\bar{u}_1 \vee u_2 \vee \bar{v}_1 \vee v_2)$ $(\bar{u}_2 \vee \bar{v}_2)$
Reduced			$(u_1 \vee u_2 \vee v_1 \vee v_2)$ $(\bar{u}_1 \vee \bar{v}_1)$ $(\bar{u}_2 \vee \bar{v}_2)$
Prefix			$(u_2 \vee u_1 \vee v_2 \vee v_1)$ $(u_2 \vee \bar{u}_1 \vee v_2 \vee \bar{v}_1)$ $(\bar{u}_2 \vee \bar{v}_2)$
Order	$(\bar{u}^2 \vee u^1)$	$(\bar{v}^2 \vee v^1)$	$(u^1 \vee v^1)$ $(\bar{u}^1 \vee u^2 \vee \bar{v}^1 \vee v^2)$ $(\bar{u}^2 \vee \bar{v}^2)$

Many other encodings are also possible. Indeed, we've already made an extensive study of the mappings $x \mapsto x_l x_r$ by which a ternary variable x can be represented by a pair of binary variables, as part of our study of Boolean techniques: Equations 7.1.3–(110) through 7.1.3–(131) showed that the best such mapping depends heavily on the context in which the representation is used.

The context of a SAT encoding within a CSP is, of course, the set of constraints that involve the encoded variable. So let's consider how to express a given relation between two ternary variables u and v, when u and v have both been suitably encoded. We might as well begin with the simplest such relation that arises frequently in applications, namely nonequality: '$u \neq v$'.

Table 2 shows nine ways to represent ternary nonequality via SAT clauses. Some clauses are usually needed for u by itself and for v by itself; then there are clauses that involve both u and v. In the direct encoding, for example, Table 2

inequality relation ($x \neq y$), see nonequality
disequality, see nonequality relation
not equality, see nonequality rel
direct encoding

lists (66) and (67) for both variables, followed by three clauses $(\bar{u}_j \vee \bar{v}_j)$ to ensure that we don't simultaneously have $u = j$ and $v = j$.

The *multivalued encoding* is like the direct encoding, except that it omits the at-most-one clauses (67). If, say, there's a solution with $u_0 = u_1 = 1$, we can obtain two other solutions by changing either u_0 or u_1 to zero; in either case u will remain unequal to v, because $u_0 = u_1 = 1$ implies that $v_0 = v_1 = 0$.

The three clauses of the *log encoding* that forbid $u = v$ in Table 2 are the ones that don't allow the quadruple $u_2u_1v_2v_1$ to be 0000, ∗1∗1, or 1∗1∗.

The *binary encoding* is similar to the log encoding, but it allows *both* 11 and 00 as acceptable encodings of the domain value 0. Therefore we must forbid not only 0000, 0101, and 1010, but also 0011, 1100, and 1111.

The *support encoding* (see exercise 7.2.2.2–399) starts out like the direct encoding; but its clauses that make $u \neq v$ are quite different. For example, the clause '$(\bar{u}_0 \vee v_1 \vee v_2)$' says that $u = 0$ implies $v = 1$ or $v = 2$.

Exercise 218 explains the *weakened encoding* and the *prefix encoding*.

The *reduced encoding* is the most economical of all. Eight values of the quadruple $u_0u_1v_0v_1$ are permissible, each of which forces $u \neq v$ (see exercise 219).

Finally, Table 2 concludes with the *order encoding*, also called the *unary encoding*, which is another important idea that we've studied earlier. In this case $v^j = [v \geq j]$ (see Eq. 7.2.2.2–(163)). However, order encoding is not a really new alternative when $d = 3$, because the possible values $v^1v^2 = 00, 10, 11$ are equivalent to the log-encoded values $v_2v_1 = 10, 00, 01$, if $v^1 \leftrightarrow \bar{v}_2$ and $v^2 \leftrightarrow v_1$.

It's a nice theory. How well do these encodings work in practice? Notice that the CSP with domains $\{0, 1, \ldots, d-1\}$ and constraints $u \neq v$ between certain pairs of variables is precisely the problem of *coloring a graph with d colors*. So we can apply any of the nine encodings to the vertices and edges of any given graph G, and use a SAT solver to see whether or not G is 3-colorable. [In fact the first seven encodings of Table 2, generalized to d colors for arbitrary d, were used to test the colorability of dozens of graphs by S. Prestwich in *LNCS* **2919** (2004), 105–119, using Algorithm 7.2.2.2W (WalkSAT) as the solver.]

Fig. 113. *The Sierpiński gasket graph* $S_n^{(3)}$, shown here for $n = 4$, is created by pasting together the corners of 3^{n-1} triangles in an interesting way. Each triangle has a ternary label $\alpha = a_1 \ldots a_{n-1}$, and its corners are labeled $\alpha 0$ (top), $\alpha 1$ (lower left), $\alpha 2$ (lower right). Every vertex whose label has the form $\alpha = a_1 \ldots a_{k-1}a_ka_n \ldots a_n$, so that $a_k \neq a_{k+1} = \cdots = a_n$, is pasted together with the vertex labeled $\alpha' = a_1 \ldots a_{k-1}a_na_k \ldots a_k$. This rule gives two labels to all vertices, except for $\{0 \ldots 0, 1 \ldots 1, 2 \ldots 2\}$; hence there are $(3^n + 3)/2$ distinct vertices altogether.

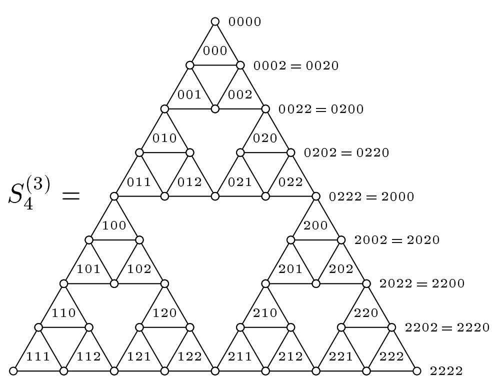


Figure 113 illustrates a family of graphs for which 3-coloring is particularly instructive. [See W. Sierpinski, *Comptes Rendus Acad. Sci.* **160** (Paris, 1915),

multivalued encoding
log encoding
binary encoding
support encoding
weakened encoding
prefix encoding
order encoding
unary encoding
coloring a graph
Prestwich
WalkSAT
Sierpiński gasket graph–
3-coloring
Sierpinski

302–305.] The reader will have no trouble coloring the vertices of $S_4^{(3)}$ with three colors; but the interesting thing is that this coloring is essentially *unique*! Indeed, vertices u and v must have the same color whenever u and v lie on the same vertical line, or on any diagonal whose slope is $\pm 30°$ (see exercise 221).

Computer programmers have little difficulty verifying the uniqueness, in their heads; but it's a different story for computers themselves. Suppose, for example, that the machine has found a way to color the lower-right third of Fig. 113. Then there are two legal colors for vertices 0202 and 0212 (whose other names are 0220 and 0221). One of those colors is correct; but the other one leads to a dead end, which the machine might not discover for a long, long time. If a conventional backtrack search is used, the running time needed to color $S_{n+1}^{(3)}$ will actually be about $3+\sqrt{5} \approx 5.24$ times as long as the time that's needed for $S_n^{(3)}$. (In fact, exercise 226 shows that Fibonacci numbers have a surprising connection to this problem.)

The corners of a Sierpiński gasket graph have different colors in any 3-coloring. Let's therefore define the *pinched Sierpiński gasket graph* $\widehat{S}_n^{(3)}$ to be the same as $S_n^{(3)}$ but with the corner vertices $0\ldots0$ and $1\ldots1$ pasted together. This graph *cannot* be 3-colored. (Notice that $\widehat{S}_n^{(3)}$ has $\lceil 3^n/2 \rceil$ vertices, each of which has degree 4 except for the remaining corner vertex $2\ldots2$; see page xiv.)

One way to compare the encodings of Table 2 is to see how long it takes for a SAT solver to prove the unsatisfiability of the clauses produced from $\widehat{S}_n^{(3)}$, with each encoding. We might save a factor of six if we introduce clauses to force the colors of the top three vertices $0\ldots00$, $0\ldots01$ and $0\ldots02$ (see exercise 224).

Detailed statistics are reported in exercise 225, and the bottom line is that

$$\text{Log} \approx \text{Reduced} < \text{Prefix} \approx \text{Direct} \approx \text{Multi} \approx \text{Support} < \text{Weakened} \ll \text{Binary},$$

at least with respect to this 3-coloring problem. For example, the running times in gigamems, when Algorithm 7.2.2.2C was applied to the clauses for $\widehat{S}_9^{(3)}$, were Log (8.1), Reduced (8.6), Prefix (11.2), Direct (12.0), Multivalued (13.1), Support (13.3), Weakened (27.0), Binary (338.0), showing the median of nine runs in each case. (The binary encoding is *terrible*; we won't discuss it further.)

We can actually do better, however, because the graph $\widehat{S}_n^{(3)}$ contains lots of triangles (3-cliques); and that means we can give *clique hints* to the SAT solver. For example, whenever $u — v — w — u$ is a 3-clique in a graph that we want to 3-color, we can include the clauses

$$(u_0 \vee v_0 \vee w_0) \;\wedge\; (u_1 \vee v_1 \vee w_1) \;\wedge\; (u_2 \vee v_2 \vee w_2) \tag{69}$$

when we're using the direct encoding, multivalued encoding, or support encoding, because each color must appear on one of those vertices. The other encodings also have appropriate clique hints (see exercise 228). So the running times for $\widehat{S}_9^{(3)}$ go down: Prefix (4.8), Log (5.8), Reduced (6.5), Multivalued (7.5), Direct (7.9), Support (9.6), Weakened (39.2). The prefix encoding has jumped into the lead!

Let's take a look under the hood, in order to understand a bit of what's going on. The SAT solver used in these experiments, Algorithm 7.2.2.2C, gets much of its prowess from its ability to learn new clauses, as it tries random possibilities

unique
Fibonacci numbers
pinched Sierpiński gasket graph
triangles

and notices the reasons for contradictions. For example, in one attempt when given the small example $\widehat{S}_4^{(3)}$ of Fig. 113 (but pinched), the first thing that it learned after inputting the prefix-encoded clauses was

$$(\overline{0202}_2 \vee 0122_2). \qquad (70)$$

It means, "if vertex 0202 has color 2, so does vertex 0122." Can you guess why? The machine tried to assume the truth of 0202_2; and that implies both $\overline{0212}_2$ and $\overline{0201}_2$; but the clique hint $(0212_2 \vee 0122_2 \vee 0201_2)$ then implies 0122_2.

Exercise 229 discusses the machine's next discovery, which was the clause '$(0202_2 \vee 0222_2)$'. Its eighth major deduction was '$(0112)_2$'; and after learning 21 clauses it was ready to deduce the empty clause, namely unsatisfiability.

Thus the magic of Boolean algebra allows a SAT solver to pursue lines of reasoning that go well beyond anything that a conventional backtracking approach would ever contemplate. But when we look at the running times by which the prefix encoding verifies uncolorability, our hopes are actually dashed:

$\widehat{S}_3^{(3)}$	$\widehat{S}_4^{(3)}$	$\widehat{S}_5^{(3)}$	$\widehat{S}_6^{(3)}$	$\widehat{S}_7^{(3)}$	$\widehat{S}_8^{(3)}$	$\widehat{S}_9^{(3)}$	$\widehat{S}_{10}^{(3)}$	$\widehat{S}_{11}^{(3)}$
1.36 Kμ	27.6 Kμ	345 Kμ	2.98 Mμ	23.0 Mμ	299 Mμ	4.77 Gμ	72.9 Gμ	1460 Gμ

This is the best of our SAT-oriented methods for $\widehat{S}_n^{(3)}$; yet when n increases by 1, its running time eventually grows by a factor exceeding 15. That's *much* worse than the factor of $3+\sqrt{5} \approx 5.236$, which we know from exercise 226 is achievable by simple backtracking! Indeed, Algorithm 7.2.2.1X is able to handle the case $n = 11$ in only 2.34 Gμ (see exercise 230), more than 600 times faster.

All is not lost, however. Algorithm 7.2.2.2C has ten tunable parameters, and the running times above were all obtained with the default settings shown in 7.2.2.2–(194). But a quite different set of parameters, 7.2.2.2–(196), is known to work much better with problems of the form *waerden*$(3, k; n)$. Filip Stappers has discovered that a similar phenomenon occurs for the pinched gasket benchmarks: He used ParamILS on small cases to obtain the somewhat eccentric settings

$$\alpha = 0.6,\ \ \rho = 0.6,\ \ \varrho = 0.99,\ \ \Delta_{\rm p} = 10000,\ \ \delta_{\rm p} = 5000,$$
$$\tau = 20,\ \ w = 1,\ \ p = 0.02,\ \ P = 0,\ \ \psi = 0.15. \qquad (71)$$

Those parameters make the algorithm run dramatically faster as n grows:

$\widehat{S}_3^{(3)}$	$\widehat{S}_4^{(3)}$	$\widehat{S}_5^{(3)}$	$\widehat{S}_6^{(3)}$	$\widehat{S}_7^{(3)}$	$\widehat{S}_8^{(3)}$	$\widehat{S}_9^{(3)}$	$\widehat{S}_{10}^{(3)}$	$\widehat{S}_{11}^{(3)}$
1.84 Kμ	49.0 Kμ	583 Kμ	2.84 Mμ	18.0 Mμ	90.8 Mμ	521 Mμ	2.27 Gμ	13.2 Gμ

And indeed the ratio for $\widehat{S}_{n+1}^{(3)}/\widehat{S}_n^{(3)}$ is now close to $3+\sqrt{5}$, as when backtracking.

The fact that $\widehat{S}_{11}^{(3)}$ can be proved 3-uncolorable in only 13 Gμ is quite impressive, considering that it's a problem with $3^{11} + 1 = 177148$ Boolean variables and $4 \cdot 3^{11} + 6 = 708594$ clauses! As the author was conducting these experiments in 2022, he considered also Armin Biere's "Kissat," one of the world's best contemporary solvers. Kissat, which is the fruit of a decade's further research since Section 7.2.2.2 was written, is more than twice as fast as the best solvers of 2012, on a majority of difficult problems. Kissat tunes its

empty clause
Boolean algebra
parameters, tuning
waerden
Stappers
ParamILS
author
Biere
Kissat

own internal parameters; and its running time when applied to $\widehat{S}_n^{(3)}$ turns out to have the same order of growth, $(3+\sqrt{5})^n$. (See exercise 246.) It appears that this kind of machine learning cannot break through that asymptotic barrier.

Recall that we did see, way back in Fig. 92 when Algorithm 7.2.2.2C was originally defined, that SAT technology does dramatically speed up similar proofs with respect to *another* family of graphs. In that problem, which deals with the "flower snark line graphs" $L(J_q)$, the graphs in question have only $6q$ vertices and $12q$ edges, so they lead to far fewer Boolean variables. Those graphs aren't 3-colorable when q is odd; so they give us lots more cases on which we can compare the effectiveness of different SAT encodings. Let's therefore pursue the exploration of flower snarks by extending the results reported in Fig. 92.

Exercise 7.2.2.2–176(c) defines clauses called $\mathit{fsnark}(q)$, which represent the multivalued encoding for the problem of 3-coloring the graph $L(J_q)$. We know now, however, that we can improve those clauses by also including clique hints. (Indeed, the $12q$ edges of $L(J_q)$ arise from $4q$ 3-cliques, because J_q is a cubic graph.) Furthermore we can of course consider the same problem with respect to the other encodings in Table 2. Exercise 232 shows that when $q = 99$ the respective running times, in megamems, are Log (240), Reduced (305), Prefix (339), Weakened (402), Direct (448), Multivalued (520), Support (1091).

Surprise: Those *aren't* the rankings that our experience with pinched gaskets has led us to expect, although both coloring problems seem to be quite similar.

A second surprise awaits us when we study the running times for larger and larger q. According to Fig. 92, those times grow linearly for $q \le 99$; thus if we change q to $2q+1$ we should expect the proof of unsatisfiability to take about twice as long. That's not what happens, however. Considering only the log encoding, which appears to be best for these graphs, we find

$L(J_{99})$	$L(J_{199})$	$L(J_{399})$	$L(J_{799})$	$L(J_{1599})$	$L(J_{3199})$	$L(J_{6399})$
$249\,\mathrm{M}\mu$	$1.10\,\mathrm{G}\mu$	$4.66\,\mathrm{G}\mu$	$21.2\,\mathrm{G}\mu$	$48.2\,\mathrm{G}\mu$	$171.7\,\mathrm{G}\mu$	$630\,\mathrm{G}\mu$

which is roughly quadratic behavior. The reasons are by no means clear, nor is much known about the effect of adapting Algorithm 7.2.2.2C's parameters $(\alpha, \rho, \varrho, \ldots, \psi)$ to the various encodings. SAT solvers are full of surprises!

So far we've been looking only at ternary domains. Domains of size 4 or more lead of course to many further questions, with seemingly endless possibilities to explore. The encodings for $d = 3$ in Table 2 can be extended to arbitrary d in interesting ways (see exercise 240). And the graphs $S_n^{(3)}$ can also be extended to *Sierpiński simplex graphs* $S_n^{(d)}$ for arbitrary d; the case $d = 4$ and $n = 3$ is illustrated in Fig. 114. When $d = 4$, $S_n^{(d)}$ is called the Sierpiński *tetrahedron* graph of order n. It was actually invented by Alexander Graham Bell [*National Geographic Magazine* **14**, 6 (June 1903), 219–251], in connection with kite designs!

Notice that $S_n^{(d)}$ is essentially a $(d-1)$-dimensional object. That makes it a bit of a challenge (but fun) to imagine when $d > 4$.

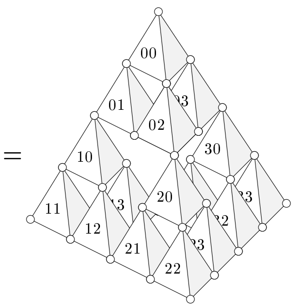
We can obtain a *pinched* version $\widehat{S}_n^{(d)}$ by pasting vertices $0\ldots0$ and $1\ldots1$ together as we did before. Exercise 239 points out that the graph $\widehat{S}_n^{(d)}$ cannot be

flower snark line graphs
line graphs
$\mathit{fsnark}(q)$
cubic graph
log encoding
parameters
Sierpiński simplex graphs
simplex graphs
cliques
pure vertices
pasting graphs together
Sierpiński *tetrahedron* graph
Bell
kite designs
Sierpiński [sub] triangle graph see gasket
pinched

augmented Sierpinski simplex gr
prefix encoding
order encoding
Hints for d-cliques

Fig. 114. The graph $S_n^{(d)}$ is obtained by pasting together d^{n-1} cliques of size d, using the same rules that were specified for $d = 3$ in Fig. 113: Each d-clique has a d-ary label $\alpha = a_1 \ldots a_{n-1}$, and its "corner vertices" are labeled αj for $0 \le j < d$. Each vertex has two d-ary labels α and α' as before, except for the d "pure" vertices labeled $j \ldots j$; exercise 234 gives examples. Therefore there are $(d^n + d)/2$ vertices altogether.

$S_3^{(4)} =$



d-colored, when d is odd; but the situation is quite different when d is even. For example, it's easy to 4-color the graph in Fig. 114 by putting the same color at each of the four corners. Yet we can't 4-color it with all-*different* corner colors.

Let's therefore define the *augmented* Sierpiński simplex graph to be

$$\overline{S}_n^{(d)} \;=\; S_n^{(d)} \text{ plus } d-1 \text{ edges } 0\ldots0 \text{ — } j\ldots j \text{ for } 0 < j < d. \qquad (72)$$

This graph cannot be d-colored when $n > 1$ and d is even.

As we saw when $d = 3$, instructive results are obtained when we experiment with various SAT encodings to verify the d-uncolorability of $\widehat{S}_n^{(d)}$ for odd domain sizes d, and of $\overline{S}_n^{(d)}$ for even domain sizes. The principal contenders when $d = 4$ are the direct, multivalued, log, support, weakened, reduced, and order encodings. (See Table 2 and exercise 240; prefix encoding is the same as log encoding when $d = 4$, and order encoding becomes distinct from the others when $d > 3$.) Hints for d-cliques, discussed in exercise 241, prove to be enormously beneficial.

Detailed statistics for $d = 4$ and $n \le 7$ show that for these problems we have

$$\text{Direct} \approx \text{Multi} \approx \text{Ordered} < \text{Reduced} < \text{Support} < \text{Log} \ll \text{Weakened},$$

roughly speaking, as reported in exercise 244. The best results overall, obtained with the direct encoding, make those relative rankings quantitative:

$\overline{S}_3^{(4)}$	$\overline{S}_4^{(4)}$	$\overline{S}_5^{(4)}$	$\overline{S}_6^{(4)}$	$\overline{S}_7^{(4)}$
57.8 Kμ	1.99 Mμ	23.8 Mμ	1.16 Gμ	135 Gμ

(possibly indicating superexponential growth in the running time as n increases).

That's great news: Those running times are a huge win for SAT-based methods — because the $\overline{S}_n^{(4)}$ problem has a much, much larger search space than the $\widehat{S}_n^{(3)}$ problem does. For example, its backtrack tree appears to have about 10^{13} nodes already when $n = 4$, and more than 10^{50} when $n = 5$. The methods that we used to beat SAT in the two-dimensional case are now hopelessly inadequate.

Moving on to domains of size $d = 5$, again there are surprises (see exercise 245). The log encoding now becomes totally outclassed, and the new champion is the *reduced* encoding! Typical running times for the latter are

$\widehat{S}_3^{(5)}$	$\widehat{S}_4^{(5)}$	$\widehat{S}_5^{(5)}$	$\widehat{S}_6^{(5)}$
156 Mμ	1.78 Gμ	17.9 Gμ	172 Gμ

although the backtrack tree for $\widehat{S}_4^{(5)}$ has $\approx 10^{17}$ nodes. These are tough problems.

SAT encodings of general relations. We've now seen a variety of Boolean representations of d-ary domains; but we've looked at only one constraint, '$\neq$'.

The next most important way to constrain two variables u and v is probably the relation '$u \le v$'—or perhaps '$u < v$', which is the same as '$u \le v - 1$'. The order encoding is particularly good for constraints such as this. Indeed, in the d-ary domain $\{0, 1, \ldots, d-1\}$, with the Boolean variable u^j standing for $[u \ge j]$, the relation '$u \le v - t$' for any fixed t is equivalent to the clauses

$$\bigwedge_{0\le j\le d-t} (\bar u^j \vee v^{j+t}), \quad \text{if } t > 0; \qquad \bigwedge_{-t<j<d} (\bar u^j \vee v^{j+t}), \quad \text{if } t \le 0. \tag{73}$$

(We omit $\bar u^0$ or v^d if they are present.) In the case $t = 1$, for example, we get

$$\text{`}u < v\text{'} \iff (v^1) \wedge (\bar u^1 \vee v^2) \wedge (\bar u^2 \vee v^3) \wedge \cdots \wedge (\bar u^{d-2} \vee v^{d-1}) \wedge (\bar u^{d-1}). \tag{74}$$

And we can even go much further: Exercises 7.2.2.2–405 and 406 give encodings for '$au + bv \le c$' as well as '$uv \le a$' and '$uv \ge a$', for any constants a, b, c, using only clauses that belong to 2SAT. Exercise 7.2.2.2–407 gives a 3SAT equivalent of the *ternary* relation '$u + v \le w$', when all three variables are order-encoded.

There's also a good way to translate the relation '$u \le v$' into SAT clauses when u and v have the log encoding, thanks to Eq. 7.2.2.2–(169). For example, suppose $d = 16$, $u = (u_8u_4u_2u_1)_2$, and $v = (v_8v_4v_2v_1)_2$, using four bits to represent each variable. Then we have $u \le v$ if and only if

$$(\bar u_8 \vee v_8) \wedge (\bar u_8 \vee a_1) \wedge (v_8 \vee a_1) \wedge (\bar a_1 \vee \bar u_4 \vee v_4) \wedge (\bar a_1 \vee \bar u_4 \vee a_2) \wedge (\bar a_1 \vee v_4 \vee a_2) \wedge$$
$$(\bar a_2 \vee \bar u_2 \vee v_2) \wedge (\bar a_2 \vee \bar u_2 \vee a_3) \wedge (\bar a_2 \vee v_2 \vee a_3) \wedge (\bar a_3 \vee \bar u_1 \vee v_1). \tag{75}$$

Notice that these clauses introduce *auxiliary variables* a_k; such variables must not be used in the encoding of any other constraint. (For instance, if we also require $v \le w$, we'd need to introduce auxiliaries called a_4, a_5, and a_6, say.) Exercise 248 shows that a similar scheme can encode '$u \le v - t$' for any t.

In general, however, a CSP can involve arbitrary constraints that don't have nice properties like the relation '$u \le v - t$'. A so-called "table constraint" is specified by tabulating the pairs (u, v) that satisfy it. (Or by listing the pairs that *don't* satisfy it, if the bad pairs are easier to specify than the good ones.) If we can deal with any table constraint, we can handle any constraint whatsoever.

Table constraints are usually translated into SAT by letting the Boolean variable v_a represent $[v = a]$, for each value a in the domain of each variable v, as we've done in most of the examples above. Here are the most popular schemes:

- *Direct encoding.* Start with the at-least-one and at-most-one clauses for each variable, as in (66) and (67). Then, for each pair of values (a, b) such that the assignments $u = a$ and $v = b$ do *not* satisfy the given relation—a so-called *nogood*—add the "preclusion clause" $(\bar u_a \vee \bar v_b)$, also called a "conflict clause."

(Thus Table 2, which encodes '$u \ne v$' in ternary domains, has three nogoods.)

Notice that the direct encoding works naturally for k-ary constraints as well as for binary constraints: If the values $(a_1, \ldots, a_k)$ don't satisfy a given relation on the variables $(v_1, \ldots, v_k)$, the preclusion clause is $(\bar v_{1a_1} \vee \cdots \vee \bar v_{ka_k})$.

order encoding
2SAT
3SAT
log encoding
auxiliary variables
table constraint
Direct encoding
at-least-one
at-most-one
nogood
preclusion clause
conflict clause, see preclusion

Support encoding
k-ary to binary
hidden variable
hypergraph
bipartite graph
direct encoding
support encoding

- *Support encoding.* Given a binary relation $R(u,v)$, start with the at-least-one and at-most-one clauses as above. Then add the "support clauses"

$$\bigwedge_{a\in D_u}\Bigl(\bar{u}_a\vee\bigvee\{v_b \mid ab\in R(u,v)\}\Bigr)\wedge\bigwedge_{b\in D_v}\Bigl(\bar{v}_b\vee\bigvee\{u_a \mid ab\in R(u,v)\}\Bigr). \quad (76)$$

(The domains are D_u and D_v. In Table 2, $D_u=D_v=[0\,.\,.\,3)$, $R(u,v)=[u\neq v]$.)

The support encoding can also be defined for k-ary relations $R(v_1,\ldots,v_k)$. But in this case we use a trick by which any k-ary relation can be regarded as a set of k *binary* relations $R_j(v_j,R)$; here R_j relates the original variable v_j to a new "hidden variable" R, whose domain D_R is the set $\{a_1\ldots a_k \mid R(a_1,\ldots,a_k)\}$ of all tuples that satisfy R. If $a\in D_{v_j}$ and $a_1\ldots a_k\in D_R$, then we have

$$R_j(a,a_1\ldots a_k)\iff a=a_j,\quad \text{for } 1\le j\le k. \quad (77)$$

(The idea concealed in this daunting notation is basically an elaboration of the way in which we represented a hypergraph as a bipartite graph in 7–(57).)

Let's study a simple example, by considering the case where $R=R(u,v,w)$ is the following more-or-less random ternary relation on ternary variables $\{u,v,w\}$:

$$R(u,v,w)\iff uvw\in\{000,001,010,012,020,121,211\}. \quad (78)$$

The direct encoding for R has $3^3-7=20$ nogoods, because R has seven tuples; so it consists of the at-least-one and at-most-one clauses together with

$$\begin{aligned}
&(\bar{u}_0\vee\bar{v}_0\vee\bar{w}_2)\wedge(\bar{u}_0\vee\bar{v}_1\vee\bar{w}_1)\wedge(\bar{u}_0\vee\bar{v}_2\vee\bar{w}_1)\wedge(\bar{u}_0\vee\bar{v}_2\vee\bar{w}_2)\wedge(\bar{u}_1\vee\bar{v}_0\vee\bar{w}_0)\wedge\\
&(\bar{u}_1\vee\bar{v}_0\vee\bar{w}_1)\wedge(\bar{u}_1\vee\bar{v}_0\vee\bar{w}_2)\wedge(\bar{u}_1\vee\bar{v}_1\vee\bar{w}_0)\wedge(\bar{u}_1\vee\bar{v}_1\vee\bar{w}_1)\wedge(\bar{u}_1\vee\bar{v}_1\vee\bar{w}_2)\wedge\\
&(\bar{u}_1\vee\bar{v}_2\vee\bar{w}_0)\wedge(\bar{u}_1\vee\bar{v}_2\vee\bar{w}_2)\wedge(\bar{u}_2\vee\bar{v}_0\vee\bar{w}_0)\wedge(\bar{u}_2\vee\bar{v}_0\vee\bar{w}_1)\wedge(\bar{u}_2\vee\bar{v}_0\vee\bar{w}_2)\wedge\\
&(\bar{u}_2\vee\bar{v}_1\vee\bar{w}_0)\wedge(\bar{u}_2\vee\bar{v}_1\vee\bar{w}_2)\wedge(\bar{u}_2\vee\bar{v}_2\vee\bar{w}_0)\wedge(\bar{u}_2\vee\bar{v}_2\vee\bar{w}_1)\wedge(\bar{u}_2\vee\bar{v}_2\vee\bar{w}_2).
\end{aligned} \quad (79)$$

The support encoding for R is obtained by combining the support encodings for the three binary relations $R_u(u,R)$, $R_v(v,R)$, and $R_w(w,R)$, namely

$$(R_{000}\vee R_{001}\vee R_{010}\vee R_{012}\vee R_{020}\vee R_{121}\vee R_{211}); \quad (80)$$

$$\begin{array}{ll}
& (\bar{u}_0\vee R_{000}\vee R_{001}\vee R_{010}\vee R_{012}\vee R_{020}),\\
(\overline{R}_{000}\vee u_0)\wedge(\overline{R}_{000}\vee v_0)\wedge(\overline{R}_{000}\vee w_0), & (\bar{u}_1\vee R_{121}),\\
(\overline{R}_{001}\vee u_0)\wedge(\overline{R}_{001}\vee v_0)\wedge(\overline{R}_{001}\vee w_1), & (\bar{u}_2\vee R_{211});\\
(\overline{R}_{010}\vee u_0)\wedge(\overline{R}_{010}\vee v_1)\wedge(\overline{R}_{010}\vee w_0), & (\bar{v}_0\vee R_{000}\vee R_{001}),\\
(\overline{R}_{012}\vee u_0)\wedge(\overline{R}_{012}\vee v_1)\wedge(\overline{R}_{012}\vee w_2), & (\bar{v}_1\vee R_{010}\vee R_{012}\vee R_{211}),\\
(\overline{R}_{020}\vee u_0)\wedge(\overline{R}_{020}\vee v_2)\wedge(\overline{R}_{020}\vee w_0), & (\bar{v}_2\vee R_{020}\vee R_{121});\\
(\overline{R}_{121}\vee u_1)\wedge(\overline{R}_{121}\vee v_2)\wedge(\overline{R}_{121}\vee w_1), & (\bar{w}_0\vee R_{000}\vee R_{010}\vee R_{020}),\\
(\overline{R}_{211}\vee u_2)\wedge(\overline{R}_{211}\vee v_1)\wedge(\overline{R}_{211}\vee w_1); & (\bar{w}_1\vee R_{001}\vee R_{121}\vee R_{211}),\\
& (\bar{w}_2\vee R_{012});
\end{array} \quad (81)$$

$$\begin{aligned}
&(u_0\vee u_1\vee u_2)\wedge(\bar{u}_0\vee\bar{u}_1)\wedge(\bar{u}_0\vee\bar{u}_2)\wedge(\bar{u}_1\vee\bar{u}_2);\\
&(v_0\vee v_1\vee v_2)\wedge(\bar{v}_0\vee\bar{v}_1)\wedge(\bar{v}_0\vee\bar{v}_2)\wedge(\bar{v}_1\vee\bar{v}_2);\\
&(w_0\vee w_1\vee w_2)\wedge(\bar{w}_0\vee\bar{w}_1)\wedge(\bar{w}_0\vee\bar{w}_2)\wedge(\bar{w}_1\vee\bar{w}_2).
\end{aligned} \quad (82)$$

At-most-one clauses for R, such as $(\overline{R}_{000}\vee\overline{R}_{001})$, aren't needed (see exercise 252).

- *Encoded projections.* A k-ary relation can be "projected" onto any subset of its variables, obtaining a weaker relation that must also be true. The conjunction of these weaker relations is an approximation to the overall one.

For example, the ternary relation (78) has three projections onto binary relations:

$$uv \in \{00, 01, 02, 12, 21\}; \tag{83}$$

$$uw \in \{00, 01, 02, 11, 21\}; \tag{84}$$

$$vw \in \{00, 01, 10, 11, 12, 20, 21\}. \tag{85}$$

We need $3+3+1$ preclusion clauses to rule out their inadmissible pairs. That leaves the seven tuples of R, and also 011; so one more preclusion clause,

$$(\bar u_0 \vee \bar v_1 \vee \bar w_1), \tag{86}$$

will give us the equivalent of (79) in the direct encoding.

(In database theory, a relation that's equal to the intersection of some of its projections is said to have a *lossless join dependency* on those projections.)

Exercise 256 shows that about 1.2% of all ternary relations on ternary domains can be decomposed losslessly into their binary projections. The remaining 98.8% are inherently ternary; but nearly half of them are *almost* decomposable, needing only five or fewer additional preclusions such as (86). (See exercise 257.)

Notice that the direct encoding is smallest when there are comparatively few nogood tuples, as we saw in the relation '$u \neq v$'; contrariwise, the support encoding is smallest when there are comparatively few *good* ones. The tradeoff is often tricky. When trying to place n queens, for example, exercise 7.2.2.2–400 concludes that the direct encoding is preferable when trying to find just one solution to that problem, but the support encoding is better for finding all solutions.

We needn't choose a single encoding scheme; the best solution for some applications might be to use two different encodings simultaneously.

Recall from Eq. 7.2.2.2–(180) that some encodings are *forcing*, in the sense that every implied consequence with respect to the individual (nonauxiliary) literals can be found efficiently by a SAT solver using only unit propagations. Furthermore, exercise 7.2.2.2–433 showed that the log encoding in (75) is forcing for the relation '$u \leq v$'. Thus, for example, if $u_4 = u_1 = 1$ and $v_2 = v_1 = 0$, then unit propagation in (75) will force $v_8 = 1$ and $u_8 = 0$.

Forcing clauses are obviously desirable, if they don't take up too much space. The direct encoding usually doesn't have the forcing property; for instance, if we assert $u_0 = 0$ in (79), unit propagation does nothing. By contrast, however, asserting $u_0 = 0$ in (81) immediately implies $\overline{R}_{000}$, $\overline{R}_{001}$, $\overline{R}_{010}$, $\overline{R}_{012}$, $\overline{R}_{020}$, $\bar v_0$, $\bar w_0$, $\bar w_2$; hence w_1, by (82). The good news is that *the support encoding is always forcing.* (See exercise 260; we can regard variables R_{000}, R_{001}, … as auxiliary.)

Consistency. Let's shift gears now and turn to CSP-solving techniques that go beyond what we've previously learned about XCC-solving and SAT-solving. One of the key concepts is the notion of "consistency," championed by Alan K. Mackworth in *Artificial Intelligence* **8** (1977), 99–118, and extended by many others. In general, we want to avoid or ameliorate the need to backtrack, by recognizing as early as possible when our current line of search is doomed to fail.

projections
lossless join dependency
join dependency
n queens
queens
forcing
unit propagations
consistency–
Mackworth

A set of constraints is "inconsistent" if and only if it has no solution; and that's a coNP-complete problem. So we can't expect to solve it efficiently. Yet it makes sense to strive for subproblems that are not easily *proved* to be inconsistent. We can in fact distinguish many degrees of consistency, increasingly difficult to check but more and more effective in pruning the search tree. Interesting and important tradeoffs arise as we try to balance the cost of consistency testing with the number of cases to be examined.

coNP-complete
domain filtering–
Horn clauses
dual Horn clauses
Horn core
support clauses

Consider, for example, the CSP that has four variables $\{w, x, y, z\}$, each with the ternary domain $\{0, 1, 2\}$, subject to the following six constraints:

$$wx \neq 22; \quad wy \notin \{10, 20\}; \quad wz \neq 02; \quad xy \notin \{11, 12, 22\};$$
$$xz \notin \{00, 02\}; \quad yz \notin \{00, 01, 10, 11, 21\}. \tag{87}$$

Instead of trying the 81 possibilities for $wxyz$, we can start by observing that $z \neq 1$, by propagating the yz constraint. Therefore $x \neq 0$, by propagating the xz constraint. Hence $y \neq 2$, by propagating the xy constraint. The yz constraint now tells us that $z \neq 0$; hence $z = 2$. Therefore $w \neq 0$; and the wy constraint tells us that $y \neq 0$. Consequently $x \neq 1$, $w \neq 2$; the unique solution is $wxyz = 1212$! This process is called *domain filtering*.

One way to understand such propagations is to set up a system of definite Horn clauses from the given constraints (87):

$$\begin{array}{rrrr}
\bar w_0 \wedge \bar w_1 \Rightarrow \bar x_2 & \bar w_1 \wedge \bar w_2 \Rightarrow \bar z_2 & \bar x_0 \wedge \bar x_2 \Rightarrow \bar y_1 & \bar y_2 \Rightarrow \bar z_0 \\
\bar x_0 \wedge \bar x_1 \Rightarrow \bar w_2 & \bar z_0 \wedge \bar z_1 \Rightarrow \bar w_0 & \bar x_0 \Rightarrow \bar y_2 & \Rightarrow \bar z_1 \\
\bar w_0 \Rightarrow \bar y_0 & \bar x_1 \wedge \bar x_2 \Rightarrow \bar z_0 & \bar y_0 \Rightarrow \bar x_1 & \bar z_2 \Rightarrow \bar y_0 \\
\bar y_1 \wedge \bar y_2 \Rightarrow \bar w_1 & \bar x_1 \wedge \bar x_2 \Rightarrow \bar z_2 & \bar y_0 \wedge \bar y_1 \Rightarrow \bar x_2 & \bar z_2 \Rightarrow \bar y_1 \\
\bar y_1 \wedge \bar y_2 \Rightarrow \bar w_2 & \bar z_1 \Rightarrow \bar x_0 & & \bar z_0 \wedge \bar z_2 \Rightarrow \bar y_2
\end{array} \tag{88}$$

(Well, these are actually *dual* Horn clauses, because every variable is complemented.) For example, if $w \neq 0$ and $w \neq 1$, then $x \neq 2$, because $wx \neq 22$. In these terms, the filtering process is identical to Algorithm 7.1.1C, the computation of the "Horn core"; and we know from that algorithm that the computation can be done efficiently, using simple data structures.

The rule to get from (87) to (88) is that, when $R(u, v)$ is a relation between variables u and v whose domains are D_u and D_v, we have $|D_u| + |D_v|$ clauses,

$$\bigwedge\{\bar v_b \mid ab \in R(u,v)\} \Rightarrow \bar u_a,\ a \in D_u; \quad \bigwedge\{\bar u_a \mid ab \in R(u,v)\} \Rightarrow \bar v_b,\ b \in D_v; \tag{89}$$

however, the clauses for $\bar u_a$ and $\bar v_b$ are omitted when the premises are false; such clauses are trivially true. (For example, (88) doesn't include '$\bar w_0 \wedge \bar w_1 \wedge \bar w_2 \Rightarrow \bar x_0$', because we can't have $\bar w_0 \wedge \bar w_1 \wedge \bar w_2$. The wx relation needs only two clauses.)

A wide-awake reader might be thinking at this point that (89) looks familiar. And indeed, (89) describes precisely the *support clauses* that were defined above in (76). For instance, '$\bar w_0 \wedge \bar w_1 \Rightarrow \bar x_2$' is the same as the support clause '$\bar x_2 \vee w_0 \vee w_1$', when it's written in CNF. The only difference is that we omit a support clause such as '$\bar x_0 \vee w_0 \vee w_1 \vee w_2$', because it's subsumed by the at-least-one clause '$w_0 \vee w_1 \vee w_2$'. The fact that support clauses are dual Horn clauses explains why SAT solvers handle them efficiently.

The process of solving a CSP can be viewed recursively as a sequence of steps in which we narrow the domains by propagating constraints. Each node α of the search tree corresponds to a set $\mathcal{D}_\alpha = \{D_{\alpha,v} \mid v \text{ is a variable}\}$, where $D_{\alpha,v}$ is the "current domain" of v when we're working on subproblem α. At the root node, o, each domain $D_{o,v}$ is simply v's initially given domain. If some v has $D_{\alpha,v} = \emptyset$, subproblem α has no solution; recursion stops. Otherwise, when node α' is a child of node α, the elements $D_{\alpha',v}$ of $\mathcal{D}_{\alpha'}$ all satisfy $D_{\alpha',v} \subseteq D_{\alpha,v}$.

A new level is entered when we break a problem into subproblems, each of which is a "branch" in the search tree. Two main branching strategies are used, in order to ensure that we find every solution exactly once; both of those strategies involve a variable called the *branch variable*, v, whose domain is being split: (i) A *d-way branch* can be made when v's domain $D_{\alpha,v}$ has d elements, $\{a_1, \ldots, a_d\}$. Then node α has d children, α_j for $1 \le j \le d$, and we have $D_{\alpha_j,v} = \{a_j\}$. We say that v *has been assigned the value* a_j in branch α_j. (ii) A *binary branch on* $v{=}a$ can be made whenever $a \in D_{\alpha,v}$. Then node α has two children, $\alpha_=$ and $\alpha_{\ne}$, and we have $D_{\alpha_=,v} = \{a\}$, $D_{\alpha_{\ne},v} = D_{\alpha,v} \setminus a$. In branch $\alpha_=$ of a binary branch we say, as in (i), that v has been assigned the value a.

If an assignment to one variable somehow forces other variables to have domain size 1, we can optionally regard those variables as all being assigned simultaneously. Similarly, if one or more variables in the right child $\alpha_{\ne}$ of a binary branch happen to have domain size 1, we can optionally call them assigned. However, there's an important restriction: *Whenever values have been assigned to all of the variables in a constraint, those values must satisfy the constraint.*

A variable that has been assigned a value in node α or in any ancestor of that node is said to be *inactive*, because we've already decided its fate. All other variables are *active*; they're the variables of subproblem α that we still need to deal with. A branch variable must always be active in its node.

The restriction that we've imposed on assigned variables makes it clear when we've found a solution: *Node α solves the given* CSP *if and only if it has no active variables*, that is, if and only if a value from its original domain has been assigned to every variable. After reaching a solution node, we backtrack and try for more.

Any pair $\beta = (v, a)$, where v is a variable and a belongs to v's domain, is called a *binding*. If the value a also happens to have been "assigned" to v, in the sense just described, β is also called an *assignment*.

Definition V. *The binding (v, a) is said to be "viable" in subproblem α when every constraint involving v contains at least one tuple τ such that (i) $v = a$ in τ, and (ii) every other variable v' in that constraint has a value $v' = a'$ in τ for which a' belongs to the current domain $D_{\alpha,v'}$ of v'. It's "weakly viable" when it is viable with respect to the constraints in which v is the only active variable.* ▮

Notice that if the binding (v, a) is *not* viable, no solution to subproblem α can have $v = a$. Hence we can safely *remove* a from v's current domain in such a case.

Armed with these definitions, we're now ready to discuss the two most important kinds of consistency, namely "forward consistency" (FC) and "domain consistency" (DC).

recursively
current domain
branch variable
assigned
binary branch
instantiation, see assignment
inactive
active
binding
viable
weakly viable
forward consistency–
domain consistency–

wipe out
supporting tuple
8 queens problem

• *Forward consistency* holds at node α if and only if every active binding is weakly viable. In other words, whenever a constraint contains only one active variable, the domain of that variable is limited to values that satisfy that constraint, together with the values already assigned to the other variables.

• *Domain consistency* holds at node α if and only if every active binding is viable. In other words, no active variable v has a value a in its domain for which the assignment $v = a$ would "wipe out" (reduce to $\emptyset$) the domain of any other active variable. In other words, every binding (v, a) of an active variable v has a supporting tuple in every constraint that involves v. In other words, domain filtering as in (88) doesn't change any domains. (See exercise 263.)

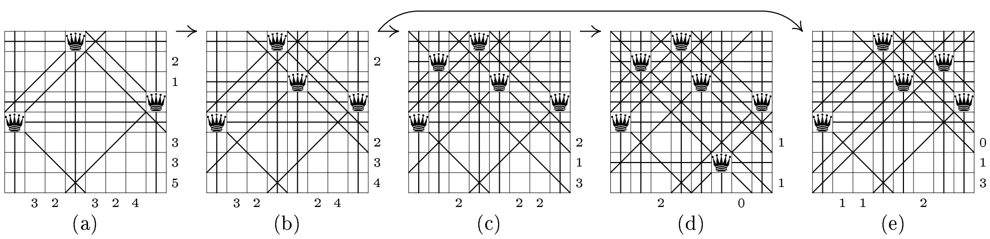


Fig. 115. If we try to extend placement (a) of three queens to eight nonattacking queens, using forward consistency, the task is found to be impossible after we've branched to subproblems (b), (c), (d), (e). (Row and column domain sizes are shown.)

Figures 115 and 116 illustrate the difference between FC and DC in a special case of the 8 queens problem, which we consider to be a CSP with 16 variables $\{r_1, \ldots, r_8, c_1, \ldots, c_8\}$. A queen that has been placed in row i and column j corresponds to having $r_i = j$ and $c_j = i$, as in 7.2.2.1–(23). In position (a), the active variables are $\{r_2, r_3, r_6, r_7, r_8, c_2, c_3, c_5, c_6, c_7\}$ and their forward-consistent domain sizes are respectively $\{2, 1, 3, 3, 5, 3, 2, 3, 2, 4\}$. We're forced to place a queen in row 3, column 5, giving (b); then we branch on two ways to occupy row 2, etc.

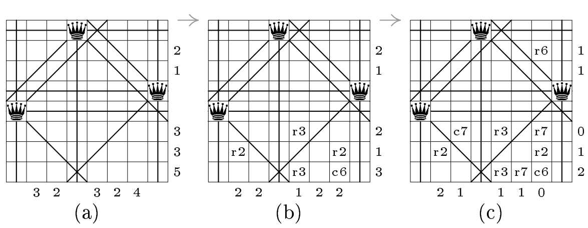


Fig. 116. When domain consistency is applied to the problem of Fig. 115, impossibility is detected before any branching is needed.

On the other hand, domain consistency takes another tack: Subproblem (a) of Fig. 116 isn't domain consistent, because (for example) the binding $(r_6, 5)$ would wipe out r_3. Also $(r_8, 7)$ would wipe out c_6, etc. This domain filtering takes us to (c), which filters out four more bindings and wipes out c_7, etc.

It's easy to maintain forward consistency at every node of a search tree, because an assignment '$v{=}a$' asks us to look only at constraints of the special form $R(v, v_1, \ldots, v_k, w)$, for $k \geq 0$, where w is currently active but we've already assigned $v_1{=}a_1, \ldots, v_k{=}a_k$. Whenever R is such a constraint, we simply restrict the domain of w to values for which $a a_1 \ldots a_k w \in R(v, v_1, \ldots, v_k, w)$.

Domain consistency is harder to maintain, because constraints that don't directly involve a newly assigned variable can also come into play. Whenever an active variable w loses an element b from its domain, there may be one or more active bindings (w', b') that were supported in some constraint by a tuple with $w = b$. All such supports must be replaced by tuples that have $w \neq b$; and if no such tuples exist, we must remove b' from the domain of w'. And so on.

Let's return, for example, to the CSP of (21) and (22), the line labeling problem for the histoscape of Fig. 101(d). It has 26 variables, each with domain $\{\texttt{+}, \texttt{-}, \texttt{>}, \texttt{<}\}$; hence it begins with $26 \cdot 4 = 104$ active bindings, namely $(\mathrm{ab}, \texttt{+})$ through $(\mathrm{rs}, \texttt{<})$. And those variables are subject to 6 binary constraints and 13 ternary constraints.

Forward consistency holds trivially, when we start, because no assignments have yet been made. But domain consistency fails spectacularly; for example, the no-brainer constraint '$(\mathrm{kn}, \mathrm{np}) \in \{\texttt{<<}\}$' supports only one value in the domains of variables kn and np. Thus we can immediately shrink D_{kn} and D_{np} to $\{\texttt{<}\}$.

Furthermore the ternary constraint '$(\mathrm{be}, \mathrm{bd}, \mathrm{ab}) \in \{\texttt{>+>}, \texttt{-+-}, \texttt{+-+}\}$' allows us to shrink D_{bd} to $\{\texttt{+}, \texttt{-}\}$, while D_{be} and D_{ab} become $\{\texttt{+}, \texttt{-}, \texttt{>}\}$. Similar reductions occur for every constraint that arose from a W junction in (20).

Constraints related to V and Y junctions don't help us immediately. But once we know that ab cannot be <, the constraint on $(\mathrm{ab}, \mathrm{ac})$ tells us that ac cannot be +. There are propagations galore! This is worth investigating further.

The *scope* of a constraint is defined to be the set of variables that it constrains. Domain reduction relies on a basic operation that can be formalized as follows, when c is a constraint and v is a variable in c's scope:

$$\mathrm{revise}(c, v) = \begin{cases} \text{For each } a \in D_v, \\ \quad \text{if } c \text{ contains no tuple having } v = a \text{, and having} \\ \qquad \text{all other variables in their current domains,} \\ \quad \text{set } D_v \leftarrow D_v \setminus a. \end{cases} \tag{90}$$

Thus we remove values from v's domain if they aren't viable with respect to c.

We can reach domain consistency if we keep applying revise(c, v) to all possible combinations of c and v, until no more changes occur. But such blind meandering will repeat a lot of unnecessary tests, so we'd like to be a bit more clever. Algorithm D below is one fairly simple yet general way to proceed.

Besides the notation D_v for the domain of variable v, we shall write S_c for the scope of constraint c. Algorithm D gives "time stamps" $\texttt{STAMP}(v)$ and $\texttt{STAMP}(c)$ to each variable and each constraint. Variables also have a field $\texttt{INQ}(v)$, to tell whether or not they're in the queue Q.

Algorithm D (*Domain filtering*). Given a CSP, this algorithm attempts to reduce the domains of variables without reducing the number of solutions. It terminates with wipeout if some domain becomes empty; otherwise it terminates with domain consistency. Values may already have been assigned to some of the variables, in which case we assume that all constraints without active variables have been satisfied. An auxiliary queue, Q, holds a set of active variables that need to be examined or reexamined.

line labeling problem
histoscape
scope
notation D_v
S_c
time stamps
stamps

D1. [Initialize.] Set $\mathtt{STAMP}(c) \leftarrow 0$ for each constraint c; $\mathtt{STAMP}(v) \leftarrow \mathtt{INQ}(v) \leftarrow [v \text{ is active}]$ for each variable v; $t \leftarrow 1$. Put all the active variables into Q.

D2. [Queue empty?] Terminate successfully if Q is empty. Otherwise set $v \Leftarrow Q$ (deleting the front of Q) and $\mathtt{INQ}(v) \leftarrow 0$.

D3. [Loop over constraints.] Do steps D4–D5 for every constraint c for which $v \in S_c$ and $\mathtt{STAMP}(v) > \mathtt{STAMP}(c)$. Then return to D2.

D4. [Loop over variables.] Do step D6 for every active variable $w \in S_c$, including $w = v$ (see exercise 267).

D5. [Certify c.] Set $\mathtt{STAMP}(c) \leftarrow t$ and $t \leftarrow t + 1$, then return to D3.

D6. [Revise w.] Do revise(c, w) (see (90)). If that routine doesn't change D_w, do nothing more. Otherwise, if $D_w = \emptyset$, terminate with wipeout. Otherwise set $\mathtt{STAMP}(w) \leftarrow t$; and if $\mathtt{INQ}(w) = 0$, also set $Q \Leftarrow w$ and $\mathtt{INQ}(w) \leftarrow 1$. ▮

The elements $v \in Q$ whose domain has changed since c was certified are precisely those with $\mathtt{STAMP}(v) > \mathtt{STAMP}(c)$. The time stamps therefore help us to avoid calling 'revise' unnecessarily.

But a more fine-grained analysis shows that Algorithm D might still make many redundant tests. Exercises 268 and 269 show how to achieve domain consistency with a near-minimum amount of work, using the Horn core algorithm.

In any case, after the dust has settled, the domains for the histoscape constraints (22) will have been reduced to the following, regardless of what algorithm was used to achieve domain consistency:

$$\begin{gathered}\mathrm{bd} = \mathrm{cd} = \mathrm{dg} = \mathrm{fh} = \mathrm{gh} = \mathrm{hj} = \mathrm{jl} = \mathrm{kl} = \mathrm{ls} = \mathrm{op} = \mathtt{+};\\ \mathrm{eg} = \mathrm{ij} = \mathrm{mn} = \mathtt{-};\quad \mathrm{ik} = \mathrm{kn} = \mathrm{np} = \mathtt{<};\\ \mathrm{ab} = \mathrm{be} = \mathrm{ef} = \mathrm{fq} = \mathrm{qs} = \{\mathtt{-},\mathtt{>}\};\quad \mathrm{ac} = \mathrm{cm} = \mathrm{mo} = \mathrm{or} = \mathrm{rs} = \{\mathtt{-},\mathtt{<}\}.\end{gathered} \tag{91}$$

All of the "interior" lines now have a fixed label, while the "boundary" lines each have only two possible labels. Thus the four possible solutions, shown in Fig. 104 and (23), have essentially been discovered by domain filtering alone.

On the other hand, forward consistency does *not* work well on the CSP of (21) and (22), because of the ternary constraints. Those constraints can, however, be converted to binary, by using "hidden variables"; and FC handles those binary constraints very nicely. (See exercises 270, 273, and 274.)

Many other kinds of consistency have been explored by CSP researchers. For example, one fairly easy way to strengthen DC is to require that the hidden variables be domain consistent with respect to each other (see exercise 271). Another straightforward way to prune domains is called *singleton domain consistency*: We can remove a from D_v if the assignment $v{=}a$ gives a subproblem that can't be made DC without emptying another variable's domain (see exercise 275).

Path consistency goes even further and takes propagation to a new level: It introduces new binary constraints, whereas domain filtering simply introduces new unary constraints. If u, v, w are any variables for which the constraint R_{uv} between u and v includes the pair ab, we can legitimately *remove* that pair if there's no value c such that $ac \in R_{uw}$ and $bc \in R_{vw}$. (See exercise 276.)

time stamps
Horn core algorithm
ternary constraints
hidden variables
singleton domain consistency
Path consistency
binary constraints
unary constraints
composition of binary relations

Efficiency. The search tree for a CSP can become significantly smaller when we use consistency to filter the domains. For example, we noted in Section 7.2.2 that the backtrack tree for all 14772512 solutions to the 16 queens problem — analogous to the tree for 8 queens, which was shown in its entirety in Fig. 68 — has 1,141,190,303 nodes. By contrast, when that same problem is regarded as an exact cover problem, and solved via Algorithm 7.2.2.1X using 16^2 options analogous to 7.2.2.1–(23), the search tree has only 193,021,021 nodes: a 6-fold reduction. Algorithm 7.2.2.1X essentially uses forward consistency, together with the MRV heuristic to choose variables for branching. And if we prune the tree further, by maintaining full domain consistency before every branch, the total number of nodes goes down to 139,562,927: fewer than 10 nodes per solution. Careful use of symmetry, as explained in exercise 483, can be combined with domain consistency to reduce the search tree size for 16 queens to about 5 million nodes.

Of course search tree size isn't the whole story. We must multiply the number of nodes by the average time spent per node, in order to get the total running time. The average time per node, when the basic backtrack Algorithm 7.2.2B was specialized to the 16 queens problem, came to only about 98 mems; and we reduced that to 30 mems per node with Algorithm 7.2.2B*. Furthermore, we saw that bitwise operations and Algorithm 7.2.2W gave a further reduction to only $8\mu/\nu$, hence a total running time of 9 Gμ to find all solutions. That was a winner over Algorithm 7.2.2.1X, which needed 40 Gμ to find those solutions, even though its search tree was only 1/6th the size. Furthermore, even the sophisticated method of exercise 483, with its "tiny" 5-meganode search tree, needs 26 Gμ to find all solutions, when it maintains domain consistency with the state-of-the-art algorithm AC6 devised by C. Bessière, *Artificial Intelligence* **65** (1994), 179–190.

Let's pause a minute to understand why it makes sense to say that "Algorithm 7.2.2.1X essentially uses forward consistency." The same is actually true also for Algorithm 7.2.2.1C, with respect to any XCC problem. Indeed, *any XCC problem can be regarded as a CSP, whose variables are the primary items and whose domains are the options.* Option o belongs to the domain of item i if and only if $i \in o$. The task is to choose nonconflicting options so that every primary item is covered by some option of its domain. In other words, whenever i and i' are distinct variables, we're allowed to assign o to i and o' to i' if and only if o and o' are *compatible*, where compatibility (written '$o \parallel o'$') is defined as follows:

$$o \parallel o' \iff \text{either } o = o' \text{ or } o \cap o' = \{\text{colored items}\}, \qquad (92)$$

where '{colored items}' means the explicitly colored secondary items of $o \cup o'$.

Using the language of Definition V, the bindings (v, a) of an XCC are the pairs (i, o) where $i \in o$. When step C5 of Algorithm 7.2.2.1C covers item i with the option o that's specified by x_l, it means that option o is *assigned* to variable i, as well as to any other primary items i' that happen to be contained in o. Those variables become inactive; so the 'cover' operation 7.2.2.1–(12) removes them from the "active list" of not-yet-covered items $i_1, \ldots, i_t$ that are accessible from `RLINK(0)` in step C3. And it's not difficult to verify that the effect of the 'hide' and 'purify' operations in 7.2.2.1–(13) and 7.2.2.1–(55) is to remove from the

efficiency–
backtrack tree
16 queens problem
forward consistency
MRV heuristic
symmetry
running time
bitwise operations
AC6
Bessière
XCC problem
XCC as CSP
primary items
options
compatible
assigned
inactive
cover
active list

system precisely the options o' that are incompatible with o. In other words, the algorithm reduces the domains so that every remaining active binding (i', o') is weakly viable; and that, by definition, is forward consistency! Notice that the current number of active bindings (i, o) for i is what the algorithm calls `LEN(i)`.

In order to maintain forward consistency throughout the search process, Algorithm 7.2.2.1C needed some elaborate data structures, which it based on the technique of "dancing links." Extensive computational experience with a wide variety of XCC problems has shown that this extra work often pays off handsomely in practice, because it substantially reduces the search tree without costing a great deal per node.

In other words, FC is usually a big win when constraints are propagated. But there are exceptions, such as the algorithm recommended for the graceful labeling problem in the answer to exercise 95. That problem doesn't benefit much from forward consistency, at the most important levels of search, in comparison with less expensive heuristics that are custom-tailored for gracefulness.

While writing the present book, the author was surprised to find that FC also worked faster than DC, in most of the problems that he studied. We did observe in Section 7.2.2.1 that many XCC problems are solved more quickly if we start by preprocessing them with Algorithm 7.2.2.1P; that's like applying DC once, at the beginning. But problems that benefit from "inprocessing" as well as preprocessing are much more rare, as we noticed in Section 7.2.2.2. Data structures that maintain full domain consistency through the search necessarily add a level of complexity that will pay for itself only in situations where the search tree is substantially scaled down.

Of course there do exist problems where DC inprocessing is dramatically better than the maintenance of FC together with preprocessing. For example, D. Sabin and E. Freuder showed in *LNCS* **874** (1994), 10–20, that *random* CSPs with suitably chosen parameters are best handled with DC. On the other hand, we know from experience with SAT solving in Section 7.2.2.2 that the study of random problems can be misleading, because random problems tend to have quite different behavior from "real" applications, with respect to backtracking.

The simplest nonrandom CSPs for which DC inprocessing can be definitely recommended are probably special cases of the "(d, n)-modstep problem," which is to find all d-ary sequences $x_0 x_1 \ldots x_{n-1}$ such that we have

$$x_{(k+1) \bmod n} \in \{x_k, (x_k + 1) \bmod d\}, \qquad \text{for } 0 \le k < n. \tag{93}$$

This problem has n variables x_k, each with domain $D_k = \{0, 1, \ldots, d-1\}$, and n binary constraints (93). The solutions when $d = 3$ and $n = 4$ are 0000, 0012, 0112, 0120, 0122, 1111, 1120, 1200, 1201, 1220, 2001, 2011, 2012, 2201, and 2222.

Consider the case $d = 4$, $n = 5$. If we assign $x_0 \leftarrow 0$, FC will hold if we reduce D_1 to $\{0, 1\}$ and D_4 to $\{0, 3\}$; domains D_2 and D_3 are still $\{0, 1, 2, 3, 4\}$. But DC would reduce D_2 to $\{0, 1, 2\}$ and D_3 to $\{0, 2, 3\}$. After subsequent assignments $x_1 \leftarrow 0$, $x_2 \leftarrow 1$, the FC-only method won't know that $x_3 \leftarrow 1$ is doomed to fail.

The behavior of the $(n-1, n)$-modstep problem for all values of n is not difficult to discover, with respect to both FC and DC. Exercise 283 proves that

weakly viable
data structures
dancing links
graceful labeling
author
preprocessing
inprocessing
Sabin
Freuder
random CSPs
modstep problem
d-ary sequence: Sequence consisting of digits $\{0, 1, \ldots, d-1\}$.

the associated search tree has approximately $2^n n$ nodes, when the domains are maintained with forward consistency only; but the size goes down to only about $n^3/2$ when full domain consistency is maintained.

Later in this section we'll discuss Algorithm S, which solves XCC problems by maintaining full DC instead of just FC. When the XCC formulation of the $(23, 24)$-modstep problem is fed to Algorithm 7.2.2.1C, the 575 solutions are found after 267 gigamems of computation; but Algorithm S polishes them off after just 29 *mega*mems. Thus the $(n-1, n)$-modstep problem is a slam dunk for DC over FC.

Another simple CSP for which DC shines is the problem of listing all "slow growth permutations" of order n: These are the permutations $p_1 p_2 \ldots p_n$ of $\{1, 2, \ldots, n\}$ for which we have

$$p_{k+1} \;\le\; p_k + 1 \qquad \text{for } 1 \le k < n. \tag{94}$$

For example, they're 1234, 2341, 3412, 3421, 4123, 4231, 4312, and 4321 when $n = 4$. It turns out that there are 2^{n-1} such permutations in general, and they can be obtained as the solutions to a nice little XCC problem (see exercise 287).

When that problem for $n = 24$ is solved by Algorithm 7.2.2.1C, its $2^{23} = 8388608$ solutions are found in 3 teramems, while traversing a 3.5-giganode search tree. By contrast, Algorithm S needs only 96 gigamems and 42 meganodes. Algorithm S beats Algorithm 7.2.2.1C on the $(n-1, n)$-modstep problem for all $n \ge 8$, and on the slow growth problem for all $n \ge 12$.

But there's a surprise: With an improved model for slow growth permutations (see exercise 288), Algorithm 7.2.2.1C comes back into the lead! Indeed, that new formulation of the problem, based on the theory in exercise 286, is able to find the 2^{23} solutions in just 17.4 gigamems (and 33.6 meganodes), using FC propagation only! And the new model makes Algorithm S slower (132 Gμ).

We looked at hundreds of XCC problems in Section 7.2.2.1: Langford pairs, polyomino packings, edge matchings, sudoku automorphisms, kenken, masyu, etc., etc.; it's natural to hope that Algorithm S will improve on the results reported there, because of its ability to look further ahead and thereby to reduce branching. Alas, it usually turns out to be *worse* than the best FC-only methods — although there are exceptions, such as double word squares (exercise 7.2.2.1–87) and "alphabet blocks" (exercise 7.2.2.1–113). Algorithm S also wins big on certain problems beyond the scope of this book, such as the *radio link frequency assignment problem* (RLFAP); see B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners, *Constraints* **4** (1999), 79–89, and the numerous benchmarks at `https://xcsp.org/assets/instances/Rlfap.tgz`. Furthermore it's a winner on fillomino puzzles (exercises 289–297), which weren't mentioned in Section 7.2.2.1.

The moral of this story seems to be that a CSP solver should usually try to maintain FC (forward consistency) throughout a search. But one should think twice before going to the extra expense of maintaining DC (domain consistency).

Representing the domains. The current domains of the active variables change frequently from level to level of the search tree. So we need efficient

slow growth permutations
word squares
alphabet blocks
radio link frequency assignment problem
RLFAP
Cabon
de Givry
Lobjois
Schiex
Warners
benchmarks
fillomino puzzles
domains, representation of–
data structures–

mechanisms to update them when we make assignments, and to downdate them when we backtrack.

The simplest expedient is to push a fresh copy of all current domains onto a stack, whenever we enter a new level of recursion. But that's usually not a great idea when efficiency is important, especially not when many domains are large.

One of the most common approaches is therefore to use Floyd's idea of *reversible memory*, as discussed in Eq. 7.2.2–(24) and (25), possibly also refined with "stamps" as in Eq. 7.2.2–(26). An auxiliary stack called the *trail* contains pairs (variable, value) that can be used to restore the previous values of state variables when backtracking. Figure 117 illustrates a typical example.

	$x\ y$	*Trail*	σ	$x{,}x'$	$y{,}y'$	*Trail*
begin α	0 0	$\vert$	1	0,0	0,0	$\vert$
$x \leftarrow 1$	1 0	$\begin{smallmatrix}\vert&x\\\vert&0\end{smallmatrix}$	1	1,1	0,0	$\begin{smallmatrix}\vert&x\\\vert&0\end{smallmatrix}$
$y \leftarrow x$	1 1	$\begin{smallmatrix}\vert&x&y\\\vert&0&0\end{smallmatrix}$	1	1,1	1,1	$\begin{smallmatrix}\vert&x&y\\\vert&0&0\end{smallmatrix}$
$x \leftarrow 2$	2 1	$\begin{smallmatrix}\vert&x&y&x\\\vert&0&0&1\end{smallmatrix}$	1	2,1	1,1	$\begin{smallmatrix}\vert&x&y\\\vert&0&0\end{smallmatrix}$
begin α_1	2 1	$\begin{smallmatrix}\vert&x&y&x&\vert\\\vert&0&0&1&\vert\end{smallmatrix}$	2	2,1	1,1	$\begin{smallmatrix}\vert&x&y&\vert\\\vert&0&0&\vert\end{smallmatrix}$
$y \leftarrow x$	2 2	$\begin{smallmatrix}\vert&x&y&x&\vert&y\\\vert&0&0&1&\vert&1\end{smallmatrix}$	2	2,1	2,2	$\begin{smallmatrix}\vert&x&y&\vert&y\\\vert&0&0&\vert&1\end{smallmatrix}$
$x \leftarrow 3$	3 2	$\begin{smallmatrix}\vert&x&y&x&\vert&y&x\\\vert&0&0&1&\vert&1&2\end{smallmatrix}$	2	3,2	2,2	$\begin{smallmatrix}\vert&x&y&\vert&y&x\\\vert&0&0&\vert&1&2\end{smallmatrix}$
back to α	2 1	$\begin{smallmatrix}\vert&x&y&x\\\vert&0&0&1\end{smallmatrix}$	3	2,2	1,2	$\begin{smallmatrix}\vert&x&y\\\vert&0&0\end{smallmatrix}$
$x \leftarrow y$	1 1	$\begin{smallmatrix}\vert&x&y&x&x\\\vert&0&0&1&2\end{smallmatrix}$	3	1,3	1,2	$\begin{smallmatrix}\vert&x&y&x\\\vert&0&0&2\end{smallmatrix}$
begin α_2	1 1	$\begin{smallmatrix}\vert&x&y&x&x&\vert\\\vert&0&0&1&2&\vert\end{smallmatrix}$	4	1,3	1,2	$\begin{smallmatrix}\vert&x&y&x&\vert\\\vert&0&0&2&\vert\end{smallmatrix}$
$y \leftarrow 4$	1 4	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y\\\vert&0&0&1&2&\vert&1\end{smallmatrix}$	4	1,3	4,4	$\begin{smallmatrix}\vert&x&y&x&\vert&y\\\vert&0&0&2&\vert&1\end{smallmatrix}$
$y \leftarrow x$	1 1	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&y\\\vert&0&0&1&2&\vert&1&4\end{smallmatrix}$	4	1,3	1,4	$\begin{smallmatrix}\vert&x&y&x&\vert&y\\\vert&0&0&2&\vert&1\end{smallmatrix}$
begin α_{21}	1 1	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&y&\vert\\\vert&0&0&1&2&\vert&1&4&\vert\end{smallmatrix}$	5	1,3	1,4	$\begin{smallmatrix}\vert&x&y&x&\vert&y&\vert\\\vert&0&0&2&\vert&1&\vert\end{smallmatrix}$
$x \leftarrow 5$	5 1	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&y&\vert&x\\\vert&0&0&1&2&\vert&1&4&\vert&1\end{smallmatrix}$	5	5,5	1,4	$\begin{smallmatrix}\vert&x&y&x&\vert&y&\vert&x\\\vert&0&0&2&\vert&1&\vert&1\end{smallmatrix}$
back to α_2	1 1	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&y\\\vert&0&0&1&2&\vert&1&4\end{smallmatrix}$	6	1,5	1,4	$\begin{smallmatrix}\vert&x&y&x&\vert&y\\\vert&0&0&2&\vert&1\end{smallmatrix}$
$y \leftarrow 2$	1 2	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&y&y\\\vert&0&0&1&2&\vert&1&4&1\end{smallmatrix}$	6	1,5	2,6	$\begin{smallmatrix}\vert&x&y&x&\vert&y&y\\\vert&0&0&2&\vert&1&1\end{smallmatrix}$
$y \leftarrow x$	1 1	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&y&y&y\\\vert&0&0&1&2&\vert&1&4&1&2\end{smallmatrix}$	6	1,5	1,6	$\begin{smallmatrix}\vert&x&y&x&\vert&y&y\\\vert&0&0&2&\vert&1&1\end{smallmatrix}$
back to α	1 1	$\begin{smallmatrix}\vert&x&y&x&x\\\vert&0&0&1&2\end{smallmatrix}$	7	1,5	1,6	$\begin{smallmatrix}\vert&x&y&x\\\vert&0&0&2\end{smallmatrix}$
$x \leftarrow 8$	8 1	$\begin{smallmatrix}\vert&x&y&x&x&x\\\vert&0&0&1&2&1\end{smallmatrix}$	7	8,7	1,6	$\begin{smallmatrix}\vert&x&y&x&x\\\vert&0&0&2&1\end{smallmatrix}$
begin α_3	8 1	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert\\\vert&0&0&1&2&1&\vert\end{smallmatrix}$	8	8,7	1,6	$\begin{smallmatrix}\vert&x&y&x&x&\vert\\\vert&0&0&2&1&\vert\end{smallmatrix}$
$y \leftarrow x$	8 8	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert&y\\\vert&0&0&1&2&1&\vert&1\end{smallmatrix}$	8	8,7	8,8	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y\\\vert&0&0&2&1&\vert&1\end{smallmatrix}$
$x \leftarrow 3$	3 8	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert&y&x\\\vert&0&0&1&2&1&\vert&1&8\end{smallmatrix}$	8	3,8	8,8	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&x\\\vert&0&0&2&1&\vert&1&8\end{smallmatrix}$
$y \leftarrow 5$	3 5	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert&y&x&y\\\vert&0&0&1&2&1&\vert&1&8&8\end{smallmatrix}$	8	3,8	5,8	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&x\\\vert&0&0&2&1&\vert&1&8\end{smallmatrix}$
begin α_{31}	3 5	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert&y&x&y&\vert\\\vert&0&0&1&2&1&\vert&1&8&8&\vert\end{smallmatrix}$	9	3,8	5,8	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&x&\vert\\\vert&0&0&2&1&\vert&1&8&\vert\end{smallmatrix}$
$y \leftarrow 4$	3 4	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert&y&x&y&\vert&y\\\vert&0&0&1&2&1&\vert&1&8&8&\vert&5\end{smallmatrix}$	9	3,8	4,9	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&x&\vert&y\\\vert&0&0&2&1&\vert&1&8&\vert&5\end{smallmatrix}$
back to α_3	3 5	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert&y&x&y\\\vert&0&0&1&2&1&\vert&1&8&8\end{smallmatrix}$	10	3,8	5,9	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&x\\\vert&0&0&2&1&\vert&1&8\end{smallmatrix}$
$x \leftarrow y$	5 5	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert&y&x&y&x\\\vert&0&0&1&2&1&\vert&1&8&8&3\end{smallmatrix}$	10	5,10	5,9	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&x&x\\\vert&0&0&2&1&\vert&1&8&3\end{smallmatrix}$
begin α_{32}	5 5	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert&y&x&y&x&\vert\\\vert&0&0&1&2&1&\vert&1&8&8&3&\vert\end{smallmatrix}$	11	5,10	5,9	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&x&x&\vert\\\vert&0&0&2&1&\vert&1&8&3&\vert\end{smallmatrix}$
$y \leftarrow 4$	5 4	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert&y&x&y&x&\vert&y\\\vert&0&0&1&2&1&\vert&1&8&8&3&\vert&5\end{smallmatrix}$	11	5,10	4,11	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&x&x&\vert&y\\\vert&0&0&2&1&\vert&1&8&3&\vert&5\end{smallmatrix}$
back to α_3	5 5	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert&y&x&y&x\\\vert&0&0&1&2&1&\vert&1&8&8&3\end{smallmatrix}$	12	5,10	5,11	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&x&x\\\vert&0&0&2&1&\vert&1&8&3\end{smallmatrix}$
$x \leftarrow 6$	6 5	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert&y&x&y&x&x\\\vert&0&0&1&2&1&\vert&1&8&8&3&5\end{smallmatrix}$	12	6,12	5,11	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&x&x&x\\\vert&0&0&2&1&\vert&1&8&3&5\end{smallmatrix}$
$y \leftarrow x$	6 6	$\begin{smallmatrix}\vert&x&y&x&x&x&\vert&y&x&y&x&x&y\\\vert&0&0&1&2&1&\vert&1&8&8&3&5&5\end{smallmatrix}$	12	6,12	6,12	$\begin{smallmatrix}\vert&x&y&x&x&\vert&y&x&x&x&y\\\vert&0&0&2&1&\vert&1&8&3&5&5\end{smallmatrix}$
back to α	8 1	$\begin{smallmatrix}\vert&x&y&x&x&x\\\vert&0&0&1&2&1\end{smallmatrix}$	13	8,12	1,12	$\begin{smallmatrix}\vert&x&y&x&x\\\vert&0&0&2&1\end{smallmatrix}$
$y \leftarrow 7$	8 7	$\begin{smallmatrix}\vert&x&y&x&x&x&y\\\vert&0&0&1&2&1&1\end{smallmatrix}$	13	8,12	7,13	$\begin{smallmatrix}\vert&x&y&x&x&y\\\vert&0&0&2&1&1\end{smallmatrix}$
back out	0 0		14	0,12	0,13	

Fig. 117. Reversible storage is implemented by keeping a trail of changes that need to be undone. An entry like '$\genfrac{}{}{0pt}{}{x}{0}$' means "reset x to 0." The variation at the right saves space by trailing changes to x only when x' doesn't match the current stamp σ.

stack
Floyd
reversible memory
stamps
trail
State variable: A variable that helps to govern a process

In Fig. 117, node α of a search tree has three children α_1, α_2, and α_3; α_1 is a "leaf" (either a solution or a contradiction), while α_2 has a child α_{21} and α_3 has two children $\{\alpha_{31}, \alpha_{32}\}$, all leaves. The illustration shows the current states of variables x and y as the computation proceeds; at the end, x and y once again have their original values. The right-hand version uses $\texttt{STAMP}(x) = x'$ and $\texttt{STAMP}(y) = y'$ to avoid placing some redundant entries on the trail; the current stamp σ increases by 1 whenever we enter a new node or backtrack to a parent.

Notice that stamping doesn't really help much in this particular example; at line 34 the trail has five entries whose net effect is simply to reset $x \leftarrow 0$ and $y \leftarrow 0$. Exercise 299 explains how we might be able to do better.

When the original domain of variable v has d elements, it's usually best to represent it internally as the set $\{0, 1, \ldots, d-1\}$. So we shall assume in the following discussion that the current domain D_v is always contained in that set.

We typically need to do four basic things with D_v:

- Determine whether or not $a \in D_v$, given a value $0 \le a < d$;
- Delete a given value a from D_v, if it is present;
- Determine the size, $|D_v|$;
- Visit ("iterate through") all elements of D_v.

Elements that are deleted will have to be undeleted later.

Three kinds of data structures suggest themselves for operations such as these: Bit vectors; doubly linked lists; "sparse-sets." Let's consider them in turn.

First, when $d \le 64$, it's attractive to work with the binary number

$$\texttt{BITS}(v) \;=\; \sum\{2^a \mid a \in D_v\}, \tag{95}$$

and to take advantage of a computer's bitwise operations. Indeed,

$$a \in D_v \iff \texttt{BITS}(v)\;\&\;(1 \ll a) \ne 0; \tag{96}$$

$$D_v \leftarrow D_v \setminus \{a\} \iff \texttt{BITS}(v) \leftarrow \texttt{BITS}(v)\;\&\;{\sim}(1 \ll a); \tag{97}$$

$$|D_v| \le 1 \iff \texttt{BITS}(v)\;\&\;(\texttt{BITS}(v) - 1) = 0. \tag{98}$$

To compute $|D_v| = \nu\,\texttt{BITS}(v)$, we can use a built-in instruction like MMIX's SADD, or a trick like 7.1.3–(62). And to iterate through D_v, we can do this:

$$\text{Set } t \leftarrow \texttt{BITS}(v)\text{; while } t \ne 0\text{, visit } \rho(t\;\&\;{-t}) \text{ and set } t \leftarrow t - (t\;\&\;{-t}). \tag{99}$$

In a CSP with only binary constraints, we can maintain forward consistency after making the assignment $v \leftarrow a$ by simply setting

$$\texttt{BITS}(w) \leftarrow \texttt{BITS}(w)\;\&\;C_{v,a,w}, \quad \text{for all } w \text{ related to } v, \tag{100}$$

where $C_{v,a,w}$ is an appropriate constant (namely, row a of the Boolean matrix for the constraint between v and w).

Second, our old standby from Section 2.2.5, the doubly linked list, is another natural choice. If $d < 2^{32}$ we can, for instance, work with an array of $d+1$ octabytes, for every variable v, where every octabyte for $0 \le a \le d$ contains two tetrabytes called $\texttt{PREV}_v(a)$ and $\texttt{NEXT}_v(a)$ that link to the neighbors of a in v's list. For simplicity we'll write PREV and NEXT instead of $\texttt{PREV}_v$ and $\texttt{NEXT}_v$.

stamping
Bit vectors
bitwise operations
MMIX
SADD
sideways addition
ruler function ρ
binary constraints
forward consistency
assignment
Boolean matrix
doubly linked list

The special value $a = d$ serves as the list head. If the current domain D_v is $\{a_1, \ldots, a_s\}$, where $0 \le a_1 < a_2 < \cdots < a_s < d$ and $s > 0$, we'll have

$$\texttt{NEXT}(d) = a_1, \quad \texttt{NEXT}(a_j) = a_{j+1} \text{ for } 1 \le j < s, \quad \texttt{NEXT}(a_s) = d; \tag{101}$$
$$\texttt{PREV}(d) = a_s, \quad \texttt{PREV}(a_j) = a_{j-1} \text{ for } 1 < j \le s, \quad \texttt{PREV}(a_1) = d; \tag{102}$$

and if $s = 0$ we'll have $\texttt{NEXT}(d) = \texttt{PREV}(d) = d$. Notice that we always have

$$\texttt{NEXT}(\texttt{PREV}(a)) = \texttt{PREV}(\texttt{NEXT}(a)) = a, \quad \text{if } a \in D_v \text{ or } a = d. \tag{103}$$

Let's also add a Boolean array

$$\texttt{IN}_v[a] = [a \in D_v], \quad \text{for } 0 \le a < d; \quad \texttt{IN}_v[d] = 0. \tag{104}$$

(If $d < 2^{31}$, these bits will fit with the octabytes containing PREV and NEXT.)

The four basic operations are obviously easy with this representation. Furthermore, the "dancing links" protocol, 7.2.2.1–(1) and 7.2.2.1–(2), tells us how to preserve historical information so that the undeletion operation is simple:

$$\texttt{NEXT}(\texttt{PREV}(a)) \leftarrow a \qquad \text{and} \qquad \texttt{PREV}(\texttt{NEXT}(a)) \leftarrow a. \tag{105}$$

Exercise 300 proves that the dancing links protocol also has an interesting property that was *not* mentioned in Section 7.2.2.1.

Third, the sequential *sparse-set representation*, which we discussed in 7.2.2–(16) through 7.2.2–(23), can be adapted to domain representation in a very nice way. Each variable v is now represented by two arrays $\texttt{DOM}_v[k]$ and $\texttt{IDOM}_v[k]$ for $0 \le k < d$, together with another variable $\texttt{SIZE}(v)$. Both $\texttt{DOM}_v$ and $\texttt{IDOM}_v$ are permutations of $\{0, 1, \ldots, d-1\}$; and $\texttt{IDOM}_v$ is the inverse of $\texttt{DOM}_v$:

$$\texttt{DOM}_v[\texttt{IDOM}_v[a]] = \texttt{IDOM}_v[\texttt{DOM}_v[a]] = a, \quad \text{for } 0 \le a < d. \tag{106}$$

Furthermore, the current value of v's domain is simply

$$D_v = \{\texttt{DOM}_v[k] \mid 0 \le k < \texttt{SIZE}(v)\}, \tag{107}$$

and these elements can appear in any order. For example, if $d = 7$ and $D_v = \{1, 3, 4, 5\}$, we might have

$$\begin{aligned} k &= 0\ 1\ 2\ 3\ 4\ 5\ 6 \\ \texttt{DOM}_v[k] &= 3\ 1\ 4\ 5\ 2\ 6\ 0 \quad \text{and} \quad \texttt{SIZE}(v) = 4. \\ \texttt{IDOM}_v[k] &= 6\ 1\ 4\ 0\ 2\ 3\ 5 \end{aligned} \tag{108}$$

(That particular domain can in fact be represented in 4! 3! different ways.) Notice that

$$a \in D_v \iff \texttt{IDOM}_v[a] < \texttt{SIZE}(v). \tag{109}$$

The main point of interest for this representation is the deletion operation:

$$\begin{aligned} &\text{Set } k \leftarrow \texttt{IDOM}_v[a]. \\ &\text{If } k < \texttt{SIZE}(v), \text{ set } \texttt{SIZE}(v) \leftarrow \texttt{SIZE}(v) - 1,\ a' \leftarrow \texttt{DOM}_v[\texttt{SIZE}(v)], \\ &\quad \texttt{DOM}_v[\texttt{SIZE}(v)] \leftarrow a,\ \texttt{IDOM}_v[a] \leftarrow \texttt{SIZE}(v),\ \texttt{DOM}_v[k] \leftarrow a',\ \texttt{IDOM}_v[a'] \leftarrow k. \end{aligned} \tag{110}$$

It's interesting because undeletion is "free": We just set $\texttt{SIZE}(v) \leftarrow \texttt{SIZE}(v) + 1$.

In fact, a whole round of deletions can be undone by just restoring the previous value of $\texttt{SIZE}(v)$; only $\texttt{SIZE}(v)$ needs to be placed in the trail.

list head
head of list: see List head
dancing links
undeletion
sparse-set representation
permutations
inverse
pi, random

The sparse-set representation has an important property that's often useful: *The elements* $\mathtt{DOM}_v\mathtt{[}d-1\mathtt{]}$, $\mathtt{DOM}_v\mathtt{[}d-2\mathtt{]}$, …, $\mathtt{DOM}_v\mathtt{[SIZE(}v\mathtt{)]}$ *are the values not present in the current domain* D_v*, in the exact order in which they were deleted.* In other words, the most recent changes to the domain appear together, in positions $\mathtt{SIZE(}v\mathtt{)}$, $\mathtt{SIZE(}v\mathtt{)}+1$, … of $\mathtt{DOM}_v$.

Another advantage is the fact that the sequentially accessed array $\mathtt{DOM}_v$ is more cache-friendly than a doubly linked list. On the other hand, the order of domain elements is not preserved; that might be a handicap (see exercise 302).

Sparse-set technology can also be applied in a rather different way, which sometimes makes bit vectors attractive even when the domain size d is very large. This combination of ideas gives us a fourth candidate for representing domains, a new type of data structure called a "reversible sparse bitset."

In general, suppose we want to represent a set D of d elements within a computer that has e-bit words, using $q = \lceil d/e \rceil$ of those words to store bit vectors b_k for $0 \le k < q$. The natural way to do this is to represent the element a of D as bit $a - ke$ of word $b_{\lfloor a/e \rfloor}$; more precisely,

$$b_k \;=\; \sum\{2^{a-ke} \mid a \in D \text{ and } ke \le a < (k+1)e\}. \tag{111}$$

This scheme is called the *bitset* representation of D.

When the set D continually gets smaller and smaller as a computation proceeds, many of the individual words b_k will be zero, and we won't want to look at them again. That's where sparse-set principles come into play: We can maintain an array of q elements, $\mathtt{D[}j\mathtt{]}$ for $0 \le j < q$, and an integer $\mathtt{S}$, with

$$b_{\mathtt{D[}j\mathtt{]}} \ne 0 \quad\iff\quad j < \mathtt{S}, \qquad \text{for } 0 \le j < q. \tag{112}$$

Thus $\mathtt{D}$ and $\mathtt{S}$ play the roles of $\mathtt{DOM}_v$ and $\mathtt{SIZE(}v\mathtt{)}$ in (107); the inverse permutation $\mathtt{IDOM}_v$ isn't needed. Figure 118 shows how it works.

	b_0	b_1	b_2	b_3	b_4	b_5	b_6	D	S
Initial set $\{0,1,\ldots,19\}$:	111	111	111	111	111	111	110	$0123456\vert$	7
AND with 011 001 001 000 011 111 10:	011	001	001	000	011	111	100	$012645\vert 3$	6
AND with 010 001 000 010 110 100 01:	010	001	000	000	010	100	000	$0154\vert 623$	4
AND with 101 001 100 010 011 000 11:	000	001	000	000	010	000	000	$41\vert 50623$	2

Fig. 118. The *sparse bitset* representation of a set with d elements, using $q = \lceil d/e \rceil$ words of e bits each, illustrated here for $d = 20$ and $e = 3$, hence $q = 7$. (Of course e would be much larger in an actual computer; with MMIX we'd choose $e = 64$. Notice that if the bits are numbered 0 to 19 from left to right, word b_6 initially contains the binary number $(011)_2$, *not* $(110)_2$, according to Eq. (111).) The first $\mathtt{S}$ elements of $\mathtt{D}$ tell us where to find all of the words b_k that are still nonzero. Exercise 304 explains how to perform the AND operations in $O(\mathtt{S})$ steps, while changing $\mathtt{D}$ as little as possible.

A sparse bitset such as this becomes *reversible* if we make all of the individual words b_k reversible, by recording their changes in the trail, together with the value of $\mathtt{S}$ before those changes were made.

cache-friendly
bit vectors
reversible sparse bitset
sparse bitset
bitset
pi, as random
trail

dancing cells–
author
Solnon
internal numbers
option list
spacers

***Dancing cells.** We've just seen that sparse-set arrays can perform many of the functions of doubly linked lists, without needing any more space. How far can we push this idea? Could a sparse-set representation possibly compete with the dancingly linked lists in the core of Algorithm 7.2.2.1C, which has been the most popular XCC solver for many years?

That question was posed to the author in 2020 by Christine Solnon, and the answer turns out to be "yes"(!). She has suggested that the newfangled XCC solver be known as "dancing cells," because exquisite choreography once again governs the computations. Algorithm C below includes many of her ideas.

The easiest way to understand dancing cells is to look again at the toy problem 7.2.2.1–(49), with which we introduced the original algorithm. That problem has three primary items $\{\mathtt{p},\mathtt{q},\mathtt{r}\}$, two secondary items $\{\mathtt{x},\mathtt{y}\}$, and five options:

$$\text{‘}\mathtt{p\ q\ x\ y{:}A}\text{’};\ \text{‘}\mathtt{p\ r\ x{:}A\ y}\text{’};\ \text{‘}\mathtt{p\ x{:}B}\text{’};\ \text{‘}\mathtt{q\ x{:}A}\text{’};\ \text{‘}\mathtt{r\ y{:}B}\text{’}. \qquad (113)$$

Table 7.2.2.1–2 illustrated the previous data structures for (113). There were doubly linked lists of primary and secondary items, using `LLINK` and `RLINK` fields; each item also had a doubly linked list of its options, using `ULINK` and `DLINK`.

Table 3 shows a convenient way to represent those same lists in sparse-set style. There are three arrays, called `ITEM`, `SET`, and `NODE`; the elements of `NODE` have three fields, called `ITM`, `LOC`, and `CLR`. Items have internal numbers, which are listed in `ITEM`; for example, `ITEM[1]` $= 11$ is the internal number for ‘q’.

Suppose `ITEM[`k`]` $= i$. Then the `SET` array, beginning at `SET[`i`]`, lists the places where item i appears in options; and the `NODE` array shows the options themselves. For example, when $k = 1$, we have `SET[11]` $= 2$, and `SET[12]` $= 14$; `NODE[2]` and `NODE[14]` are the two nodes whose `ITM` field is 11. Furthermore, `LOC(2)` $= 11$ and `LOC(14)` $= 12$; these are cross-references back to the `SET` array. Notice also that `SET[`$i-1$`]`, also called `SIZE(`i`)`, is the length of i's option list; `SET[`$i-2$`]`, also called `POS(`i`)`, is k; and item i's name appears before its `POS`.

Options in the `NODE` array are separated by "spacers" as before; the spacer before an option of length l has `LOC` $= l$, and the spacer after it has `ITM` $= -l$.

Table 3

THE INITIAL CONTENTS OF MEMORY CORRESPONDING TO (113)

	i	`SET[`i`]`		i	`SET[`i`]`
LNAME	0	p	•	17	7
RNAME	1			18	17
POS	2	0	LNAME	19	x
SIZE	3	3	RNAME	20	
•	4	1	POS	21	3
	5	6	SIZE	22	4
	6	11	•	23	3
LNAME	7	q		24	8
RNAME	8			25	12
POS	9	1		26	15
SIZE	10	2	LNAME	27	y
•	11	2	RNAME	28	
	12	14	POS	29	4
LNAME	13	r	SIZE	30	3
RNAME	14		•	31	4
POS	15	2		32	9
SIZE	16	2		33	18

k:	0	1	2	3	4		
`ITEM[`k`]`:	4	11	17	23	31		
x:	0	1	2	3	4	5	6
`ITM(`x`)`:	0	4	11	23	31	−4	4
`LOC(`x`)`:	4	4	11	23	31	4	5
`CLR(`x`)`:	—	0	0	0	A	—	0
x:	7	8	9	10	11	12	13
`ITM(`x`)`:	17	23	31	−4	4	23	−2
`LOC(`x`)`:	17	24	32	2	6	25	2
`CLR(`x`)`:	0	A	0	—	0	B	—
x:	14	15	16	17	18	19	
`ITM(`x`)`:	11	23	−2	17	31	−2	
`LOC(`x`)`:	12	26	2	18	33	—	
`CLR(`x`)`:	0	A	—	0	B	—	

Dear reader, please study Table 3 carefully, until you understand exactly how it was obtained from (113). Notice that the `SET` array is heterogeneous: Some of its entries are parts of names, some of its entries are integers, some of its entries point into `NODE`. You should have no trouble figuring out the meaning of $\texttt{CLR}(x)$ — see exercise 307. More importantly, you should understand how `ITEM` and `POS` play the roles of `DOM` and `IDOM` in the discussion above (see (108)), with respect to the list of items, except that `POS` appears in scattered entries of `SET` instead of having an array of its own. Similarly, `SET` and `LOC` play those roles with respect to the option lists. In particular, when i, k, and x have appropriate values, the following relations are invariant, analogous to (106):

$$\texttt{POS}(\texttt{ITEM}[k]) = k \quad\text{and}\quad \texttt{ITEM}[\texttt{POS}(i)] = i; \tag{114}$$

$$\texttt{LOC}(\texttt{SET}[i]) = i \quad\text{and}\quad \texttt{SET}[\texttt{LOC}(x)] = x. \tag{115}$$

Table 3 shows the initial setup, when problem (113) has been input but not yet solved. The algorithm will permute the values of `ITEM` and `POS` as it runs, so that the currently active items all appear at the beginning of `ITEM`. It will also permute `SET` and `LOC` entries, so that the elements of i's current option list are the nodes

$$D_i \;=\; \bigl\{\texttt{SET}[i+j] \mid 0 \le j < \texttt{SIZE}(i)\bigr\}. \tag{116}$$

(It's appropriate to call this set D_i, because — as remarked earlier — the domain of variable i is the set of all options that contain item i and are consistent with previous choices, when an XCC problem is regarded as a CSP.)

On the other hand, the algorithm never changes the `ITM` or `CLR` fields.

$\texttt{ITEM}[k]$ is a primary item if and only if $\texttt{ITEM}[k] < \texttt{SECOND}$. It is currently active if and only if $k < \texttt{ACTIVE}$, where `ACTIVE` is initially the total number of items. (Thus, $\texttt{ACTIVE} = 5$ and $\texttt{SECOND} = 23$ in Table 3.) As the algorithm proceeds, an item is active if and only if it hasn't yet appeared in a chosen option.

If $\texttt{NODE}[x]$ is not a spacer, it represents an item in some option. We say that the *siblings* of x are the nodes for the *other* items in that option; for example, the siblings of 2 are 1, 3, and 4. Here's a simple way to visit all the siblings of x:

$$\begin{array}{l} \text{Set } x' \leftarrow x+1, \text{ and repeat the following while } x' \ne x\text{:} \\ \quad \text{If } \texttt{ITM}(x') > 0, \text{ visit } x' \text{ and set } x' \leftarrow x'+1; \\ \quad \text{otherwise set } x' \leftarrow x' + \texttt{ITM}(x'). \end{array} \tag{117}$$

Algorithm C relies on a technical subroutine called 'hide', which takes the place of routines in Section 7.2.2.1 that were called 'cover', 'hide', 'commit', 'purify', etc. With sparse-set technology, we won't need to write an 'unhide' routine.

The purpose of 'hide(i,c)' is to remove all options of item i's current list from every *other* option list to which they belong, except when i is secondary and $c \ne 0$. In the latter case, we don't discard an option that gives i the specified color c. (More precisely, an option is retained if it includes 'i:c'.) For example, if we hide $i = 11$ in Table 3, the options in nodes 2 and 14 will be removed. (Node 2 is the '`q`' part of the option '`p q x y:A`'; hence that option will disappear from the option lists of `p`, `x`, and `y`.) If we hide $i = 23$ with $c = $ `A`, we'll remove the options in nodes 3 and 12, but not 8 or 15, because they color `x` with `A`.

option lists
invariant
active items
ITEM–
SET–
NODE–
ITM–
LOC–
CLR–
POS–
SIZE–
XCC as CSP
primary item
ACTIVE
SECOND
spacer
siblings
hide(i,c)

FLAG
global variables
OACTIVE
secondary item
MRV: Minimum remaining values

If some primary item is about to lose its last remaining option, the hide routine stops what it was doing and sets $\mathtt{FLAG} \leftarrow 1$, where FLAG is a global variable. This will allow backtracking to occur immediately. (See exercises 313 and 317. Sparse-set principles win here, because the hide routine of 7.2.2.1–(13) had no way to catch the condition $\mathtt{LEN}(x) = 0$ without greatly complicating the unhiding process, when traversing doubly linked lists.) This feature ensures that no primary item's option list will ever become empty.

The hide routine also relies on global variables ACTIVE and OACTIVE, where OACTIVE is the value that ACTIVE had just before items of the current option were being deactivated. It uses local variables j, x, x', x'', i', i'', and s':

$$\text{hide}(i,c) = \begin{cases} \text{For } 0 \le j < \mathtt{SIZE}(i)\text{, set } x \leftarrow \mathtt{SET}[i+j] \text{ and} \\ \qquad \text{do the following if } c = 0 \text{ or } c \ne \mathtt{CLR}(x)\text{:} \\ \quad \text{For all siblings } x' \text{ of } x\text{, set } i' \leftarrow \mathtt{ITM}(x') \text{ and} \\ \qquad\quad \text{do the following if } \mathtt{POS}(i') < \mathtt{OACTIVE}\text{:} \\ \qquad \text{Set } s' \leftarrow \mathtt{SIZE}(i') - 1. \\ \qquad \text{If } s' = 0 \text{ and } \mathtt{FLAG} = 0 \text{ and } i' < \mathtt{SECOND} \text{ and} \\ \qquad\qquad \mathtt{POS}(i') < \mathtt{ACTIVE}\text{, set } \mathtt{FLAG} \leftarrow 1 \text{ and return.} \\ \qquad \text{Otherwise set } x'' \leftarrow \mathtt{SET}[i'+s'],\ \mathtt{SIZE}(i') \leftarrow s', \\ \qquad\qquad \mathtt{SET}[i'+s'] \leftarrow x',\ i'' \leftarrow \mathtt{LOC}(x'),\ \mathtt{LOC}(x') \leftarrow i'+s', \\ \qquad\qquad \mathtt{SET}[i''] \leftarrow x'',\ \mathtt{LOC}(x'') \leftarrow i''. \end{cases} \tag{118}$$

Algorithm C (*Exact covering with colors*). This algorithm visits all solutions to a given XCC problem, using the same conventions as Algorithm 7.2.2.1C; but it's based on sparse-set structures ("dancing cells") instead of doubly linked lists. It maintains sequential lists $x_0x_1\ldots x_T$ and $y_0y_1\ldots y_T$ for backtracking, where T is large enough to accommodate one entry for each option in a partial solution, as well as a sequential stack called TRAIL[0], TRAIL[1], ..., containing pairs.

C1. [Initialize.] Set the problem up in memory, as in Table 3; but terminate if there's any primary item with no options. (See exercise 311.) Also set ACTIVE to the number of items, SECOND to the internal number of the smallest secondary item (or ∞ if there are none), and $l \leftarrow y_0 \leftarrow t \leftarrow 0$.

C2. [Choose i.] Set $i \leftarrow \mathtt{ITEM}[k]$ for some k with $0 \le k < \mathtt{ACTIVE}$ and $\mathtt{ITEM}[k] < \mathtt{SECOND}$ and minimum $\mathtt{SIZE}(i)$. But if no such k exists, go to C9. (The tie-breaking rule in exercise 312 often works well for this step.)

C3. [Deactivate i.] Set $k' \leftarrow \mathtt{ACTIVE} - 1$, $\mathtt{ACTIVE} \leftarrow k'$, $i' \leftarrow \mathtt{ITEM}[k']$, $k \leftarrow \mathtt{POS}(i)$, $\mathtt{ITEM}[k'] \leftarrow i$, $\mathtt{ITEM}[k] \leftarrow i'$, $\mathtt{POS}(i') \leftarrow k$, $\mathtt{POS}(i) \leftarrow k'$.

C4. [Hide i.] Set $\mathtt{OACTIVE} \leftarrow \mathtt{ACTIVE}$, $\mathtt{FLAG} \leftarrow -1$; hide$(i,0)$ and set $j \leftarrow i$.

C5. [Trail the sizes.] Terminate with trail overflow if $t + \mathtt{ACTIVE}$ exceeds the maximum available TRAIL size. Otherwise set $\mathtt{TRAIL}[t+k] \leftarrow \big(\mathtt{ITEM}[k], \mathtt{SIZE}(\mathtt{ITEM}[k])\big)$ for $0 \le k < \mathtt{ACTIVE}$; then set $y_{l+1} \leftarrow t \leftarrow t + \mathtt{ACTIVE}$.

C6. [Try $\mathtt{SET}[j]$.] Set $x_l \leftarrow \mathtt{SET}[j]$ and $k \leftarrow \mathtt{OACTIVE} \leftarrow \mathtt{ACTIVE}$. For all siblings x' of x_l, set $i' \leftarrow \mathtt{ITM}(x')$, $k' \leftarrow \mathtt{POS}(i')$, and if $k' < k$ set $k \leftarrow k-1$, $i'' \leftarrow \mathtt{ITEM}[k]$, $\mathtt{ITEM}[k] \leftarrow i'$, $\mathtt{ITEM}[k'] \leftarrow i''$, $\mathtt{POS}(i'') \leftarrow k'$, $\mathtt{POS}(i') \leftarrow k$. Then set $\mathtt{ACTIVE} \leftarrow k$. (We've deactivated the other items of option x_l.)

C7. [Hide SET[j].] Set FLAG $\leftarrow 0$. For all siblings x' of x_l, set $i' \leftarrow$ ITM(x'); and if $i' <$ SECOND or POS(i') $<$ OACTIVE, hide(i', CLR(x')), and go to C11 if FLAG $= 1$. (See exercise 317.)

C8. [Advance to the next level.] Set $l \leftarrow l + 1$ and return to C2.

C9. [Visit a solution.] Visit the solution that's specified by nodes $x_0x_1 \ldots x_{l-1}$.

C10. [Leave level l.] Terminate if $l = 0$. Otherwise set $l \leftarrow l - 1$, $i \leftarrow$ ITM(x_l), $j \leftarrow$ LOC(x_l).

C11. [Try again?] If $j+1 \geq i+$SIZE(i), go to C10. Otherwise, for $y_l \leq k < y_{l+1}$, set SIZE(i') $\leftarrow s'$ if TRAIL[k] $= (i', s')$. Then set $t \leftarrow y_{l+1}$, ACTIVE $\leftarrow t - y_l$, $j \leftarrow j + 1$, and return to C6. ▮

"extreme" XC problem
mems
update
bizarre
XCC benchmarks
benchmarks
MacMahon
cubes
Dudeney
chessboard
pentominoes
Grabarchuk
snake
windmill dominoes
queen graph
graph coloring

Exercise 314 presents a play-by-play account of the sequel to Table 3.

How well does this new algorithm compete with its predecessor? Hundreds of tests on a wide variety of nontrivial examples from Section 7.2.2.1 give it an excellent scorecard indeed! For example, when we try the "extreme" XC problem with all $2^n - 1$ possible options on n primary items, for $n = 15$ (see 7.2.2.1–(82)), it finds all $\varpi_{15} = 1{,}382{,}958{,}545$ solutions in just 432 gigamems, compared to 611 gigamems for Algorithm 7.2.2.1C. That's just 313 mems per solution (and 10.8 mems per update), for dancing cells, compared to 442 mems per solution (and 15.2 mems per update) for dancing links.

Similarly, when we look for all 108,056,025 matchings of the "bizarre" graph 7.2.2.1–(89) for $q = r = 6$, the new data structures find them in just 15.2 Gμ (141 mems/sol, 14 μ/υ), beating the old 19.4 Gμ (179 mems/sol, 17.8 μ/υ).

Here are a baker's dozen typical XCC benchmarks, for further insights:

code name	(options, items, solutions)	dancing cells runtime	dancing links runtime	ratio	
C	(4320, 30+61, 1566720)	42.2 Gμ	51.9 Gμ	0.813	
D	(2327, 77+1, 16146)	12.5 Gμ	20.3 Gμ	0.614	
H	(1416, 196+93, 5224)	623.4 Gμ	613.4 Gμ	1.016	
K	(343, 49+288, 110968)	9.6 Gμ	3.1 Gμ	3.089	
L	(352, 48+0, 326721800)	881.2 Gμ	1123.3 Gμ	0.784	
M	(1514, 49+42, 987816)	21.6 Gμ	25.3 Gμ	0.854	
O*	(6966, 180+0, 16928)	7105.4 Gμ	12732.0 Gμ	0.558	(119)
Q	(256, 32+58, 14772512)	58.9 Gμ	40.2 Gμ	1.467	
R	(121, 11+741, 401800)	4.8 Gμ	0.8 Gμ	5.816	
S	(3858, 342+90, 30258432)	211.4 Gμ	170.5 Gμ	1.240	
U	(2440, 72+0, 31520)	119.7 Gμ	194.5 Gμ	0.615	
W	(1212, 12+36, 352)	7.4 Gμ	10.5 Gμ	0.702	
Y*	(949, 205+276, 16)	26.3 Gμ	23.2 Gμ	1.132	

You win some, you lose some; it's not clear why.

Problem C in this list comes from MacMahon's 30 colored cubes, exercise 7.2.2.1–146. Problem D is based on Dudeney's original dissection of a chessboard into pentominoes and a square tetromino, exercise 7.2.2.1–274. Problem H comes from the 5×7 subcase of Grabarchuk's double-snake puzzle for windmill dominoes (exercise 7.2.2.1–306). Problem K colors the 7×7 queen graph with 8 colors, using the clique encoding of exercise 7.2.2.1–117(b). Problem L finds all possible Langford pairs for $n = 16$ (exercise 7.2.2.1–15). Problem M makes a hexagon from MacMahon's 24 colored triangles, fixing the position of tile `aaa` (exercise 7.2.2.1–126). Problem O solves another packing problem, called "chunky-octs"; see exercise 298 below. The asterisk in 'O^*' means that Algorithm 7.2.2.1P has been used to preprocess the XCC data, removing unnecessary options and items. Problem Q is the classical 16 queens problem, as in 7.2.2.1–(23), using organ-pipe order for the primary items. Problem R finds all ways to radio-color Mycielski's graph M_4 with 11 colors (exercises 7.2.2.1–116 and 7.2.2.2–36). Problem S enumerates sudoku solutions that are symmetric under transposition (exercise 7.2.2.1–114). Problem U packs the twelve solid pentominoes into a $3\times4\times5$ box (exercise 7.2.2.1–340(b)). Problem W makes 6×6 word search puzzles for the words `ONE` to `TWELVE` (exercise 7.2.2.1–105, but not requiring '`.`'). And Problem Y comes from Fig. 73, considering H-equivalence of Y pentominoes. (See the remarks preceding 7.2.2.1–(97); restrict the central cell to $40/8 = 5$ options.)

A closer look at Algorithm C shows that we can often speed it up by streamlining the cases where an item has only one option left (the "forced moves"). Exercise 318 presents this improvement, which we shall call Algorithm C^+. Algorithm C^+ has an even better scorecard than (119):

code name	(options, items, solutions)	dancing cells runtime	dancing links runtime	ratio
C	(4320, 30+61, 1566720)	41.6 Gμ	51.9 Gμ	0.802
D	(2327, 77+1, 16146)	12.4 Gμ	20.3 Gμ	0.612
H	(1416, 196+93, 5224)	407.4 Gμ	613.4 Gμ	0.664
K	(343, 49+288, 110968)	4.1 Gμ	3.1 Gμ	1.313
L	(352, 48+0, 326721800)	814.7 Gμ	1123.3 Gμ	0.725
M	(1514, 49+42, 987816)	20.6 Gμ	25.3 Gμ	0.814
O*	(6966, 180+0, 16928)	7090.2 Gμ	12732.0 Gμ	0.557
Q	(256, 32+58, 14772512)	43.9 Gμ	40.2 Gμ	1.093
R	(121, 11+741, 401800)	2.9 Gμ	0.8 Gμ	3.515
S	(3858, 342+90, 30258432)	125.9 Gμ	170.5 Gμ	0.738
U	(2440, 72+0, 31520)	119.1 Gμ	194.5 Gμ	0.613
W	(1212, 12+36, 352)	7.4 Gμ	10.5 Gμ	0.702
Y*	(949, 205+276, 16)	23.6 Gμ	23.2 Gμ	1.018

(120)

***Dynamic variable ordering heuristics.** All of the timings reported in (120) were obtained by using the "minimum remaining values" heuristic, aka MRV, to choose the item on which branching will occur. (This is the choice of i in step $\mathrm{C2}^+$ of Algorithm C^+, or in step C3 of Algorithm 7.2.2.1C using exercise 7.2.2.1–9.)

At every node of the search tree, the MRV heuristic requires us to run through all of the active primary items, in order to find one for which `SIZE`(i) is as small as possible.* That might seem unattractive, because the traditional goal of backtrack search is to minimize the amount of computation per node. However, a good choice of i often dramatically decreases the number of nodes.

* More precisely, step C2 of Algorithm C can terminate the loop early if it finds an item with `SIZE`$(i) = 1$. Step C3 of Algorithm 7.2.2.1C can terminate early if it finds `SIZE`$(i) = 0$.

clique
Langford pairs
hexagon
MacMahon
triangles
chunky-octs
preprocess
16 queens problem
organ-pipe order
radio-color
Mycielski
sudoku solutions
transposition
solid pentominoes
pentacubes
word search puzzles
Y pentominoes
pentominoes+
symmetry breaking
forced moves
variable ordering heuristics–
minimum remaining values
MRV
backtrack search

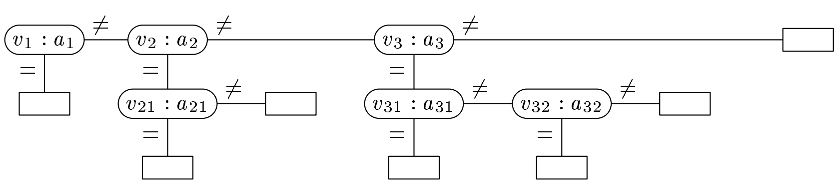
In fact, the amount of time devoted to the MRV heuristic in (120) is comparatively small; it's at most 8%, except in Problem R. In Problem L, for example, only about 33 Gμ of the 814.7 Gμ total running time is spent in step C2. In Problems C, O*, U, and W the MRV time is completely negligible.

Thus it's natural to wonder whether or not we should devote even more time to the choice of an item on which to branch, by somehow improving on MRV. Similarly, in a general CSP, it might be wise to have a better strategy than MRV for choosing the variable on which to branch at each node of the search.

When MRV is not used, a *binary branching* strategy might well be more appropriate than the d-ary branching that's done by Algorithm C. Indeed, after we've explored the subproblem in which item i is covered by option o, our new heuristic might well want to branch next on another item $i' \neq i$, instead of exploring all of the ways to cover i at this stage of the search. (See exercise 320.)

The binary branching strategy requires a reformulation of Algorithm C. Step C4 of that algorithm hides item i, once and for all, before branching on any of i's options, because it knows that i would have been hidden repeatedly later on when each of those options was actually tried. Unfortunately, that optimization is no longer legitimate. (Incidentally, we made the same optimization in step C4 of Algorithm 7.2.2.1C; but we couldn't do it in Algorithm 7.2.2.1M.)

When we do binary branching it's best to view the search tree from a slightly different angle than before, with subtree pointers going south and east instead of southwest and southeast:



(121)

(This tree has basically the same structure as the computation in Fig. 117; node α of Fig. 117 corresponds to the root of (121), namely the node labeled '$v_1 : a_1$'.) Every branch that we take while solving a CSP is represented visually by a node labeled '$v : a$', where v is a variable whose domain contains the value a. A downward branch leads to the subproblem for which $v = a$ in the solution; a rightward branch leads to the subproblem for which $v \neq a$. A contradiction, or a solution, is represented by an unlabeled external node.

(If the CSP is actually an XCC problem, a label such as '$i : o$' would be more appropriate than '$v : a$', where i is an item for which o is an option.)

It's convenient to speak of both "stages" and "levels" in the search tree that arises during binary branching: A node is at *stage* s and *level* l if the path to that node from the root involves exactly s downward branches and l total branches. For example, node '$v_3 : a_3$' in (121) is at stage 0 and level 2. The child node directly below a branch at stage s and level l has stage $s+1$ and level $l+1$; the other child of that node has stage s and level $l+1$. Stages are significant because we always backtrack upward to the right child of a node in the previous stage; after finishing a subtree we never backtrack leftward to a node in the same stage.

branching variable, choice of
binary branching
d-ary branching
stages
levels

Of course the search tree doesn't really appear inside a computer! It exists only in our minds, as a mental model by which we try to understand the steps that a computer takes while solving a CSP. However, when the computer is currently operating in stage s, its data structures do physically record enough information to resume work on each of the s subproblems in prior stages.

A detailed reformulation of Algorithm C^+, using the framework of binary branching rather than d-way branching, appears in the answer to exercise 322, where it is presented as Algorithm B. When no forced moves are present, Algorithm B chooses an item for branching by using an arbitrary user-supplied *heuristic function* h, which returns a floating point value. The idea is to find the active primary item for which $h(i)$ is minimum, breaking ties if necessary by using i's internal code number (its position in `SET`). The special case $h(i) = \mathtt{SIZE}(i)$ gives us MRV; but sometimes we can do much better, by gathering statistics on the fly with respect to combinations of values that have proved to be good or bad. If necessary we can allocate space in the `SET` array to gather such statistics, just as we've already made room for `LNAME`, `RNAME`, `POS`, and `SIZE` in Table 3.

An ideal heuristic function will be relatively easy to compute, while keeping the search tree as small as possible. Delicate tradeoffs are involved; hence it's not surprising that dozens of heuristics for dynamic variable ordering have been proposed. We shall consider two that are particularly appealing because of their simplicity and their effectiveness in a variety of situations.

One of the first significant alternatives to MRV was introduced in 2004 by F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais [*European Conf. on Artificial Intelligence* **16** (2004), 146–150], who called it a "conflict-directed heuristic." When stated in XCC terminology, their idea is to maintain a dynamic weight, $\mathtt{WT}(i)$, for each primary item i. We start with $\mathtt{WT}(i) \leftarrow 1$; then we set $\mathtt{WT}(i) \leftarrow \mathtt{WT}(i) + 1$ whenever we're forced to backtrack when i has lost its last remaining option. In this way the items that are most difficult to handle will tend to get the highest weight, and the following heuristic function suggests itself:

$$h(i) \;=\; \mathtt{SIZE}(i)\big/\mathtt{WT}(i). \qquad (122)$$

Boussemart and his coauthors explained their heuristic by considering an academic yet instructive problem that involves queens and knights: "Place eight queens and five knights on a chessboard in such a way that (a) no two queens are in the same row, column, or diagonal; and (b) the knights are connected by a cycle of knight moves." In other words, the queens must satisfy the conditions of the classical 8 queens problem, and the knights must form a 5-cycle.

This queens-and-knights problem clearly has no solution, because knight moves cannot form a cycle whose length is odd. But the MRV heuristic is a terrible way to establish unsatisfiability! Each queen has at most 8 options, while each knight has more than 50; the algorithm will therefore place all of the queens before trying to do anything with the knights. Laborious trials will show that five knights cannot coexist properly with the existing queens, and the algorithm will go back to reposition the queens. Eventually the fact that five knights can't make a cycle will be re-proved 92 times, once for every valid queen placement.

search tree
forced moves
heuristic function h
internal code number
MRV
tradeoffs
Boussemart
Hemery
Lecoutre
Sais
conflict-directed heuristic
weight
academic
cycle
8 queens problem
queens-and-knights problem

When Algorithm B is applied to the queens-and-knights problem using heuristic (122), it soon gives high weights to knight placement. Consequently it needs to discover the 5-cycle failure only five times instead of 92 (see exercise 326), and it's able to prove unsatisfiability after about 759 megamems of computation. By contrast, the MRV-based Algorithm C^+ needs 7.9 *giga*mems.

We shall call the heuristic (122) 'WTD', meaning "weighted," in order to have a convenient three-letter counterpart to the name 'MRV'.

Of course the queens-and-knights problem has been specially contrived so as to make WTD look good. But WTD often handles "real" instances very nicely too. For example, MRV takes 23.6 Gμ to solve Problem Y* of (120), but WTD needs only 6.6 Gμ. Even better is Problem H, where WTD makes a spectacular improvement from 407.4 Gμ to 19.6 Gμ! It makes small gains also in Problems K and M. But WTD is slower than MRV in the other nine problems considered in (120); for example, it needs 1080.5 Gμ, not 814.7 Gμ, to solve Problem L.

Boussemart, Hemery, Lecoutre, and Sais originally defined their heuristic for *general* CSPs, not for the special case of XCC problems. They associated a weight with each *constraint*; then the weight of a variable v was the sum of the weights of all constraints that contain v and at least one other unassigned variable. For technical reasons they called their heuristic "*dom/wdeg*."

The second heuristic we shall consider is called 'FRB', meaning "failure rate based." It's sort of dual to WTD: When a trial assignment to item i causes the option list of another item i' to be wiped out, WTD increases the weight of i'; but FRB increases the *failure rate* of i. This idea was pioneered by H. Li, M. Yin, and Z. Li [*LIPIcs* **210** (2021), 9:1–9:10], whose paper also introduced several other methods and gave a historical survey of variable ordering heuristics.

To implement FRB, we maintain two new quantities $\mathtt{FR}(i)$ and $\mathtt{TRY}(i)$ for each primary item i, where $\mathtt{FR}(i)$ is initially 0.5 and $\mathtt{TRY}(i)$ is initially 1. After trying to cover item i with one of its options o, we set $\mathtt{TRY}(i) \leftarrow \mathtt{TRY}(i)+1$ and

$$\mathtt{FR}(i) \;=\; \begin{cases} \mathtt{FR}(i)-\mathtt{FR}(i)/\mathtt{TRY}(i), & \text{if nonfailure;}\\ \mathtt{FR}(i)+(1.0-\mathtt{FR}(i))/\mathtt{TRY}(i), & \text{if failure;}\end{cases} \qquad (123)$$

here "failure" means that some primary item i' not in o has lost its last option, causing us to backtrack. The FRB heuristic function for branching is then

$$h(i) \;=\; \mathtt{SIZE}(i)/\mathtt{FR}(i). \qquad (124)$$

(Like MRV, it tends to help us "fail early" in a search, rather than later.)

By definition, we have $\mathtt{TRY}(i)=2$ just after the first time we try to branch on i; and the failure rate $\mathtt{FR}(i)$ is either 0.25 or 0.75. Later, after the second try, we'll have $\mathtt{TRY}(i)=3$ and $\mathtt{FR}(i)\in\{0.1666\ldots,0.5,0.8333\ldots\}$. In general, after the mth attempt to branch on i, $\mathtt{TRY}(i)$ will be $m+1$ and $\mathtt{FR}(i)$ will be in the set $\{\frac{1}{2m+2},\frac{3}{2m+2},\ldots,\frac{2m+1}{2m+2}\}$, an odd multiple of $\frac{1}{2m+2}$. Formula (123) is designed to preserve accuracy in the floating point calculations.

It's a bit of a surprise that FRB does *not* do well on the queens-and-knights problem: It needs 11.0 Gμ, compared to 7.9 Gμ with Algorithm C^+. (See also exercise 330.)

WTD
weighted
Boussemart
Hemery
Lecoutre
Sais
dom/wdeg
failure rate based
Li
Yin
Li
historical survey
failure
fail early
floating point calculations

But FRB really shines on quite a few "real" problems. For example, it solves problem Y* of (120) in only 2.6 Gμ; that's much better than the 6.6 Gμ achieved by WTD, which was already a big improvement on MRV. Here are the complete scores, on all thirteen of the benchmark problems that we've been considering:

code name	(options, items, solutions)	MRV runtime	WTD runtime	FRB runtime	winner
C	(4320, 30+61, 1566720)	41.6 Gμ	54.3 Gμ	45.6 Gμ	MRV
D	(2327, 77+1, 16146)	12.4 Gμ	21.3 Gμ	12.8 Gμ	MRV
H	(1416, 196+93, 5224)	407.4 Gμ	19.6 Gμ	34.8 Gμ	WTD
J	(264, 144+0, 1)	50.3 Gμ	0.8 Mμ	1.9 Mμ	WTD
K	(343, 49+288, 110968)	4.1 Gμ	2.7 Gμ	2.8 Gμ	WTD
L	(352, 48+0, 326721800)	814.7 Gμ	1080.5 Gμ	1126.1 Gμ	MRV
M	(1514, 49+42, 987816)	20.6 Gμ	19.1 Gμ	8.1 Gμ	FRB
O*	(6966, 180+0, 16928)	7090.2 Gμ	8363.8 Gμ	6104.5 Gμ	FRB
Q	(256, 32+58, 14772512)	43.9 Gμ	66.6 Gμ	65.6 Gμ	MRV
R	(121, 11+741, 401800)	2.9 Gμ	3.3 Gμ	3.5 Gμ	MRV
S	(3858, 342+90, 30258432)	125.9 Gμ	149.9 Gμ	147.5 Gμ	MRV
U	(2440, 72+0, 31520)	119.1 Gμ	189.1 Gμ	124.8 Gμ	MRV
W	(1212, 12+36, 352)	7.4 Gμ	10.3 Gμ	9.2 Gμ	MRV
Y*	(949, 205+276, 16)	23.6 Gμ	6.6 Gμ	2.6 Gμ	FRB

(125)

Here MRV means Algorithm C$^+$; WTD and FRB are variants of Algorithm B.

This list includes also a fourteenth problem, Problem J, which isn't really real: Problem J is the toy problem that we get by taking 24 independent copies of the options 7.2.2.1–(92), for which MRV has "bad focus." (The corresponding runtime for dancing links is 27.4 Gμ.) It illustrates the fact that WTD and FRB both help to maintain a good focus.

***Maintaining XCC supports.** All of the results of (125) were obtained by using forward consistency to prune the search: Whenever an option o was included in a partial solution, all options o' that were incompatible with o were excluded from the remaining subproblem. Some of those problem instances could have been solved with a much smaller search tree, if full domain consistency had been used to look ahead further at each step. For example, the 461-meganode tree that's implicitly traversed by the FRB solution of Problem O* could have been reduced to only 7.2 meganodes — a 64-fold reduction! But the extra time needed per node to maintain DC in that problem would have more than canceled the advantage of fewer nodes; the total runtime would have risen from 6.1 Tμ to 7.3 Tμ.

There are, of course, classes of difficult problems for which DC maintenance does give a winning strategy, and we naturally want to solve those problems as efficiently as we can. Algorithm S below achieves that goal, by adding further data structures and mechanisms to the dancing cells technology.

Recall from (92) that we write $o \parallel o'$ when options o and o' of an XCC problem are compatible. It means that o and o' are either equal or they have no items in common, except for secondary items with identical nonnull colors.

In order to maintain DC, we must remove an option o from consideration whenever the addition of o to the current partial solution would cause some active item $i \notin o$ to lose all of its current options.

benchmark problems
dancing links
focus
forward consistency
incompatible
domain consistency
compatible

support matrix
domain
purging

A nice way to understand the task at hand is to imagine a giant "support matrix" $S[o,i]$, which has one row for every active option o and one column for every active primary item i. If o is one of i's options (that is, if $i \in o$), we set $S[o,i]$ to the special symbol #. Otherwise $S[o,i]$ should be some option o' such that $o \parallel o'$ and $i \in o'$. Such an option is a *support* for o and i, namely a witness to the fact that option o can appear in a solution without wiping out the domain of item i, which is the set of i's available options. It's easy to see that a set of XCC options is domain consistent if and only if there exists a support matrix S for which all of the non-# entries $S[o,i]$ are appropriate options o'.

For example, let's consider again the small XCC problem (113), with its primary items $\{$p, q, r$\}$ and secondary items $\{$x, y$\}$. To make it more interesting, we shall add an additional option 'r y:A'. Then we can almost — but not quite — construct a support matrix for the resulting six options:

	p	q	r
00 'p q x y:A'	#	#	19 'r y:A'
05 'p r x:A y'	#	13 'q x:A'	#
10 'p x:B'	#		16 'r y:B'
13 'q x:A'	05 'p r x:A y'	#	05 'p r x:A y'
16 'r y:B'	10 'p x:B'	13 'q x:A'	#
19 'r y:A'	10 'p x:B'	13 'q x:A'	#

(126)

(Each option has been given a two-digit identifying number, for convenience, based on its position in Table 3. Thus we can speak of options $\{00, 05, 10, 13, 16, 19\}$ instead of spelling them out.) We have, for instance, $S[05,\mathtt{q}] = 13$; and 13 is indeed a support for $(05,\mathtt{q})$ because $05 \parallel 13$ and $\mathtt{q} \in 13$.

Unfortunately, (126) contains an unavoidable "hole" in position $S[10,\mathtt{q}]$. There is no option compatible with 10 that contains q. Therefore the options aren't domain consistent; we must delete option 10 from the domain of p.

Deleting an option is called "purging"; it makes that option inactive.

After 10 has been purged, we cannot use it in the support matrix. So $S[19,\mathtt{p}]$ has to be changed. No problem: We can set $S[19,\mathtt{p}] \leftarrow 00$.

But $S[16,\mathtt{p}]$ must also be changed; and that's impossible. Hence option 16 must also be purged. This leaves us with a valid S, establishing DC:

	p	q	r
00 'p q x y:A'	#	#	19 'r y:A'
05 'p r x:A y'	#	13 'q x:A'	#
13 'q x:A'	05 'p r x:A y'	#	05 'p r x:A y'
19 'r y:A'	00 'p q x y:A'	13 'q x:A'	#

(127)

Algorithm S doesn't actually represent the support matrix directly; it represents the inverse function instead: For each option o', we maintain a list of all the pairs (o,i) for which $S[o,i]=o'$. This list is called the *trigger stack* of o', because we use it to maintain the support conditions. If option o' becomes inactive for any reason, thereby leaving one or more holes in S, its loss will trigger a series of events that will refill those holes, one by one.

Each option o also has a *fixit stack*, containing all pairs (o',i) for which the event (o,i) has been triggered by o' and the corresponding hole is still unfilled.

There's also a queue Q, containing all options whose fixit stack is nonempty.

In (127), for example, the trigger stack of 13 is $(05,\mathtt{q})\,(19,\mathtt{q})$; all fixit stacks are empty, and so is Q. At this point the algorithm might want to consider the subproblem in which option 13 is removed; that would push $(13,\mathtt{q})$ onto the fixit stacks of 05 and 19, also inserting 05 and 19 into Q. The hole in 05 can't be filled; therefore we'll have to purge option 05. (Its trigger stack $(13,\mathtt{p})\,(13,\mathtt{r})$ won't trigger any new events, because option 13 is no longer active.) To fill the hole in 19, we implicitly set $S[19,\mathtt{q}]\leftarrow 00$, by pushing $(19,\mathtt{q})$ onto 00's trigger stack. The queue is now empty; hence we've established DC for $\{00,19\}$.

The support matrix is huge. But fewer and fewer portions of it are relevant as we get into deeper and deeper levels of the search, because we need supports only for the active options and the active items.

Suppose we're currently operating in stage s, having chosen mutually compatible options c_1, …, c_s to be part of a solution. Then the set I_s of currently active items is the set of all items that don't appear in $c_1\cup\cdots\cup c_s$. (And that's the same as the set $\{\mathtt{ITEM}[k]\mid 0\le k<\mathtt{ACTIVE}\}$.)

The set O_s of currently active *options* is a bit trickier to characterize. Let O_{-1} be the set of all options that were present in the original problem. Algorithm S will begin by reducing them, if necessary, to $O_0^{\rm init}$, which is the largest subset of O_{-1} that is domain consistent, and it will enter stage 0. And $O_r^{\rm init}$, for $r>0$, will be the set of all options that were active when we most recently chose c_r and entered stage r. As we continue to work in stage r without backtracking to a previous stage, the set O_r begins as $O_r^{\rm init}$ and gradually shrinks as we return from exploring fresh choices of c_{r+1}. This leads to an interesting dynamic nested structure when we're currently in stage s:

$$O_{-1}\supseteq O_0^{\rm init}\supseteq O_0\supset O_1^{\rm init}\supseteq O_1\supset\cdots\supset O_s^{\rm init}\supseteq O_s. \qquad (128)$$

Here every set $O_r^{\rm init}$ and O_r is domain consistent, for $0\le r\le s$.

An option can become inactive in four different ways: It can be (i) *chosen*, that is, c_r for some r; or (ii) *blocked*, that is, incompatible with c_r when c_r was chosen; or (iii) *removed*, that is, no longer c_{r+1} when backtracking to stage r; or (iv) *purged*, that is, taken out of consideration because it has no active support. (Forward consistency deactivates options only in the first three ways.)

Every option is assigned an "age" whenever it is deactivated. Option c_r and any options that it blocks get age $2r-1$; when c_{r+1} is removed after backtracking, its age decreases from $2r+1$ to $2r$; and purged options inherit the age of the most recently deactivated option. Options of $O_{-1}\setminus O_0^{\rm init}$, which were purged at

trigger stack
fixit stack
queue
active items and options
domain consistent
chosen
blocked
removed
purged
Forward consistency
age

sort
undoing
T_0
first item of every option
stamping
domain consistency

the beginning before entering stage 0, have age -1. Consequently

$$\begin{aligned} O_{r-1} &= \{o \mid \texttt{AGE}(o) \geq 2r-1\}, \\ O_r^{\text{init}} &= \{o \mid \texttt{AGE}(o) \geq 2r\}, \end{aligned} \qquad \text{for } 0 \leq r \leq s, \tag{129}$$

if we regard $\texttt{AGE}(o)$ as infinite when o is currently active. (See exercise 336. The ages of active options are not actually stored in memory.)

Most of the work of Algorithm S is done by two subroutines, opt_out(o) and empty_q(), which are presented in exercises 339 and 340 as Algorithms O and E, respectively. The task of opt_out(o) is to deactivate a given option, possibly leaving holes in the support matrix; if holes do appear, their locations are recorded in Q and the fixit stacks. The task of empty_q() is to fill all of the remaining holes. Algorithm E might call Algorithm O as a subroutine, but Algorithm O never calls Algorithm E. Either algorithm might fail, if a contradiction is detected; in such a case it will terminate unsuccessfully, after repairing any inconsistencies that may have arisen in the data structures.

The main point of interest, with respect to those subroutines, is that a naïve approach to Algorithm O turns out to be much too slow, because the trigger stacks are full of irrelevant information about inactive items and options. (The support matrix for a problem with M options and N items has nearly MN non-# entries; hence the average length of each trigger stack is nearly N.) The remedy is to sort the trigger stacks by age of their entries, thereby making it possible to avoid looking at unimportant data about various supports that are known to be OK. This requires a rather elaborate mechanism, because partial re-sorting is constantly necessary as options change their age. The good news is that we don't have to worry about undoing changes that were made to S; *any* support, $S[o,i]$, remains a support when we backtrack. The resulting improved procedure, Algorithm O^+, is a marvel to behold (exercise 344).

Here now is the chef-d'œuvre for which we've been preparing ourselves:

Algorithm S (*XCC with supports*). This algorithm solves the same problems as Algorithm C; but it "looks ahead" by purging unsupported options that cannot be part of a solution. It uses auxiliary arrays $x_0x_1\ldots x_T$, $y_0y_1\ldots y_{T_0}$, $d_0d_1\ldots d_T$, `TRAIL`, and `LS` as in Algorithm B, as well as linked lists for the special data structures described in exercise 339. Also $\texttt{SS}[s]$ for $0 \leq s < T_0$. Variable `A` denotes the current age.

S1. [Initialize.] Perform step C1 of Algorithm C, ensuring also that the first item of every option is primary. Set `LAST` to the final value of x in Algorithm I (exercise 311). Set $\texttt{TRIG}(o) \leftarrow \texttt{FIX}(o) \leftarrow 0$ for every option o; also $\texttt{STAMP} \leftarrow \texttt{SSTAMP} \leftarrow 0$, $s \leftarrow l \leftarrow -1$. Perform Algorithm A (exercise 341) to establish domain consistency. Terminate if it detects inconsistent input; otherwise use exercise 342 to tidy up the trigger stacks.

S2. [Enter new stage.] Set $s \leftarrow s+1$; increase `SSTAMP` (see exercise 346); and set $\texttt{SS}[s] \leftarrow \texttt{SSTAMP}$.

S3. [Enter new level.] Set $l \leftarrow l+1$ and $\texttt{LS}[s] \leftarrow l$. Terminate with level overflow if $l > T$ (there's no room to store x_l).

MRV
forced move
stamping
Gμ
gigamems
Mν
meganodes
nodes
heuristic functions
forward consistency

S4. [Choose i.] Set $i \leftarrow$ `ITEM[`k`]` for some k with $0 \le k <$ `ACTIVE` and `ITEM[`k`]` $<$ `SECOND`. But if no such k exists, go to S8. The chosen i need not minimize `SIZE(`i`)`; however, if `SIZE(`i`)` > 1, there must be no forced move, that is, no active primary item with `SIZE(`i`)` $= 1$.

S5. [Trail the sizes.] Set $y_s \leftarrow t$ and $d_l \leftarrow$ `SIZE(`i`)`. If $d_l = 1$, go to S6. Otherwise terminate with trail overflow if $t +$ `ACTIVE` exceeds the maximum available `TRAIL` size. Otherwise set `TRAIL[`$t+k$`]` $\leftarrow$ (`ITEM[`k`]`, `SIZE(ITEM[`k`])`) for $0 \le k <$ `ACTIVE`; then set $t \leftarrow t +$ `ACTIVE`.

S6. [Try `SET[`i`]`.] Set $x_l \leftarrow$ `SET[`i`]` and `A` $\leftarrow 2s + 1$. Use the algorithm of exercise 347 to block all options incompatible with x_l and to choose option x_l. Then call empty_q() (exercise 340), and return to S2 if successful.

S7. [Try again.] Go to S9 if $d_l = 1$, otherwise to S10.

S8. [Visit a solution.] Visit the solution specified by nodes $x_{\texttt{LS[}j\texttt{]}}$ for $0 \le j < s$.

S9. [Back up.] Terminate if $s = 0$. Otherwise set $s \leftarrow s - 1$, $l \leftarrow$ `LS[`s`]`, and repeat step S9 if $d_l = 1$.

S10. [Untrail the sizes.] For $y_s \le k < t$, set `SIZE(`i'`)` $\leftarrow s'$ if `TRAIL[`k`]` $= (i', s')$. Then set `ACTIVE` $\leftarrow t - y_s$, $t \leftarrow y_s$.

S11. [Remove x_l.] Set `A` $\leftarrow 2s$ and `OACTIVE` $\leftarrow$ `ACTIVE`. Call opt_out(x_l); go to S9 if it fails. Call empty_q(); go to S9 if it fails. Otherwise go back to S3. ▮

The 'opt_out' subroutine called in step S11 should use the improved Algorithm O$^+$ that is described in exercise 344. (That's the reason for `SS` and `SSTAMP`.)

Performance on benchmarks. "The proof of the pudding is in the eating," according to an ancient proverb. How well does Algorithm S work in practice? Well, we can look first at the problems already considered in (119) and (125):

name	best FC	DC-MRV	DC-WTD	DC-FRB	
C	MRV, 41.6 Gμ, 5.4 Mν	109.2 Gμ, 6.7 Mν	126.6 Gμ, 7.1 Mν	130.2 Gμ, 8.2 Mν	
D	MRV, 12.4 Gμ, 1.6 Mν	19.9 Gμ, 0.2 Mν	23.5 Gμ, 0.3 Mν	27.0 Gμ, 0.3 Mν	
H	WTD, 19.6 Gμ, 32.5 Mν	137.4 Gμ, 4.3 Mν	312.6 Gμ, 7.1 Mν	397.4 Gμ, 7.3 Mν	
K	MRV†, 3.1 Gμ, 10.6 Mν	28.1 Gμ, 3.1 Mν	20.7 Gμ, 2.2 Mν	96.8 Gμ, 6.6 Mν	
L	MRV, 814.7 Gμ, 4.0 Gν	7.0 Tμ, 2.8 Gν	8.0 Tμ, 3.0 Gν	12.5 Tμ, 3.7 Gν	
M	FRB, 8.1 Gμ, 15.7 Mν	73.5 Gμ, 10.4 Mν	67.5 Gμ, 10.3 Mν	333.6 Gμ, 19.3 Mν	
O	FRB*, 6.1 Tμ, 461.5 Mν	8.2 Tμ, 7.1 Mν	8.5 Tμ, 9.6 Mν	10.5 Tμ, 10.1 Mν	(130)
Q	MRV†, 40.2 Gμ, 193.0 Mν	208.5 Gμ, 121.0 Mν	307.7 Gμ, 137.8 Mν	384.7 Gμ, 158.7 Mν	
R	MRV, 2.9 Gμ, 1.6 Mν	4.4 Gμ, 1.9 Mν	4.6 Gμ, 2.0 Mν	4.9 Gμ, 2.0 Mν	
S	MRV, 125.9 Gμ, 548.0 Mν	2.7 Tμ, 568.5 Mν	7.5 Tμ, 691.7 Mν	3.2 Tμ, 667.8 Mν	
U	MRV, 119.1 Gμ, 17.1 Mν	210.6 Gμ, 1.7 Mν	255.9 Gμ, 2.2 Mν	393.4 Gμ, 2.8 Mν	
W	MRV, 7.4 Gμ, 1.7 Mν	12.0 Gμ, 0.8 Mν	18.5 Gμ, 1.2 Mν	27.0 Gμ, 1.7 Mν	
Y	FRB*, 2.6 Gμ, 1.3 Mν	7.7 Gμ, 54.9 Kν	4.5 Gμ, 40.1 Kν	5.8 Gμ, 42.0 Kν	

Here 'Gμ' stands for gigamems, as usual, while 'Mν' stands for *meganodes* — one million nodes in the search tree. The number of nodes is the total number of times that step S3 is executed (or an analogous step such as C3$^+$ or B10).

The first main column of (130) shows the shortest runtimes obtained with algorithms that use only forward consistency checks;* the other columns show various flavors of Algorithm S, using different heuristic functions in step S4.

* MRV† refers to dancing links, Algorithm 7.2.2.1C, while MRV refers to dancing cells, Algorithm C$^+$; WTD and FRB refer to the corresponding heuristics in Algorithm B (see (120)).

Preprocessing by Algorithm 7.2.2.1P has been used for the FC versions of Problems O and Y, but not for any of the DC versions. (There are occasional instances where preprocessing does turn out to be mildly helpful to Algorithm S, due to quirks of fate when branching. However, they're too rare to matter.)

One of the chief surprises in (130) is that FC sometimes gives a smaller search tree than DC does (Problems C, R, S). Again, quirks of fate are responsible: DC isn't always helpful, and FC can make lucky choices. On the other hand, DC makes an order of magnitude improvement in Problems H, O, U, and Y — most notably in Problem O, where there's a 65-fold reduction.

These statistics give us another reminder that there's tremendous variability between problems. The various ratios of mems per node in (130) are "all over the map," ranging from about 200 in the FC versions of Problems L, M, S to more than 100,000 in the DC versions of Problems U and Y, and a million in Problem O! The μ/ν ratios are roughly comparable for FC and DC in Problems C, R, W; but DC expends more than 60 times as many mems per node as FC does on Problems H, O, Y.

There seems to be only one thing consistently true about all thirteen of the experiments reported in (130), namely that FC was always better than DC. Sometimes it was marginally better (Problems D, O, R, Y); sometimes it was spectacularly better (Problems L, S); and it always was the method of choice.

Of course that's not the whole story! There also are tough problems that are challenging for FC but amenable to DC, and it's high time to look at them now:

code name	(options, items, solutions)	best FC runtime	DC-MRV runtime	DC-WTD runtime	DC-FRB runtime
A	(18486, 30+110, 8)	FRB$^{\dagger *}$, 59.1 Gμ	54.5 Gμ	13.0 Gμ	22.6 Gμ
E	(2536, 54+14, 89328)	FRB*, 33.2 Gμ	28.1 Gμ	55.6 Gμ	62.3 Gμ
F	(7800, 81+594, 1)	WTD*, 10.5 Gμ	158.3 Mμ	139.1 Mμ	149.5 Mμ
G	(576, 48+506, 8388608)	FRB, 41.7 Gμ	96.3 Gμ	87.6 Gμ	70.3 Gμ
I	(20088, 81+72, 16)	MRV*, 28.9 Gμ	999.5 Mμ	1.1 Gμ	1.0 Gμ
N	(5546, 17+668, 43)	FRB*, 77.4 Gμ	30.5 Gμ	13.5 Gμ	32.7 Gμ
P	(14179, 200+100, 3)	FRB*, 1.4 Tμ	3.0 Tμ	531.7 Gμ	4.3 Tμ
T	(2658, 29+338, 416)	FRB†, 4.9 Tμ	12.4 Tμ	5.8 Tμ	4.1 Tμ
V	(22000, 9+20, 32620)	FRB$^{\dagger *}$, 112.9 Gμ	65.7 Gμ	73.8 Gμ	81.0 Gμ
Z	(1104, 24+24, 575)	MRV, 203.7 Gμ	29.4 Mμ	29.9 Mμ	30.1 Mμ

(131)

Here Problem A is part of the "alphabet blocks" challenge in exercise 7.2.2.1–113, after all but one of the options for `FIRST` have been removed. Problem E finds the all-interval 14-tone rows, using the XCC model of exercise 7.2.2.1–103(b). Problem F solves the "fillomino π" puzzle of exercise 292(b). Problem G visits the slow growth permutations of order 24, using the options defined in exercise 287. Problem I fits nine different small-and-slim nonominoes into a 9×9 box (exercise 7.2.2.1–302). Problem N solves Nick Baxter's Square Dissection puzzle (exercise 7.2.2.1–359). Problem P is a 10×10 case of the "prime queen attacking" problem, discussed further below. Problem T comes from 'Torto' (exercise 7.2.2.1–112). Problem V finds all 4×5 word rectangles, using the 2000 most common 4-letter words of English together with `WORDS`(3000). And finally, Problem Z is an artificial benchmark discussed earlier, the $(23, 24)$-modstep problem, which was designed specifically to make DC look good.

Preprocessing
mems per node
alphabet blocks
all-interval
tone rows
music
n-tone rows
rows of musical tones
fillomino
π
slow growth permutations
small-and-slim nonominoes
slim nonominoes
nonominoes
Baxter
Square Dissection
prime queen attacking
queen attacking
Torto
word rectangles
4-letter words
`WORDS`(3000)
5-letter words
modstep problem

preprocessing
FC versus DC+
slow growth perms
prime queen attacking problem
Honaker
Keith
knight's tour
queen
prime numbers+
Tramu
author
reentrant knight's tour, see closed
closed
Jelliss

Twelve FC experiments lie behind each row of (131), namely the application of algorithms that we may call MRV†, MRV, WTD, FRB, WTD†, FRB†; MRV$^{\dagger *}$, MRV*, WTD*, FRB*, WTD$^{\dagger *}$, FRB$^{\dagger *}$. The dagger after MRV indicates dancing links, and the dagger after WTD or FRB indicates the d-ary variants in exercise 331; an asterisk indicates preprocessing. (However, only six experiments were needed for Problems G and Z, because preprocessing has no effect on the options of those cases.) For example, the twelve scores for Problem A were

$$(202.1, 168.8, 98.9, 1653.7, 77.6, 94.6;\ 202.0{+}10.2, 168.8{+}10.2,\\ 94.5{+}10.2, 1729.3{+}10.2, 77.6{+}10.2, 48.9{+}10.2)\ \mathrm{G}\mu,$$

where 10.2 Gμ was the preprocessing time. In this problem, FRB$^{\dagger *}$ was a clear winner and FRB* was a clear loser; WTD† was a close second.

The biggest surprise in (131) was the result of Problem G, whose six scores

$$(2999.1,\ 5405.7,\ 918.2,\ 41.7,\ 1129.1,\ 539.9)\ \mathrm{G}\mu$$

testified to a tremendous victory for the FRB heuristic, placing it ahead of all three variants of Algorithm S. Previous experiences with MRV methods had suggested that FC couldn't possibly do well with the options of Problem G.

DC was the champion, in all other cases of (131) — convincingly so, in Problems A, F, I, N, P, and of course Z. However, method FRB* unexpectedly turned out to be second best in Problems E and P.

Of all these instances, the most instructive is probably Problem P, which is based on the "prime queen attacking problem," proposed in 1998 by G. L. Honaker, Jr., and solved for $n \le 8$ by Michael Keith that same year. [See *Virginia Chess Newsletter* 1999 #1 (February 1999), 4–6.] The goal is to construct an $n \times n$ knight's tour, labeling the kth move with k for $1 \le k \le n^2$, and also to place a queen on some cell of the board, in such a way that the queen attacks as many prime numbers as possible. Here, for example, are solutions for $n = 10$

15 28 33 30 **13** 06 09 50 **97** 00	**47** 10 49 78 **67** 08 65 76 **73** 70	**59** 64 57 68 **71** 62 91 08 **73** 96
34 **31** 14 69 10 49 98 **05** 08 51	50 **79** 46 09 04 77 68 **71** 64 75	56 **67** 60 63 06 69 72 **97** 92 09
27 16 **29** 32 **71** 12 **07** 52 99 96	45 48 **11** 80 **07** 66 **03** 74 69 72	65 58 **03** 70 **61** 90 **07** 10 95 74
38 35 70 **11** 68 **41** 48 95 04 93	54 51 44 **13** 42 **05** 84 99 40 63	36 55 66 **89** **02** **05** 44 75 98 93
17 26 **37** 40 45 72 **03** 92 **53** 56	**19** 12 **53** 06 81 60 **41** **02** **97** 00	**19** 22 **37** 04 45 14 **11** 94 **43** 76
36 39 44 **67** **02** **47** 42 55 94 91	52 55 20 **43** 14 **83** 98 85 62 39	54 35 20 **17** 88 01 46 15 12 99
25 18 **23** 46 **43** 66 **73** 82 57 54	21 18 **23** 82 **59** 32 **61** 94 01 96	21 18 **23** 38 **47** 16 **13** 00 77 42
22 **61** 20 01 74 85 58 **79** 90 81	56 **89** 58 15 24 93 86 **29** 38 35	34 **53** 32 87 50 39 28 **79** 84 81
19 24 63 86 **59** 76 65 88 **83** 78	**17** 22 91 88 **31** 26 33 36 95 28	**31** 24 51 48 **29** 26 85 82 **41** 78
62 21 60 75 64 87 84 77 80 **89**	90 57 16 25 92 87 30 27 34 **37**	52 33 30 25 86 49 40 27 80 **83**

in which a queen near the center attacks all 25 of the primes ≤ 100. (Prime numbers are shown in bold; 00 is equivalent to 100.) The first of these was found by Jacques Tramu in 2004; the other two were found by the author in 2022 as he was writing the present section. The middle one adds a further constraint, namely that the tour should be *closed*: cells 00 and 01 should be a knight move apart. The rightmost one adds yet another constraint, suggested by George Jelliss: Every odd-numbered cell attacked by the queen must be either prime or 01. Both of these solutions were obtained with Algorithm S, using the straightforward XCC formulation that's discussed in exercise 348.*

(It's fun to watch the knight as it springs from 01 to 02 to $\cdots$ to 99 to 00 in these tours, because it must get perked up whenever it comes into prime-rich territory, yet stay out of contact during a run of composite numbers.)

For Problem P we add *further* constraints, thus making the knight's task almost impossible: First, we require that every power of 2, as well as the primes, must be attacked by the queen. (Thus, not only 02, but also 01, 04, 08, 16, 32, and 64 must be hit.) Second, we require that 00 appears in cell $(1,4)$, near the top middle. Third, we require that the first eight digits of π appear in fixed positions that make a nice pattern: 31, 41, 59, 26 must be in the respective cells $(4,2)$, $(5,3)$, $(6,4)$, and $(7,4)$. Amazingly, this problem turns out to be solvable, and it has exactly three solutions:

11	34	99	96	**71**	06	75	94	**73**	82
98	**37**	10	33	**00**	95	72	**83**	76	93
35	12	**97**	70	**07**	18	**05**	74	81	84
38	09	36	**17**	**32**	**01**	80	85	92	77
13	**16**	**31**	**08**	69	**04**	**19**	78	**89**	86
30	39	14	**41**	**02**	**79**	58	87	20	91
15	42	**47**	68	**59**	66	**03**	90	57	88
48	**29**	40	45	**26**	63	60	**23**	54	21
43	46	27	50	**67**	24	65	52	**61**	56
28	49	44	25	**64**	51	62	55	22	**53**

11	34	99	96	**71**	06	85	94	**73**	78
98	**37**	10	33	**00**	95	72	**79**	86	93
35	12	**97**	70	**07**	18	**05**	84	77	74
38	09	36	**17**	**32**	**01**	80	75	92	87
13	**16**	**31**	**08**	69	**04**	**19**	88	**83**	76
30	39	14	**41**	**02**	**89**	58	81	20	91
15	42	**47**	68	**59**	66	**03**	90	57	82
48	**29**	40	45	**26**	63	60	**23**	54	21
43	46	27	50	**67**	24	65	52	**61**	56
28	49	44	25	**64**	51	62	55	22	**53**

11	34	99	96	**79**	72	77	94	**67**	70
98	**37**	10	33	**00**	95	68	**71**	74	93
35	12	**97**	80	**05**	78	**73**	76	69	66
38	09	36	**17**	**32**	**01**	06	65	92	75
13	**16**	**31**	**08**	81	**04**	**19**	**02**	**61**	**64**
30	39	14	**41**	18	**07**	60	63	20	91
15	42	**47**	82	**59**	84	**03**	90	57	62
48	**29**	40	45	**26**	87	58	**23**	54	21
43	46	27	50	**83**	24	85	52	**89**	56
28	49	44	25	86	51	88	55	22	**53**

The options defined in exercise 348 aren't actually good enough to carry out an exhaustive search for all solutions to Problem P in a reasonable time, even though this extension of the prime queen attacking problem is very highly constrained. Fortunately, however, Peter Weigel has discovered a way to exploit the fact that the graph of knight moves is bipartite, leading to a refined XCC formulation that works considerably faster. Problem P therefore incorporates his improved options, which are explained in exercise 349.

Incidentally, the surprising performance of method FRB* on Problem P can be appreciated from the twelve scores that lie behind the result reported in (131):

$$(422.3, 295.0, 73.7, 10.8, 50.4, 11.0;\\ 29.8{+}.005, 21.2{+}.005, 2.7{+}.005, 1.4{+}.005, 2.2{+}.005, 2.0{+}.005)\ \mathrm{T}\mu.$$

We have, of course, only scratched the surface with respect to possible heuristics; further developments are likely to lead to even better results.

***Sparse-set methods for MCC problems.** Section 7.2.2.1 introduced a wide-ranging generalization of XCC problems called *multiple covering with colors*, or MCC for short. In an MCC problem we can, for example, insist that a particular primary item must appear in exactly five of the chosen options, not in exactly one option as in XCC. Each primary item i has in fact a designated *interval* $[u_i \mathinner{..} v_i]$ of multiplicities, governing the number of times it must appear in a solution.

* Indeed, the middle one, obtained after 6.4 Tμ of computation, was sort of "epic" for me: It was the first time I'd ever solved a problem with DC methods that I couldn't solve with FC!

composite numbers
Weigel
bipartite
MCC–
multiple covering with colors
XCC

Of course MCC problems can be enormously difficult, even harder than XCC problems. But we learned in Algorithm 7.2.2.1M that dancing links technology can solve lots of important examples. That algorithm incorporates an additional dance step called "tweaking," 7.2.2.1–(69), which can be viewed as a way to switch from the d-way branching of Algorithm 7.2.2.1C to binary branching.

Filip Stappers demonstrated in 2023 that MCC problems are amenable also to dancing *cells* technology. In fact, he extended Algorithm B to Algorithm F (see exercise 352), which usually outperforms the algorithm of Section 7.2.2.1(!).

Let's pause a moment to define MCC problems more formally. We're given a set O of *options*, each of which is a set of *items*. Items are either primary or secondary; secondary items have *colors*. An interval $[u_i \,.\,.\, v_i]$ is specified for every primary item i, where $u_i \le v_i$ and $v_i > 0$. Two options are *compatible* if their secondary items are colored in the same way. A *solution* is a subset $S \subseteq O$ of mutually compatible options, for which each primary item i occurs in at least u_i and at most v_i of S's options. Every option must include at least one primary item. An XCC problem is the special case where $u_i = v_i = 1$ for all i.

(It often happens that a particular color occurs only once with a particular item, in the entire set O. Such unmatchable colors are conventionally left blank, instead of being given an explicit name. Thus, if secondary item i is blank in two different options, those options aren't compatible.)

The design of Algorithm F, like its precursor Algorithm 7.2.2.1M, is essentially recursive. We choose, in some fashion, an option $o \in O$, and make a two-way branch: Either $o \in S$ or $o \notin S$. Each branch reduces our job to an MCC subproblem that's simpler than the original one. Eventually we get to a subproblem that is obviously solvable (because $O = \emptyset$ and all items are properly covered), or a subproblem that obviously has no solution (because some primary item i has fewer than u_i remaining options).

As in Algorithm C above, we let $\mathtt{SIZE}(i)$ denote the number of options that contain item i in the current subproblem. And as in 7.2.2.1–(72), we maintain auxiliary quantities $\mathtt{SLACK}(i)$ and $\mathtt{BOUND}(i)$, where

$$\mathtt{SLACK}(i) = v_i - u_i \qquad \text{and} \qquad \mathtt{BOUND}(i) = v_i. \tag{132}$$

The value of $\mathtt{SLACK}(i)$ remains unchanged throughout the computation; but $\mathtt{BOUND}(i)$ decreases by 1 whenever we've included an option containing i into the partial solution S. (This policy means that we'll be working on subproblems for which $u_i < 0$, whenever the current upper bound $v_i = \mathtt{BOUND}(i)$ has become less than $\mathtt{SLACK}(i)$. But a negative lower bound doesn't cause any trouble.)

The input to Algorithm F is a list of the given problem's items and their multiplicities, followed by the problem's options. It might turn out that $\mathtt{SIZE}(i) = 0$ for some item i, namely that i doesn't show up in any of the options; that makes the specifications unsatisfiable if i is primary and $u_i > 0$. But otherwise such a scenario is perfectly legitimate, and we simply make i inactive, hence invisible, in such cases. Algorithm F is careful to ensure that $\mathtt{SIZE}(i)$ remains nonzero for all other items i, throughout the rest of the computation.

dancing links
tweaking
d-way branching
binary branching
Stappers
dancing *cells*
options
items
primary
secondary
colors
compatible
XCC problem
recursive
branching–
$\mathtt{SLACK}$ field+

monus operation
MRV
forced move
partridge puzzle
balls
disks
pentominoes
balanced
5-queens
domination
piles

How do we choose an option o on which to branch? Algorithm F follows the lead of Algorithm B, and chooses a primary *item*, i, on which to branch. Then o is `SET[`i`]`, the first option in i's current list of options.

OK then, how do we choose a primary *item* i on which to branch? Suppose, for example, that i currently appears in $\mathtt{SIZE}(i) = 5$ options $\{o_1, o_2, o_3, o_4, o_5\}$, and that $\mathtt{SLACK}(i) = 1$, $\mathtt{BOUND}(i) = 4$; the problem requires us to include either three or four of those five options in the eventual solution S. The first option to be included must therefore be either o_1 or o_2 or o_3; we'll fail if we omit all three. Hence we're faced with a 3-way decision about how to select the first option.

In general, as observed in exercise 7.2.2.1–166(a), we're faced with a d_i-way decision, where

$$d_i \;=\; \mathtt{SIZE}(i) + 1 \,-\, \bigl(\mathtt{BOUND}(i) \mathbin{\dot-} \mathtt{SLACK}(i)\bigr) \qquad (133)$$

and '$\dot-$' is the "monus operation," $x \mathbin{\dot-} y = \max(x-y, 0)$. Algorithm F takes care to ensure not only that $\mathtt{SIZE}(i) > 0$, as mentioned above, but also that $d_i > 0$. One way to choose i is to adopt the MRV strategy, which selects an item for which the branching degree d_i is a small as possible.

Notice that a "forced move" arises when $d_i = 1$, namely when $\mathtt{SIZE}(i) = \mathtt{BOUND}(i) - \mathtt{SLACK}(i)$, because $\mathtt{SIZE}(i) > 0$. This means that all $\mathtt{SIZE}(i)$ of i's current options must be included in S; otherwise we wouldn't satisfy the lower bound $u_i = \mathtt{BOUND}(i) - \mathtt{SLACK}(i)$. (This analysis generalizes the forced-move condition of XCC problems, where $\mathtt{BOUND}(i) = 1$ and $\mathtt{SLACK}(i) = 0$; in Algorithms B and C, a move for i was forced if and only if $\mathtt{SIZE}(i) = 1$.)

Full implementation details are in exercise 352. So let's look at some results:

code name	(options, items, solutions)	dancing links (MRV)	dancing cells (MRV)	dancing cells (WTD)	dancing cells (FRB)
$\mathcal{A}$	(811, 202+0, 60568)	58.6 Gμ	49.2 Gμ	26.1 Gμ	61.5 Gμ
$\mathcal{B}$	(77, 97+0, 1)	222.8 Gμ	99.9 Gμ	37.9 Gμ	133.0 Gμ
$\mathcal{C}^{\neg\sharp}$	(4068, 132+0, 5347)	4607.2 Gμ	4080.2 Gμ	6774.6 Gμ	2646.0 Gμ
$\mathcal{D}^{\sharp}$	(64, 65+0, 4860)	4.2 Gμ	17.4 Gμ	17.7 Gμ	17.7 Gμ
$\mathcal{E}$	(1393, 61+0, 10343858)	2267.3 Gμ	2168.5 Gμ	2344.3 Gμ	2055.7 Gμ
$\mathcal{H}^{\neg\sharp}$	(1335, 15+61 720)	6.9 Gμ	6.2 Gμ	7.8 Gμ	8.5 Gμ
$\mathcal{M}$	(1504, 88+102, 696)	199.4 Gμ	159.0 Gμ	200.9 Gμ	120.9 Gμ
$\mathcal{N}^{\neg\sharp}$	(256, 2700+58, 71486)	1786.8 Gμ	87.4 Gμ	134.5 Gμ	140.6 Gμ
$\mathcal{P}$	(2436, 1730+0, 112)	438.7 Gμ	354.4 Gμ	991.1 Gμ	379.9 Gμ
$\mathcal{Q}^{\sharp}$	(3940, 65+126, 512)	284.0 Gμ	138.0 Gμ	138.7 Gμ	138.7 Gμ
$\mathcal{R}^{\neg\sharp}$	(13052, 36+46, 6)	28.1 Gμ	20.8 Gμ	20.6 Gμ	17.3 Gμ
$\mathcal{S}$	(4038, 132+0, 98)	281.0 Gμ	297.4 Gμ	408.7 Gμ	183.9 Gμ
$\mathcal{T}$	(1740, 280+400, 8)	1081.4 Gμ	1256.4 Gμ	504.2 Gμ	283.0 Gμ
$\mathcal{W}$	(2071, 447+0, 0)	6.0 Gμ	4.7 Gμ	3048.4 Gμ	10.4 Gμ
$\mathcal{X}^{\sharp}$	(576, 115+128, 4)	550.2 Gμ	361.3 Gμ	158.9 Gμ	411.0 Gμ

(134)

Here Problem $\mathcal{A}$ is the "authentic" partridge puzzle (exercise 7.2.2.1–155) with $n = 6$. Problem $\mathcal{B}$ covers an 8×12 grid with 10 two-dimensional balls of diameter 4 (see exercise 354). Problem $\mathcal{C}$ covers the diagonals of a 10×10 grid with the twelve pentominoes, in a nicely balanced fashion (exercise 7.2.2.1–300(b)). Problem $\mathcal{D}$ is the classic 5-queens domination problem: 7.2.2.1–(64) with $(m, n) = (5, 8)$. Problem $\mathcal{E}$ piles all twelve pentominoes on a 7×7 board, allowing multiplicities $[1\,..\,2]$ at the edges (see exercise 355). Problem $\mathcal{H}$ packs eleven hypersolid pentominoes — all but the V — into a $2 \times 2 \times 3 \times 5$ hypercube (exercise 7.2.2.1–352). Problem $\mathcal{M}$ enumerates the motley dissections of a 6×12 rectangle (exercise 7.2.2.1–369). Problem $\mathcal{N}$ solves the 16 queens problem with no-three-in-a-line (see exercise 357). Problem $\mathcal{P}$ solves the Perfect Packing puzzle (exercise 7.2.2.1–350). Problem $\mathcal{Q}$ fits five $\mathcal{Q}_6$ configurations into a $\mathcal{Q}_{32}$ (exercise 7.2.2.1–162(i)). Problem $\mathcal{R}$ finds the 4×5 word rectangles with fewest distinct letters, 7.2.2.1–(66). Problem $\mathcal{S}$ achieves central symmetry in the blank regions of a balanced 10×10 pentomino pattern (exercise 7.2.2.1–300(c)). Problem $\mathcal{T}$ discovers Tullis's remarkable tapestry (see exercise 362). Problem $\mathcal{W}$ is Wainwright's original partridge puzzle (exercise 7.2.2.1–157) with $n = 6$. And Problem $\mathcal{X}$ finds all ways to put exactly $(12, 12, 4)$ words of lengths $(3, 4, 5)$ into an 8×8 crossword diagram (exercise 7.2.2.1–111(a)).

The notation '$\mathcal{D}^\sharp$' means that Problem $\mathcal{D}$ was solved with the "sharp preference heuristic" of exercise 7.2.2.1–10. (A primary item whose name begins with # is chosen for branching, unless some other primary item has a forced move.) Similarly, '$\mathcal{C}^{\neg\sharp}$' calls for the analogous "nonsharp preference heuristic." In each problem we've used the preference heuristic that wins for dancing links.

The results exhibited in (134) are, of course, just the "tip of an iceberg," because many other strategies for choosing an option on which to branch are clearly possible, and because many different flavors of problems exist. We can expect that a portfolio of complementary techniques will continue to evolve, as more and more people discover the wondrous world of MCC-solving.

On the other hand, many of the problems that we've been discussing lie at the edge of what is computationally feasible with known methods; slight extensions of those problems might forever be hopelessly difficult.

Tractable families of CSPs. But there's good news too. Despite the fact that many flavors of CSP are NP-hard in general, researchers have found polynomial-time algorithms for a substantial number of special cases that arise in practice.

For example, a CSP can be solved quickly if the network of dependencies between variables isn't too complicated. If, say, all constraints are binary, and the graph of constraints is a tree or becomes a tree after we've assigned values to just a few variables, we're in luck (see exercise 366). In future sections we shall study concepts such as "treewidth," which explain how large problems can often be decomposed into manageable pieces.

Another class of CSPs, which we shall investigate now, is able to succeed even when the variables of the problem are entangled in complex ways, because we know that all of the individual constraints have a special form.

Suppose, for instance, that every variable has the same domain, and that the elements of that domain form a finite field. (They might be the integers $\{0, 1, \ldots, p-1\}$, where p is prime, in which case we can add, subtract, multiply, and divide them modulo p.) If every constraint is a linear equation between variables, something like '$x + 3y - 2z = 7$', the CSP is just a system of simultaneous linear equations; so we can solve it by standard methods such as Gaussian elimination. (See, for example, exercise 7.2.1.3–12 and Algorithm 4.6.2N.)

hypersolid pentominoes
motley dissections
16 queens problem
no-three-in-a-line
Perfect Packing puzzle
word rectangles
balanced
pentomino
Tullis
tapestry
Wainwright
partridge puzzle
crossword diagram
sharp preference heuristic
nonsharp preference heuristic
tractable–
treewidth
finite field
simultaneous linear equations
Gaussian elimination

Implicational constraints. Even when the variables of a CSP are interrelated in ways that have nothing to do with ordinary arithmetic, we can still solve the problem with guaranteed efficiency if every constraint between them is sufficiently tame. One such family of algorithm-friendly dependencies is the class of *implicational constraints*, which are binary relations of a particularly simple kind.

implicational constraints–
complete relation
correspondence
two-fan
matrices of 0s and 1s
matrix representation of binary relation
0/1/all
consistency
unary constraint

Suppose v and w are variables whose domains are D_v and D_w, and let A and B be designated subsets of D_v and D_w. A binary relation between v and w is called "implicational" if it is either

i) a *complete relation*: "$v \in A$ and $w \in B$"; or
ii) a *correspondence*: "$v \in A$ and $w = f(v)$," where f is a one-to-one correspondence between A and B; or
iii) a *two-fan*: "$v = a$ and $w \in B$, or $v \in A$ and $w = b$," where a and b are designated elements of A and B.

For example, Fig. 119 illustrates each of these three possibilities, using matrices of 0s and 1s to represent the relations. An implicational constraint is sometimes called '0/1/all', because every row of its matrix either has no 1, or one 1, or 1s in all columns of B; every column either has no 1, or one 1, or 1s in all rows of A.

$$\begin{pmatrix} 0&1&0&0&1&1\\ 0&0&0&0&0&0\\ 0&1&0&0&1&1\\ 0&1&0&0&1&1\\ 0&1&0&0&1&1 \end{pmatrix} \qquad \begin{pmatrix} 0&0&0&1&0&0\\ 0&1&0&0&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&0\\ 0&0&1&0&0&0 \end{pmatrix} \qquad \begin{pmatrix} 0&1&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&0&0&0&0\\ 1&1&0&1&1&1\\ 0&0&0&0&0&0 \end{pmatrix}$$

complete — $A = \{0, 2, 3, 4\}$, $B = \{1, 4, 5\}$

correspondence — $f(0) = 3$, $f(1) = 1$, $f(2) = 4$, $f(4) = 2$

two-fan — $A = \{0, 1, 3\}$, $a = 3$, $B = \{0, 1, 3, 4, 5\}$, $b = 1$

Fig. 119. Typical examples of implicational constraints, when $D_v = \{0, 1, 2, 3, 4\}$ and $D_w = \{0, 1, 2, 3, 4, 5\}$. A matrix entry is 1 if and only if v and w satisfy the relation.

Notice that an implicational relation between v and w is also an implicational relation between w and v, if we interchange the roles of A and B, a and b. Degenerate cases are worthy of note: If $|A| \le 1$ or $|B| \le 1$, the relation will be regarded as *complete*, even though it might also qualify as a correspondence or a two-fan.

CSPs that are based on implicational constraints can not only be solved efficiently; they're actually among the best we can hope for, in an asymptotic sense:

Theorem I. *Every CSP $\mathcal{P}$ whose constraints all are implicational can be solved or proved unsatisfiable in linear time.*

Proof. First we'll look at an algorithm; then we'll explain precisely why its running time is at worst proportional to the length of the given CSP.

The algorithm begins by refining the domains, to establish a weak kind of consistency: Every variable v has an initially specified domain D_v, which is essentially a unary constraint; and every binary constraint between v and w has specified subsets A_{vw} and B_{vw}. We shall set $D_v \leftarrow D_v \cap A_{vw}$ and $D_w \leftarrow$

strong components
Tarjan
bad strong component
topological order

$D_w \cap B_{vw}$. Every solution to $\mathcal{P}$ will remain a solution with respect to these new domains, and every constraint will still be implicational. (See exercise 368.)

If any D_v is empty, problem $\mathcal{P}$ has no solution. Otherwise we construct a digraph G, whose vertices are all pairs (v, a) where $a \in D_v$, together with a special vertex '$\perp$'. There's an arc $(v,a) \longrightarrow (w,b)$ if and only if the constraint between v and w is either a correspondence with $b = f_{vw}(a)$ or a two-fan with $a \neq a_{vw}$ and $b = b_{vw}$. (Such an arc means, "$v = a$ implies $w = b$.") There's an arc $(v,a) \longrightarrow \perp$ if and only if there's either a correspondence with $f_{vw}(a) \notin D_w$ or a two-fan with $a \neq a_{vw}$ and $b_{vw} \notin D_w$. (Such an arc means, "$v = a$ is disallowed.")

For example, suppose $\mathcal{P}$ has four variables $\{w, x, y, z\}$, with domains $D_w = D_x = \{0,1,2,3,4\}$ and $D_y = D_z = \{0,1,2,3,4,5\}$. Suppose further that there are four constraints, C_{wy}, C_{wz}, C_{xy}, C_{xz}, all of which appear in Fig. 119: C_{wz} and C_{xy} are copies of the complete constraint; C_{wy} is the correspondence; and C_{xz} is the two-fan. Domain reduction (see exercise 369) gives

$$D_w = \{0,2,4\}, \quad D_x = \{0,3\}, \quad D_y = \{1,4\}, \quad D_z = \{1,4,5\}. \qquad (135)$$

Therefore G has $3+2+2+3+1$ vertices, $\{(w,0),(w,2),\ldots,(z,5),\perp\}$. The arcs for the correspondence C_{wy} (and for its converse, C_{yw}) are

$$(w,0) \longrightarrow \perp,\ \ (w,2) \longrightarrow (y,4),\ \ (w,4) \longrightarrow \perp;\ \ (y,1) \longrightarrow \perp,\ \ (y,4) \longrightarrow (w,2); \qquad (136)$$

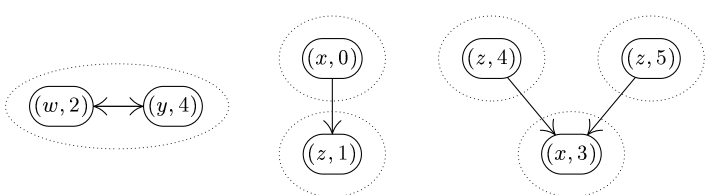
the arcs for the two fan C_{xz} (and for C_{zx}) are

$$(x,0) \longrightarrow (z,1); \quad (z,4) \longrightarrow (x,3), \quad (z,5) \longrightarrow (x,3). \qquad (137)$$

Once G has been constructed, we find its *strong components* using Algorithm 7.4.1.2T (Tarjan's algorithm on page viii). Every such component groups together the sets of assignments that mutually imply each other; and if we shrink those components to single points, the resulting digraph $\widehat{G}$ is acyclic.

One of the components will be the solitary vertex $\perp$; we call it *bad*. A component is also called bad if it contains more than one assignment to the same variable, or if it points to a bad component. Tarjan's algorithm is readily adapted so that it will automatically remove all of the bad components from $\widehat{G}$.

In our example, with G defined by (136) and (137), the acyclic digraph $\widehat{G}$ is



(138)

after the bad components $\{\perp\}$, $\{(w,0)\}$, $\{(w,4)\}$, $\{(y,1)\}$ have been discarded.

Finally, we can assign consistent values to variables of $\mathcal{P}$ by traversing $\widehat{G}$ in reverse topological order as follows: "Choose any component $\{(v_1,a_1),\ldots,(v_t,a_t)\}$ that has no outgoing arcs in $\widehat{G}$. If v_1 has not already been assigned a value, set $v_j \leftarrow a_j$ for $1 \le j \le t$. (Variables v_2, ..., v_t won't have been set yet either; see exercise 370.) Delete this component, and repeat until $\widehat{G}$ is empty."

We could, for instance, deal with (138) by always choosing the rightmost sink component, as follows: Set $x \leftarrow 3$ and delete $\{(x,3)\}$; set $z \leftarrow 5$ and delete $\{(z,5)\}$; delete $\{(z,4)\}$, $\{(z,1)\}$, $\{(x,0)\}$, since z and x have already been set; finish with $w \leftarrow 2$ and $y \leftarrow 4$. We've found a solution to the example problem.

If problem $\mathcal{P}$ is satisfiable, exercise 370 proves that this traversal procedure will find a solution. Conversely, if the traversal ends without giving values to all of the variables, problem $\mathcal{P}$ is unsatisfiable. The key fact that makes everything work is the following crucial property: "*If the digraph G contains a path*

$$(v_1,a_1)\longrightarrow(v_2,a_2)\longrightarrow\;\cdots\;\longrightarrow(v_{t-1},a_{t-1})\longrightarrow(v_t,a_t), \tag{139}$$

and if $a'_t \neq a_t$ is another element of D_{v_t}, then G also contains a path

$$(v_t,a'_t)\longrightarrow(v_{t-1},a'_{t-1})\longrightarrow\;\cdots\;\longrightarrow(v_2,a'_2)\longrightarrow(v_1,a'_1), \tag{140}$$

for some values $a'_1 \neq a_1$, $a'_2 \neq a_2$, $\ldots$, $a'_{t-1} \neq a_{t-1}$." This property clearly holds for $t = 2$, regardless of whether the constraint between v_{t-1} and v_t is a connection or a two-fan; and for $t > 2$ it is true by induction.

The algorithm that we have discussed requires "linear time" in the following sense: We assume that each domain D_v is specified by a linked list of values $a_1 < \cdots < a_t$. So are the sets A_{vw} and B_{vw} that occur in every implicational relation between variables v and w. If that relation is complete, and if $A_{vw} = D_v$ and $B_{vw} = D_w$, it need not appear in the input. If that relation is a correspondence, it should be specified by listing $f_{vw}(a)$ for each $a \in D_v$ and $f^{-}_{vw}(b)$ for each $b \in D_w$. If that relation is a two-fan, the values of a_{vw} and b_{vw} should be specified. The total number of elementary steps needed by the algorithm is then linear in the length of the input specification, namely $O\bigl(\sum_v(1+|D_v|)+\sum_{vw}(1+|A_{vw}|+|B_{vw}|)\bigr)$, because (i) the domain reduction takes linear time using exercise 371; (ii) the construction of G and $\widehat{G}$ takes linear time using Tarjan's algorithm. ▮

Theorem I tells us how to find *one* solution quickly when $\mathcal{P}$ is satisfiable. We shall discuss later the task of finding *all* solutions.

Max-closed constraints. Another family of manageable constraints takes advantage of the fact that domains are often *ordered* in a meaningful way. These relations, called "max-closed," arise frequently in practice, and they are easy to recognize because they're characterized by a simple property: "*If $(x_1,\ldots,x_t)$ and $(x'_1,\ldots,x'_t)$ both satisfy the relation, so does $\bigl(\max(x_1,x'_1),\ldots,\max(x_t,x'_t)\bigr)$.*"

For example, the relation '$x < y$' is obviously max-closed. A CSP in which all variables have domain $\{0,1,\ldots,d-1\}$ and all relations are '$<$' is equivalent to asking whether a given digraph is acyclic and has no oriented paths of length d.

Here are some less obvious examples of max-closed relations on nonnegative variables:

$$x \text{ is prime};$$
$$3x+5y+8z \geq 13w+21;$$
$$xyz^2+3w \geq t^3-7;$$
$$w \geq 2 \text{ or } x \geq 3 \text{ or } y \geq 4 \text{ or } z \leq 5.$$

sink
linked list
Tarjan
max-closed constraints–
digraph
acyclic
oriented paths

In general, if f and g are arbitrary monotone functions, which increase or stay the same when any of their arguments is increased, the inequality

$$f(x_1,\ldots,x_t) \;\geq\; g(x) \tag{141}$$

is a max-closed relation on $t+1$ variables. For if, say, $x \leq x'$, we have

$$g\bigl(\max(x,x')\bigr) = g(x') \leq f(x'_1,\ldots,x'_t) \leq f\bigl(\max(x_1,x'_1),\ldots,\max(x_t,x'_t)\bigr).$$

Exercise 374 proves that, similarly, every relation of the form

$$f(x_1,\ldots,x_t) \geq \alpha \quad \text{or} \quad g(x) \leq \beta, \tag{142}$$

where α and β are constants, is max-closed when f and g are monotone.

To find a solution to a CSP whose constraints are all max-closed, or to prove that no solution exists, we can begin by using a watered-down version of domain consistency (Algorithm D). Instead of using the 'revise' subroutine (90) in step D6, we use a faster-but-weaker routine:

$$\text{mc\_revise}(c,v) = \begin{cases} \text{Set } a \text{ to the largest element of } D_v. \\ \text{If } c \text{ contains no tuple having } v=a\text{, and having} \\ \quad \text{all other variables in their current domains,} \\ \text{set } D_v \leftarrow D_v \setminus a, \\ \quad \text{and repeat this process if } D_v \neq \emptyset. \end{cases} \tag{143}$$

The net effect is to filter all domains so that the largest element $v_{\max}$ remaining for every variable v has support in every constraint that involves v. (Unless, of course, the problem is unsatisfiable.) And now — ta da — we simply set $v \leftarrow v_{\max}$ for all v. This solves the problem(!). (See exercise 376.)

The running time of this backtrack-free process is clearly polynomial in the size of the input: It is dominated by the time needed to test the condition 'If c contains ...' in (143), which is at most the number of tuples in c. Variable v is placed into Q at most $|D_v|$ times; and every time it leaves Q, we call the subroutine 'mc_revise' at most mK times, where m is the number of constraints and K is the maximum number of variables in the scope of any constraint.

Of course the actual running time will usually be much smaller than this worst-case estimate. And much faster algorithms are possible when the constraints never get more complicated than so-called *basic constraints* of the form '$ax \leq by + c$'. Even more efficiency beckons when every constraint is simply '$x_j \geq x_i + t_{ij}$', as in job scheduling problems or applications to PERT charts.

The family of max-closed constraints can be generalized from total ordering '$\leq$' to *partial* ordering '$\preceq$', whenever we can interpret the domains in such a way that any two elements have a *least upper bound*, namely an element $x \vee y$ such that $x \preceq z$ and $y \preceq z$ implies $x \vee y \preceq z$. This concept is called a "semilattice" (see exercise 377).

For example, if the domain is $\{0,1,2,3,4\}$, we might have $0 \preceq 1 \preceq 2 \preceq 4$ and $0 \preceq 3 \preceq 4$, but no relation between 1 and 3 or between 2 and 3. In that case, $x \vee y = \max(x,y)$ except that $1 \vee 3 = 2 \vee 3 = 4$.

monotone functions
domain consistency
revise
filter all domains
backtrack-free process
basic constraints
PERT: Project Evaluation and Review Technique
partial ordering
least upper bound
semilattice

A CSP whose relations are closed under a semilattice ordering can be solved in polynomial time by using the ordinary Algorithm D with *full* domain consistency, but not with (143), because the concept of "largest element" no longer makes sense. Exercise 378 has the details.

CRC (connected row convex) constraints–
row convex
column convex
transpose
path consistent
domain consistent
polynomial time
complete relation
partial assignment

CRC constraints. A third kind of constraints that never "blow up," timewise, has suitably connected patterns of 1s when they're represented as matrices. A matrix of 0s and 1s is called *row convex* if no row contains a 0 between two 1s. Similarly, it's called *column convex* if no column has a 0 between two 1s. The transpose of a row convex matrix is column convex, and vice versa. (Coordinatewise convexity is rather different from the usual geometric notion of convexity: $(1,2)$ is between $(1,1)$ and $(1,4)$, but $(2,2)$ is *not* between $(1,1)$ and $(4,4)$!)

Row convexity helps us if our CSP is *path consistent*, which is a stronger condition than being domain consistent. Consider three variables $\{u,v,w\}$ that are constrained by binary relations R_{uv}, R_{uw}, and R_{vw}. If $ab \in R_{uv}$, meaning that $u=a$ is consistent with $v=b$, the local assignment $(u,v) \leftarrow (a,b)$ can be part of a global solution only if there's a value c such that $ac \in R_{uw}$ and $bc \in R_{vw}$; otherwise that assignment would wipe out all possibilities for w. Therefore we can remove the pair ab from R_{uv} whenever no such c exists. Removing that pair may lead to the removal of other pairs, and so on.

A procedure analogous to Algorithm D can be used to make the constraints of any given CSP path consistent, when all the constraints are binary. This process runs in polynomial time. Yet it can be very slow, because it removes pairs from relations, while Algorithm D merely removes elements from domains (and leaves the relations intact). Indeed, path consistency requires us to consider the $\binom{n}{2}$ relations between *all* pairs of variables u and v. Even when u and v aren't directly constrained, in the problem as given, every pair $u \in D_u$ and $v \in D_v$ that belongs to the complete relation between their domains needs to have a supporting value c in every other pair of relations R_{uw} and R_{vw}.

If, however, we do happen to have path consistent relations, and if those relations are row convex, then we're home free: The following lemma tells us that we can cruise to a global solution without backtracking.

Lemma R. *If a CSP has n variables $v_1, \ldots, v_n$, with domains of nonzero sizes $d_1, \ldots, d_n$, together with $\binom{n}{2}$ path consistent relations R_{ij} for $1 \le i < j \le n$ defined by $d_i \times d_j$ matrices that have no all-0 rows and are row convex, then it has a global solution $a_1 \ldots a_n$ that can be found in $O(n^2)$ steps.*

Proof. In fact, any locally consistent partial assignment $a_1 \ldots a_k$ that satisfies $a_i a_j \in R_{ij}$ for $1 \le i < j \le k < n$ can be extended to a locally consistent partial assignment $a_1 \ldots a_k a_{k+1}$ as follows: We suppose for convenience that the domain of v_i is $\{0,1,\ldots,d_i-1\}$. If $k=0$, set $a_1 \leftarrow 0$. Otherwise, row convexity tells us that there are indices l_i and r_i, with $0 \le l_i \le r_i < d_{k+1}$ for $1 \le i \le k$, such that $a_i a_{k+1} \in R_{i(k+1)}$ if and only if $l_i \le a_{k+1} \le r_i$. Furthermore, whenever $1 \le i < j \le k$, path consistency tells us that there's an element c such that $l_i \le c \le r_i$ and $l_j \le c \le r_j$. Hence $l_i \le r_j$.

Set $a_{k+1} \leftarrow \max\{l_1,\ldots,l_k\}$. This works because it's $\le \min\{r_1,\ldots,r_k\}$. ▮

On a normal day, we can't realistically expect to be given a CSP that satisfies all the conditions of Lemma R! But CRC constraints come to the rescue. A binary constraint is *connected row convex*, also called CRC, when its matrix M has the following properties after its all-0 rows and all-0 columns have been removed:

i) Every row is convex (that is, of the form $0^a 1^{b+1} 0^c$, for some $a, b, c \geq 0$).
ii) Every column is convex (that is, of the form $0^a 1^{b+1} 0^c$, for some $a, b, c \geq 0$).
iii) The 1s are kingwise connected. (If $M_{ab} = M_{a'b'} = 1$, there's a sequence $ab = a_1 b_1, \ldots, a_t b_t = a'b'$ such that $M_{a_1 b_1} = \cdots = M_{a_t b_t} = 1$, and both $\{a_i, a_{i+1}\}$ and $\{b_i, b_{i+1}\}$ are equal or adjacent, for $1 \leq i < t$.)

For example, two of the three implicational matrices in Fig. 119 are CRC; but the middle one fails to be kingwise connected.

Exercise 381 gives a less intuitive but mathematically cleaner characterization of conditions (i), (ii), and (iii): *A binary constraint is CRC if and only if it is closed under the median operation* $\langle xyz \rangle$. In other words, if (a_1, b_1), (a_2, b_2), and (a_3, b_3) all satisfy the constraint, so does $(\langle a_1 a_2 a_3 \rangle, \langle b_1 b_2 b_3 \rangle)$.

Theorem R. *If a satisfiable CSP has only CRC constraints, the matrices obtained after making it path consistent will satisfy the hypothesis of Lemma R.*

Proof. See exercise 382. ∎

Consider, for example, the CSP on the balanced ternary domain $\{-, 0, +\}$ that has four variables $\{u, v, u', v'\}$ and four constraints, namely $u < 0$ or $v < 0$; $u' < 0$ or $v' < 0$; $u' = -u$; $v' = -v$. Each of those constraints is implicational and also CRC. Indeed, one matrix is $\substack{111\\100\\100}$ and the other is $\substack{001\\010\\100}$. Domain consistency is initially present, but path consistency is not; for example, we can't have the pair $0-$ in $R_{uv'}$ because $(u, v') = (0, -)$ wipes out the choices for v. The path consistent matrices turn out to be $\substack{101\\000\\100}$ for uv and $u'v'$; $\substack{001\\000\\100}$ for uu' and vv'; also $\substack{100\\000\\001}$ for uv' and vu'. Lemma R applies after we remove 0 from all domains, and remove the middle row and the middle column from all matrices.

In summary, we've now seen three infinite families of relations — implicational, max-closed, and CRC — whose CSPs can be solved in polynomial time using relatively straightforward algorithms. They're galaxies of efficiency in a universe full of NP-complete problems. Implicational constraints are the easiest of these (linear time); max-closed constraints are a bit more challenging (via the weak domain consistency checking of (143)); and CRC constraints have still further complexity (path consistency checks), yet no exponential explosion.

How often do problems in those families arise in practice? That's hard to say, but we can get some idea by counting how many binary relations actually meet the restrictions when the domains have a moderate size. For example, there are $2^{100} \approx 10^{30}$ binary relations on a 10-element domain. The formulas derived in exercises 385, 386, and 387 tell us that exactly $P^{+}_{10,10} = 261{,}820{,}760$ of them are implicational; $Q^{+}_{10,10} = 8{,}787{,}513{,}806{,}478{,}134$ are max-closed; and $R^{+}_{10,10} = 2{,}018{,}581{,}537{,}596$ are CRC. Such statistics are good news for the fans of max-closed relations.

connected row convex
median operation $\langle xyz \rangle$
implicational
Domain consistency
path consistency
polynomial time

Polymorphisms. Two of the tractable families of CSPs that we've just studied are "closed" under — that is, preserved by — certain functions that operate on the domains, namely the *max* function and the *median* function. In general, functions that leave a relation invariant turn out to be quite important, and they are called *polymorphisms.**

Suppose $R(x_1,\ldots,x_m)$ is an m-ary relation on a domain D, namely a set of m-tuples $a_1\ldots a_m$ where each a_j is in D. And let $f(x_1,\ldots,x_k)$ be a k-ary operation on D, namely a function that maps k elements of D into D. We say that f is a polymorphism of R when the following basic condition holds:

$$a_{11}\ldots a_{1m}\in R,\ a_{21}\ldots a_{2m}\in R,\ \ldots,\ \text{and}\ \ a_{k1}\ldots a_{km}\in R\quad\text{implies}\qquad\qquad\qquad f(a_{11},\ldots,a_{k1})f(a_{12},\ldots,a_{k2})\ldots f(a_{1m},\ldots,a_{km})\ \in\ R. \qquad (144)$$

(The m-tuples $a_{i1}\ldots a_{im}$ need not be distinct.) "If the rows of a $k\times m$ matrix satisfy relation R, so do the f's of the columns."

Furthermore we say that f is a polymorphism of a set Γ of relations if f is a polymorphism of every relation in Γ. And f is a polymorphism of a CSP if f is a polymorphism of every relation that appears as a constraint of that CSP.

The *unary* polymorphisms ($k=1$) are Γ's *endomorphisms* (see exercise 442).

Exercise 391 shows that a binary relation is implicational if and only if it is closed under a certain ternary polymorphism $\Delta(x,y,z)$. And in exercise 392, relations that correspond to simultaneous linear equations mod d are shown to be closed under a certain ternary operation $\mu_d(x,y,z)$. Thus all four of the tractable CSP families that we've discussed so far turn out to depend on the existence of suitable polymorphisms! So it's abundantly clear that polymorphisms give us a key to understanding why CSPs can sometimes be solved efficiently.

Consider, for example, the Boolean domain $D=\{0,1\}$, and the binary OR relation $R=\{01,10,11\}$. What ternary operations $f(x,y,z)$ are polymorphisms of R? This is an instance of (144) with $m=2$ and $k=3$: f must satisfy

$$\text{if}\ \ a_1\vee b_1\ \ \text{and}\ \ a_2\vee b_2\ \ \text{and}\ \ a_3\vee b_3,\ \ \text{then}\ \ f(a_1,a_2,a_3)\vee f(b_1,b_2,b_3). \qquad (145)$$

For brevity let's write '$[a_1a_2a_3]$' instead of '$f(a_1,a_2,a_3)$'. It's clear that $[111]=1$, because we can set $a_j=b_j=1$ in (145). Furthermore we must have

$$([001]\vee[110])\ \wedge\ ([010]\vee[101])\ \wedge\ ([011]\vee[100])\ \wedge\qquad ([011]\vee[101])\ \wedge\ ([011]\vee[110])\ \wedge\ ([101]\vee[110]). \qquad (146)$$

And it's not hard to see that $[000]$ can be either 0 or 1, independently, because we know that $[111]=1$.

* What's in a name? This word 'polymorphism' is used also in the study of programming languages, but with a completely different meaning. A function symbol is called "polymorphic" if it can represent quantities of different types. That symbol might, for instance, behave differently with integer arguments than with floating point arguments or with array arguments. The '+' operation is often polymorphic; yet '+' isn't a polymorphism, it's an instance of the *concept* of polymorphism. Christopher Strachey introduced this aspect of language design while developing the language ML in the 1960s. Independently, at about the same time, V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov began to use the name 'polymorphism' for the mathematical notion of a function that preserves relations.

polymorphisms–
Strachey
Bodnarchuk
Kaluzhnin
Kotov
Romov
programming languages
ML language
types
historical notes
name
relation
operation
endomorphisms
implicational
linear equations mod d
OR
notation $[a_1\ldots a_k]$

Hence the number of polymorphisms in this case is twice the number of solutions to (146), which is a 2SAT problem in the six variables [001], [010], ..., [110]. Every clause in that formula is necessary. For if, say, ([011] ∨ [110]) were absent, we could set [011] = [110] = 0 and all other variables to 1, violating (145). Exercise 393 shows that, for $k = (1, 2, 3, 4, 5, \ldots)$, the OR relation has exactly $(2, 6, 40, 1376, 1314816, \ldots)$ k-ary polymorphisms — an interesting sequence!

Instead of OR, let's consider the relation '≤', namely $R = \{00, 01, 11\}$. Its k-ary polymorphisms are, by definition, characterized by the condition

$$a_1 \le b_1 \text{ and } \cdots \text{ and } a_k \le b_k \text{ implies } f(a_1, \ldots, a_k) \le f(b_1, \ldots, b_k); \qquad (147)$$

so they are precisely the *monotone Boolean functions*. We've already investigated the number of such functions ("Dedekind's problem," $\delta_n + 2$) — *another* interesting sequence! — in exercise 5.3.4–31 and in 7.1.4–(49).

There are 16 subsets of $\{00, 01, 10, 11\}$, hence 16 Boolean binary relations; and we've seen that two of them have interesting polymorphisms. The other 14 are instructive too. Therefore exercise 394, which studies them all, is highly recommended.

An energetic reader might, in fact, wish to look next at the 256 Boolean *ternary* relations, or at least at some of them. For example, it turns out that ternary OR ($\{001, 010, 011, 100, 101, 110, 111\}$), for which (145) becomes

$$a_1 \vee b_1 \vee c_1 \text{ and } a_2 \vee b_2 \vee c_2 \text{ and } a_3 \vee b_3 \vee c_3 \text{ implies}$$
$$f(a_1, a_2, a_3) \vee f(b_1, b_2, b_3) \vee f(c_1, c_2, c_3), \qquad (148)$$

has only 38 polymorphisms $f(x, y, z)$, not 40. Every solution to (148) must satisfy (145), because we can let $b_j = c_j$. But two of the solutions to (145) don't work for (148). Can you spot them? (See exercise 395.)

Of course we can also consider the $2^9 = 512$ binary relations on the ternary domain $D = \{0, 1, 2\}$, for which a polymorphism $f(x, y, z)$ is specified by $3^3 = 27$ ternary values $[xyz]$; now there are $3^{27} \approx 7.6 \times 10^{12}$ possibilities. We can also consider operations that are polymorphisms for more than one relation at a time; and so on. The number of potentially interesting questions about polymorphisms greatly exceeds the total number of computer scientists and mathematicians; each of us has a cornucopia of new toys to play with.

The indicator problem. A key point in this regard is the fact that the problem of finding all polymorphisms for a given CSP is itself a CSP, called the *indicator problem*. Given a set Γ of relations over the domain $D = \{0, 1, \ldots, d-1\}$, the indicator problem $\mathcal{I}_k(\Gamma)$ has d^k variables $[x_1 \ldots x_k]$, one for each k-tuple with $0 \le x_1, \ldots, x_k < d$, representing the values $f(x_1, \ldots, x_k)$ of a k-ary polymorphism f. Each of those variables has domain D. And, for each relation $R \in \Gamma$, where R is m-ary and contains t tuples, $\mathcal{I}_k(\Gamma)$ has t^k m-ary constraints based on (144), the definition of polymorphism: Every $k \times m$ matrix A whose rows $a_{11} \ldots a_{1m}, \ldots, a_{k1} \ldots a_{km}$ all belong to R generates a constraint

$$[a_{11} \ldots a_{k1}][a_{12} \ldots a_{k2}] \ldots [a_{1m} \ldots a_{km}] \in R \qquad (149)$$

between the m variables of $\mathcal{I}_k(\Gamma)$ that are named by its columns.

2SAT
monotone Boolean functions
Dedekind's problem
Boolean binary relations
Boolean *ternary* relations
indicator problem–
notation $[x_1 \ldots x_k]$: the value of $f(x_1, \ldots, x_k)$ in a polymorphism

For example, suppose $\Gamma = \{R_1, R_2\}$ contains the two relations $R_1 = \{01, 10, 11\}$ (OR) and $R_2 = \{00, 01, 11\}$ ('$\le$') that we just looked at. Then $\mathcal{I}_3(\Gamma)$ involves the eight variables [000], [001], ..., [111]. We get $3^3 = 27$ matrices A for R_1,

$$\begin{pmatrix}0&1\\0&1\\0&1\end{pmatrix},\ \begin{pmatrix}0&1\\0&1\\1&0\end{pmatrix},\ \begin{pmatrix}0&1\\0&1\\1&1\end{pmatrix},\ \begin{pmatrix}0&1\\1&0\\0&1\end{pmatrix},\ \ldots,\ \begin{pmatrix}1&1\\1&1\\1&0\end{pmatrix},\ \begin{pmatrix}1&1\\1&1\\1&1\end{pmatrix};$$

they generate the 27 constraints $[000][111] \in R_1$, $[001][110] \in R_1$, $[001][111] \in R_1$, ..., $[111][111] \in R_1$. In a similar way we get 27 matrices A for R_2, leading to the 27 further constraints $[000][000] \in R_2$, $[000][001] \in R_2$, ..., $[111][111] \in R_2$.

There's one slightly subtle point: Most of the constraints (149) can be satisfied in t different ways, because R contains t different tuples. For example, '$[000][111] \in \{01, 10, 11\}$' means that we must have either ($[000] = 0$ and $[111] = 1$) or ($[000] = 1$ and $[111] = 0$) or ($[000] = 1$ and $[111] = 1$). But '$[111][111] \in \{01, 10, 11\}$' is different, because we can't have both $[111] = 0$ and $[111] = 1$; there's only one possibility, not three. In general, there are at most t ways to satisfy the generic constraint (149), namely

$$[a_{11} \ldots a_{k1}] = b_1 \text{ and } [a_{12} \ldots a_{k2}] = b_2 \text{ and } \cdots \text{ and } [a_{1m} \ldots a_{km}] = b_m, \quad \text{for each } b_1 b_2 \ldots b_m \in R; \qquad (150)$$

we omit all cases where $a_{1i} \ldots a_{ki} = a_{1j} \ldots a_{kj}$ and $b_i \ne b_j$.

The indicator problem $\mathcal{I}_k(\Gamma)$ always has at least k solutions, regardless of Γ, because we can easily verify that the jth *projection function*

$$\pi_j(x_1, \ldots, x_k) \;=\; x_j \qquad (151)$$

is always a polymorphism, for $1 \le j \le k$. Consequently (150) always has at least one solution, when $t > 0$. (When $t = 0$ there are no matrices A, hence no (150).)

The indicator problem is important not only because it computes the polymorphisms of Γ, but also because it tells us how to make "gadgets" from Γ. A *gadget* is a combination of constraints that builds a new constraint, often using auxiliary variables as we did in (75). If Γ is a set of relations with relatively few polymorphisms, we can construct lots of useful gadgets from those relations.

For example, consider the symmetric Boolean ternary relation $R = S_{0,2,3}$ that's studied in exercise 399. This relation, '$x + y + z \ne 1$', has five tuples $\{000, 011, 101, 110, 111\}$ and exactly nine ternary polymorphisms $f(x, y, z)$:

[000]	[001]	[010]	[011]	[100]	[101]	[110]	[111]	
0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	x
0	0	1	1	0	0	1	1	y
0	0	1	1	1	1	1	1	$x \vee y$
0	1	0	1	0	1	0	1	z
0	1	0	1	1	1	1	1	$x \vee z$
0	1	1	1	0	1	1	1	$y \vee z$
0	1	1	1	1	1	1	1	$x \vee y \vee z$
1	1	1	1	1	1	1	1	1

(152)

projection function
notation π_j (projection to coordinate j)
gadgets
auxiliary variables
symmetric Boolean ternary relation
ternary relation

(Notice that (152) expresses each f by exhibiting its truth table. The next-to-last row confirms that fact that R is max-closed, namely that $\max\{x,y,z\} = x \vee y \vee z$ is a polymorphism.)

truth table
max-closed
auxiliary variables
Goldberg
3SAT

Columns [001], [010], and [011] of (152) are somewhat remarkable because they jointly contain only the four tuples $\widehat{R} = \{000, 011, 101, 111\}$, while R has five of the possible eight. The rows of (152) are the solutions to $\mathcal{I}_3(\{R\})$, which is a CSP in which every constraint is the relation R. Thus *we can regard* $\mathcal{I}_3(\{R\})$ *as a gadget for constructing the relation* $\widehat{R}$*, on variables* $\{[001], [010], [011]\}$*, from the relation* R*, using five auxiliary variables* $\{[000], [100], [101], [110], [111]\}$.

Realistically, of course, $\mathcal{I}_3(\{R\})$ is the kind of gadget that Rube Goldberg would have loved, because it uses relation R $5^3 = 125$ times! It does, however, show that the $\widehat{R}$ constraint can be enforced by enforcing only the R constraint. Furthermore, a close look at $\mathcal{I}_3(\{R\})$ shows that its mechanisms can be greatly simplified by removing redundancies. For example, the first constraint '$[000][000][000] \in R$' boils down to '$[000] \in \{0,1\}$', which is always true anyway. The next three constraints all boil down to '$[000][001] \in \{00, 01, 11\}$', meaning that $[000] \le [001]$; in essence, $R(x,y,y)$ is a gadget for '$x \le y$'. (See exercise 402.) When all the detritus is cleared away, a very simple gadget emerges:

$$\widehat{R}(x,y,z) \;=\; R(x,y,z) \;\wedge\; R(x,z,z). \tag{153}$$

Conversely, $\mathcal{I}_3(\{\widehat{R}\})$ gives us a gadget for R in terms of $\widehat{R}$ (exercise 404).

Another four-element relation, $M = \{000, 001, 011, 111\}$, can be seen in columns [000][001][011] of (152). So M, like $\widehat{R}$, can be expressed in terms of R. But that fact is no surprise, because M is just the relation '$x \le y \le z$'; we clearly have $M(x,y,z) = R(x,y,y) \wedge R(y,z,z)$. In this case there's no gadget for R in terms of M. One reason is that M has 20 ternary polymorphisms, but R has only 9; nothing constructed from M alone will lose any of M's polymorphisms.

The indicator problem $\mathcal{I}_4(\{R\})$ for *quaternary* polymorphisms provides us with another interesting example. In this case there are 16 variables, $\{[0000], [0001], \ldots, [1111]\}$, and $5^4 = 625$ constraints. It has 17 solutions, which are analogous to (152) but they include also the Boolean functions w, $w \vee x$, $w \vee y$, $w \vee x \vee y$, …, $w \vee x \vee y \vee z$. Columns [0101][1010][1100] of those solutions turn out to have only seven of the eight binary possibilities, namely

$$T \;=\; \{000, 010, 011, 100, 101, 110, 111\}. \tag{154}$$

In other words, T is the Boolean relation '$xyz \ne 001$', which also can be formulated nicely as a 3SAT clause: '$(x \vee y \vee \bar{z})$'.

This relation T is substantially weaker than R; in fact, we easily see that

$$R(x,y,z) \;=\; T(x,y,z) \;\wedge\; T(y,z,x) \;\wedge\; T(z,x,y). \tag{155}$$

Yet $\mathcal{I}_4(\{R\})$ is a gadget that expresses T in terms of R. As before, this gadget appears to be hopelessly complicated; indeed, there now are thirteen auxiliary variables, not five. But exercise 405 shows that massive simplifications are possible, and they lead to a simple formula with only two auxiliaries:

$$T(x,y,z) \;=\; R(v,w,z) \wedge R(v,x,x) \wedge R(w,y,y), \text{ for some } v \text{ and } w. \tag{156}$$

NP-complete–
potent
pi as source
magic masks
3SAT
one-in-three satisfiability
shortcut

NP-complete families of CSPs. When a set Γ of relations on a domain D has only the projection functions π_j of (151) as polymorphisms, we shall call it *potent* for D, because its indicator problems give us a "universal gadget": If R is any relation whatsoever, on D, we can use the construction above to express R in terms of Γ. For if R is an m-ary relation with k tuples, we can find m columns of $\mathcal{I}_k(\Gamma)$ where the tuple τ appears if and only if $\tau \in R$. Indeed, we get such columns from any $k \times m$ matrix that exhibits the tuples of R.

For example, one of the simplest relations of all is $S = \{001, 010, 100\}$, otherwise known as the Boolean symmetric function $S_1(x, y, z)$, '$x + y + z = 1$'. It's not difficult to show (exercise 400) that S is potent for $\{0,1\}$. So we can encode any given Boolean relation R in terms of S by making a gadget for R from an indicator problem for S. If R has, say, four tuples, $\{11001, 00100, 00111, 11101\}$, we can fabricate it from $\mathcal{I}_4(\{S\})$, because only four solutions satisfy the constraints of that problem:

$$\begin{matrix} 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\ 0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1\\ 0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1\\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \end{matrix}\;. \qquad (157)$$

In this list, the values of variable $[a_1a_2a_3a_4]$ appear in the column whose entries are a_1, a_2, a_3, a_4, from top to bottom. (Notice the similarity to the "magic masks" of 7.1.3–(47).) Thus the five columns [1001][1001][0111][0010][1011] will be a gadget for R in terms of S, using $16 - 4 = 12$ auxiliary variables. (This particular "random" relation R happens to use column [1001] twice.)

Consequently *any* CSP on $\{0,1\}$ can be rewritten so that each of its constraints is simply the relation S applied to three of the variables. We just replace all of the original constraints by appropriate gadgets. If we have a 3SAT problem, for instance, it's a CSP that uses at most four basic relations,

$$R_0(x,y,z) = (x \vee y \vee z), \quad R_1(x,y,z) = (\bar{x} \vee y \vee z),$$
$$R_2(x,y,z) = (\bar{x} \vee \bar{y} \vee z), \quad R_3(x,y,z) = (\bar{x} \vee \bar{y} \vee \bar{z}), \qquad (158)$$

depending on the number of negated literals. So we need only four gadgets to turn it into a problem that's based on S alone. And the size of that problem is at most a constant multiple of the original problem size. Since 3SAT is NP-complete, the family of all CSPs based on S is NP-complete.

Of course we've already discussed Boolean constraints quite thoroughly in Section 7.2.2.2, and the potency of S isn't really a surprise. Exercise 7.2.2.2–517 noted that "one-in-three satisfiability" is NP-complete. Let's turn now to relations that are potent in larger domains.

The simplest potent relation for $\{0,1,2\}$ is a binary relation Θ called a ternary *shortcut* (see exercise 412). It has only four tuples:

$$\Theta \;=\; \{01, 12, 20, 02\}. \qquad (159)$$

(A three-element domain is often represented more elegantly when the elements are given the "balanced" labels $\{-, 0, +\}$; then Θ becomes $\{-0, 0+, +-, -+\}$, and its structure becomes more clear. But we shall stick to $\{0,1,2\}$ for now.)

Since Θ is potent, we can encode a "random" relation R on $\{0,1,2\}$ by making a gadget from $\mathcal{I}_k(\{\Theta\})$, where k is the number of tuples in R. For example, if $k=3$ and $R=\{10010, 21101, 22220\}$, the solutions of $\mathcal{I}_3(\{\Theta\})$ are

$$\begin{matrix} 0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&2&2&2&2&2&2&2&2&2\\ 0&0&0&1&1&1&2&2&2&0&0&0&1&1&1&2&2&2&0&0&0&1&1&1&2&2&2\\ 0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2&0&1&2 \end{matrix}\,; \qquad (160)$$

and we obtain R from variables/columns [122][012][012][102][010], using $27-4=23$ auxiliary variables. (Compare with (157).)

As before, potency implies that the family of all CSPs on Θ is NP-complete. We can, for example, express the NP-complete problem of 3-coloring a graph by using $\mathcal{I}_6(\{\Theta\})$ to make a gadget for the relation '$x \neq y$' on $\{0,1,2\}$.

Notice that the relation '$x \neq y$' is *not* potent, by our definition, although it can yield NP-complete problems. Indeed, exercise 401 determines its k-ary polymorphisms, of which there are $k \cdot d!$. But we'll see later that when we beef it up to the set

$$\Gamma \;=\; \bigl\{\neq, \{0\}, \{1\}, \ldots, \{d{-}1\}\bigr\}, \qquad (161)$$

where $\{a\}$ is the unary relation that's satisfied only by the constant a, we *do* get a potent set of relations for $\{0,1,\ldots,d{-}1\}$, if $d>2$.

It's easy to verify by computer that $\mathcal{I}_3(\{\Theta\})$ has only the three solutions in (160). But $\mathcal{I}_6(\{\Theta\})$ has $3^6=729$ variables; the task of verifying that $\mathcal{I}_6(\{\Theta\})$ has only six solutions is more formidable. In general, a set Γ of relations is potent for its domain only if it passes an infinite number of tests: Its indicator problem $\mathcal{I}_k(\Gamma)$ must have exactly k solutions, for all $k \geq 1$. We can, however, prove that a single test will actually suffice:

Theorem F. *Let Γ be a set of relations on a domain D of size $d \geq 3$. Then Γ is potent for D if and only if the indicator problem $\mathcal{I}_d(\Gamma)$ has exactly d solutions.*

Before we prove this theorem, we need to learn about the "essential arity" of a function. When $f(x_1,\ldots,x_k)$ is a k-ary function we say that f *depends on* x_j if there are $k+1$ elements $a_1, \ldots, a_k, a'_j$ such that the substitution of a'_j for a_j gives $f(a_1,\ldots,a_j,\ldots,a_k) \neq f(a_1,\ldots,a'_j,\ldots,a_k)$. The "essential arity" of f, denoted by $\mathrm{ea}(f)$, is the number of variables x_j on which f depends. In particular, f is constant if and only if $\mathrm{ea}(f)=0$. We say that f is *full* when its essential arity is k, that is, when f depends on all of its arguments.

Consequently a function has essential arity r if and only if it can be written

$$f(x_1,\ldots,x_k) \;=\; h(x_{j_1},\ldots,x_{j_r}), \quad \text{where } j_1<\cdots<j_r \text{ and } h \text{ is full.} \qquad (162)$$

For example, consider the function

$$q(x_1,\ldots,x_k) = (x_1+2x_2+\cdots+kx_k) \bmod d \qquad (163)$$

on the domain $\{0,1,\ldots,d-1\}$. It clearly depends on variable x_j if and only if j is not a multiple of d; so its essential arity is $k-\lfloor k/d\rfloor = \bigl\lceil \frac{d-1}{d}k \bigr\rceil$. If, say, $k=7$ and $d=3$, we have $q(x_1,\ldots,x_7)=h(x_1,x_2,x_4,x_5,x_7)$, where the function $h(x_1,\ldots,x_5)=(x_1-x_2+x_3-x_4+x_5) \bmod 3$ is full.

pi as source
3-coloring a graph
nonequality
unary relation
constant
depends on
essential arity
notation $\mathrm{ea}(f)$
full

essential arity
polarities–
polarities
Birkhoff

Returning to Theorem F, one direction of the proof is obvious: If $\mathcal{I}_d(\Gamma)$ has more than d solutions, Γ certainly isn't potent. Suppose, therefore, that every d-ary polymorphism of Γ is a projection, although Γ does have a k-ary polymorphism $f(x_1,\ldots,x_k)$ that *isn't* a projection, where k is as small as possible.

We can't have $k<d$, because $g(x_1,\ldots,x_d)=f(x_1,\ldots,x_k)$ would be a d-ary polymorphism in that case, and g isn't a projection.

Therefore $k>d$, and the minimality of k implies that f must be full. For if f doesn't depend on x_j, and if $i\neq j$, then the function

$$g(x_1,\ldots,x_k)=f(x_1,\ldots,x_{j-1},x_i,x_{j+1},\ldots,x_k) \tag{164}$$

would be a $(k-1)$-ary polymorphism.

Exercise 420 is now the clincher. It proves that, when $k>d$, every full k-ary polymorphism f on domain D has a pair of indices $i\neq j$ such that the essential arity of the polymorphism (164) is at least $k-2$. And that polymorphism cannot be a projection, because $k-2>1$ when $d\geq 3$. ▮

***Polarities and clones.** The theory of CSP tractability makes use of *polarity*, a fundamental notion that underlies numerous parts of mathematics. In general, whenever we have a binary relation R between the elements of two sets A and B, there are "polarities" between the subsets of A and the subsets of B:

$$\text{If } X\subseteq A,\quad X^{\nearrow}=\{y\in B\mid x\,R\,y \text{ for all } x\in X\}. \tag{165}$$

$$\text{If } Y\subseteq B,\quad Y^{\swarrow}=\{x\in A\mid x\,R\,y \text{ for all } y\in Y\}. \tag{166}$$

[See G. Birkhoff, *Lattice Theory* (American Mathematical Society, 1940), §32.] For example, if $A=\{0,1,2\}$ and $B=\{0,1,2,3,4\}$, the relation R can be specified by a 3×5 matrix and the possibilities for R, X, and Y might be

$$R=\begin{pmatrix}0&1&0&1&0\\1&0&0&1&1\\1&0&1&0&0\end{pmatrix};$$

X	$X^{\nearrow}$	Y	$Y^{\swarrow}$	Y	$Y^{\swarrow}$	Y	$Y^{\swarrow}$	Y	$Y^{\swarrow}$
ϵ	01234	ϵ	012	3	01	4	1	34	1
0	13	0	12	03	1	04	1	034	1
1	034	1	0	13	0	14	ϵ	134	ϵ
01	3	01	ϵ	013	ϵ	014	ϵ	0134	ϵ
2	02	2	2	23	ϵ	24	ϵ	234	ϵ
02	ϵ	02	2	023	ϵ	024	ϵ	0234	ϵ
12	0	12	ϵ	123	ϵ	124	ϵ	1234	ϵ
012	ϵ	012	ϵ	0123	ϵ	0124	ϵ	01234	ϵ

(Here 'ϵ' stands for $\emptyset$ and '01234' stands for $\{0,1,2,3,4\}$, etc.)

One of the immediate consequences of (165) and (166) is that

$$X\ \subseteq\ X^{\nearrow\swarrow} \qquad\text{and}\qquad Y\ \subseteq\ Y^{\swarrow\nearrow}; \tag{167}$$

indeed, $X^{\nearrow\swarrow}=\{x\in X\mid x\,R\,y \text{ for all } y\in X^{\nearrow}\}$, and every element of X is clearly in that set. For instance, if $X=\{0,2\}$ in the relation R above, $X^{\nearrow\swarrow}$ is $\{0,1,2\}$.

Even more is true, because polarities reverse the subset order:

$$X_0\subseteq X_1 \quad\text{implies}\quad X_0{}^{\nearrow}\supseteq X_1{}^{\nearrow}; \tag{168}$$

$$Y_0\subseteq Y_1 \quad\text{implies}\quad Y_0{}^{\swarrow}\supseteq Y_1{}^{\swarrow}. \tag{169}$$

closures
closed subsets
order-reversing
Galois connection
Ore

Applying (168) to (167) tells us, for example, that $X^{\nearrow} \supseteq (X^{\nearrow\swarrow})^{\nearrow} = X^{\nearrow\swarrow\nearrow}$. But we also know, by setting $Y = X^{\nearrow}$ in (167), that $X^{\nearrow} \subseteq (X^{\nearrow})^{\swarrow\nearrow} = X^{\nearrow\swarrow\nearrow}$. Hence

$$X^{\nearrow} \;=\; X^{\nearrow\swarrow\nearrow} \qquad \text{and} \qquad Y^{\swarrow} \;=\; Y^{\swarrow\nearrow\swarrow}. \tag{170}$$

This rule is nice; but it's also a bit confusing, because $X^{\nearrow}$ is a subset of B while X is a subset of A. Let's step back and write

$$\overline{X} \;=\; X^{\nearrow\swarrow}, \qquad \overline{Y} \;=\; Y^{\swarrow\nearrow}, \tag{171}$$

calling those sets the "closures" of X and Y. According to (167) and (170),

$$X \subseteq \overline{X}, \quad \overline{\overline{X}} = \overline{X}, \quad Y \subseteq \overline{Y}, \quad \overline{\overline{Y}} = \overline{Y}. \tag{172}$$

A subset is called *closed* if it equals its own closure. Notice that the closed subsets of A are precisely the subsets of the form $Y^{\swarrow}$ for some $Y \subseteq B$, and the closed subsets of B are precisely the subsets of the form $X^{\nearrow}$ for some $X \subseteq A$.

For example, the closed subsets of A and B with respect to the 3×5 relation R above are

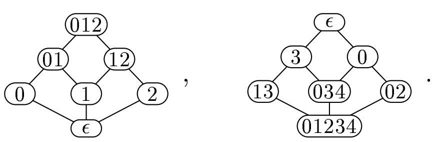
$$\epsilon, 0, 1, 2, 01, 12, 012 \;\subseteq\; A; \qquad \epsilon, 0, 3, 02, 13, 034, 01234 \;\subseteq\; B.$$

There happen to be seven of each. And that's *not* a coincidence(!):

Theorem P. *Let R be a relation between sets A and B, giving rise to polarities and closed subsets as above. There is a one-to-one correspondence between the closed subsets of A and the closed subsets of B. Moreover, if X_0 corresponds to Y_0 and X_1 corresponds to Y_1, then $X_0 \subseteq X_1$ if and only if $Y_0 \supseteq Y_1$.*

Proof. The mappings $X \mapsto X^{\nearrow}$ and $Y \mapsto Y^{\swarrow}$ take closed subsets to closed subsets and are inverses of each other. ▮

An order-reversing correspondence of this kind is called a "Galois connection" (see O. Ore, *Trans. Amer. Math. Soc.* **55** (1944), 493–513). In our 3×5 example it can be seen in the set-inclusion diagrams



(173)

That was a small, finite example. But the same reasoning works equally well when the sets A and B are infinite. It's instructive to ponder, for example, the polarities and closed subsets that arise when $A = B$ is the set of all positive integers, and R is the relation "x divides y." Or when $A = B$ is the set of all 2×2 matrices of integers, and R is the relation "$xy = yx$." Or when A is the set of all real numbers, B is the set of all polynomials with integer coefficients, and R is the relation "$y(x) = 0$." Just about *any* noteworthy relation leads to interesting questions when considered from this standpoint (see exercises 417–418).

We also will exploit another key idea, the mathematical notion of a “clone of operations.” An operation, also called an operator, is a function whose arguments and result belong to the same domain. If O is a set of operations on a domain D, the *clone* of O is the set $\mathcal{C}$ of all operations expressible as an algebraic formula that involves only variable parameters and the operators in O.

More precisely, a k-ary operation $f(x_1,\ldots,x_k)$ is in $\mathcal{C}$ if and only if either

i) $f(x_1,\ldots,x_k)=x_j$ for some j with $1\le j\le k$; or

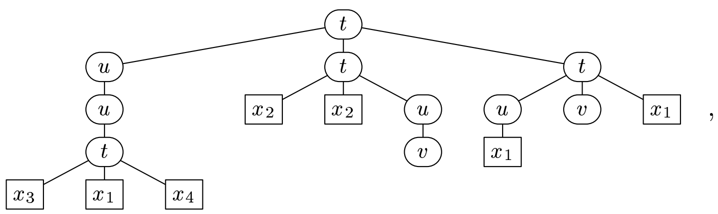
ii) $f(x_1,\ldots,x_k) = g\bigl(h_1(x_1,\ldots,x_k),\ldots,h_l(x_1,\ldots,x_k)\bigr)$, where g is an l-ary operation in O and h_1, …, h_l are k-ary operations in $\mathcal{C}$.

For example, if O contains just a ternary operation t, a unary operation u, and a nullary (constant) operation v, one of the 4-ary operations in $\mathcal{C}$ is

$$f(x_1,x_2,x_3,x_4)=t\bigl(u(u(t(x_3,x_1,x_4))),t(x_2,x_2,u(v)),t(u(x_1),v,x_1)\bigr). \qquad (174)$$

Such formulas are called *terms* in the study of universal algebra. The elements of $\mathcal{C}$ are called *derived operators* of O, or — informally — “combos” of O.

Computer scientists are familiar with terms, because it’s customary to represent algebraic formulas as trees, whose internal nodes are labeled with the names of operations and whose external nodes are labeled with the names of variables. (See Section 2.3.2.) For example, the tree that corresponds to the term (174) is



, (175)

and trees like this help give us an intuitive understanding of clones. Notice that the nodes labeled with v, a nullary operator, are considered to be internal nodes (shown as ovals, not rectangles), even though they have no children.

There are infinitely many trees, but only finitely many k-ary operations on a finite domain. Thus every operation in a clone can typically be represented by many different trees/terms, depending on the actual operations in O. We can compute all of $\mathcal{C}$’s distinct k-ary operations, at least in principle, with the algorithm of exercise 421; but in practice there might be so many of them that the computation isn’t feasible.

Sometimes we’re lucky, and we can psych out all of the achievable k-ary operations by simple reasoning, even when the domain is infinite. For example, if O is the single operation ‘+’ on the domain of real numbers, it’s easy to see that its k-ary combos are just the linear combinations $a_1x_1+\cdots+a_kx_k$, where each coefficient a_j is a nonnegative integer, and the coefficients aren’t all zero.

A more interesting case arises when the sole operation in O is the binary operation that maps x and y to $x+2y$. Again, all k-ary combos clearly have the form $a_1x_1+\cdots+a_kx_k$, with nonnegative integer coefficients. But many of those operations, such as x_1+x_2, are no longer present. It turns out that we can achieve the function $6x_1+7x_2+8x_3$, but not $7x_1+8x_2+9x_3$! What’s the secret?

operation
clone
composition of functions
superposition of functions
ternary operation
unary operation
nullary (constant) operation
terms
universal algebra
derived operators
combos
trees
internal nodes
external nodes

projection functions
median operation
monotone self-dual Boolean functions
self-dual Boolean functions
Boolean functions
Post
Pol(Γ)
Inv(O)
max-closed relations

After some experimentation, a simple pattern emerges:

$$a_1x_1 + \cdots + a_kx_k \in \mathcal{C} \iff \text{exactly one of the coefficients is odd.} \qquad (176)$$

Indeed, that condition is clearly necessary, because it is true of the projection functions 'x_j' that satisfy condition (i) in our definition of a clone, and it is preserved by condition (ii): If both $h_1(x_1,\ldots,x_k)$ and $h_2(x_1,\ldots,x_k)$ have exactly one odd coefficient, so does $h_1(x_1,\ldots,x_k)+2h_2(x_1,\ldots,x_k)$. Conversely, if $f(x_1,\ldots,x_k) = a_1x_1 + \cdots + a_kx_k$ has exactly one odd coefficient and is not a projection function, some coefficient a_j exceeds 1. Hence $g(x_1,\ldots,x_k) = f(x_1,\ldots,x_k) - 2x_j$ satisfies the condition, and we have $g \in \mathcal{C}$ by induction on $a_1 + \cdots + a_k$. It follows that $f = g + 2x_j \in \mathcal{C}$.

We've seen clones before, without calling them by name. For example, Theorem 7.1.1P says that the clone of the median operation $\langle xyz\rangle$ on the domain $\{0,1\}$ is the set of all monotone self-dual Boolean functions. Emil Post, who proved that theorem in 1920, took the ideas much further and identified *every* clone on $\{0,1\}$. For example, he showed that the Boolean functions

$$m_k(x_1,\ldots,x_k) \;=\; \bigvee_{1\le i<j\le k} (x_i \wedge x_j) \;=\; [x_1 + \cdots + x_k > 1] \qquad (177)$$

have the property that m_l is in the clone of m_k but m_k isn't in the clone of m_l, when $l > k \ge 4$. (See exercise 424.)

Clones are particularly interesting to us because we get them from polarities and polymorphisms. *If* Γ *is any set of relations on a finite domain* D*, we let* $\mathrm{Pol}(\Gamma)$ *be the set of all its polymorphisms; and if* O *is any set of operations on* D*, we let* $\mathrm{Inv}(O)$ *be the set of all relations that are invariant under every operation in* O. This is a polarity, where '$x\,R\,y$' in (165) and (166) means "relation x is preserved by operation y." Indeed, by definition, $\mathrm{Pol}(\Gamma) = \Gamma^{\nearrow}$ and $\mathrm{Inv}(O) = O^{\swarrow}$.

According to polarity theory, $\overline{\Gamma}$ is $\mathrm{Inv}(\mathrm{Pol}(\Gamma))$, the set of all relations that are preserved by all polymorphisms of Γ. And $\overline{O}$ is $\mathrm{Pol}(\mathrm{Inv}(O))$, the set of all operations that preserve every relation that's invariant under O. And there's a one-to-one correspondence between the closed sets of relations and the closed sets of operations. This correspondence reverses set inclusion, as in (173).

For example, if Γ is potent, then $\mathrm{Pol}(\Gamma)$ is the set of projection operators, and $\overline{\Gamma}$ is the set of *all* relations on D. If O is the operation '$\max(x,y)$', then $\mathrm{Inv}(O)$ is the set of max-closed relations, and $\overline{O}$ is the set of all $f(x_1,\ldots,x_k) = \max(x_{j_1},\ldots,x_{j_l})$, for some $1 \le j_1 < \cdots < j_l \le k$.

Polarity theory also tells us that the closed sets of operators are the sets that are $\mathrm{Pol}(\Gamma)$ for some Γ. In other words, O is closed if and only if it's the set of polymorphisms of some set of relations. And — here's the point — the set of polymorphisms of Γ is always a *clone*, by rules (i) and (ii) in the definition of clone: A projection function is always a polymorphism; and any composition of polymorphisms is itself a polymorphism. (If t, u, and v in (174) are polymorphisms of Γ, so is the 4-ary term that corresponds to every subtree of (175).) Conversely, if O is a clone, exercise 429 proves that every element of $\mathrm{Pol}(\mathrm{Inv}(O))$ belongs to O. We conclude that a set of operations is "closed", $\overline{O} = O$, if and only if it's a clone.

Similarly, a set of *relations* is closed if and only if it's a so-called *relational clone*. Any set of the form $\mathrm{Inv}(O)$ is a clone of relations, just as any set of the form $\mathrm{Pol}(\Gamma)$ is a clone of operations.

The closure $\overline{\Gamma}$ of a set of relations is perhaps best understood as the set of all *gadgets* that can be made from Γ. According to the formula $\overline{\Gamma} = \mathrm{Inv}(\mathrm{Pol}(\Gamma))$, we start with the polymorphisms of Γ, which are the solutions to indicator problems, and pass to the set of all relations that are preserved by all of those polymorphisms. Exercise 409 proves that those are the achievable gadgets, the derivable relations free of tuples that aren't preserved.

We've seen that the construction of gadgets — that is, the construction of relations from other relations — boils down to the solution of certain indicator problems, which are CSPs. One instructive way to view this is to define the clone $\mathcal{C}$, of a given set Γ of relations on domain D, to be the set of all relations R for which R is either

i) the empty set $\emptyset$, regarded as a unary relation (see exercise 431); or
ii) the unary relation D; or
iii) a relation of Γ; or
iv) the product $R' \otimes R''$ of relations R' and R'' in $\mathcal{C}$ (see below); or
v) the projection $\Pi_\sigma R'$ of some relation R' in $\mathcal{C}$ (see below); or
vi) the equality selection relation

$$\Sigma_{ij} R' \;=\; \{a_1 \ldots a_m \in R' \mid a_i = a_j\}, \tag{178}$$

where R' is an m-ary relation in $\mathcal{C}$ and $1 \le i < j \le m$.

If $R' \subseteq D^{m'}$ and $R'' \subseteq D^{m''}$ are m'-ary and m''-ary relations, their product

$$R' \otimes R'' \;=\; \{a_1 \ldots a_{m'} b_1 \ldots b_{m''} \mid a_1 \ldots a_{m'} \in R' \text{ and } b_1 \ldots b_{m''} \in R''\} \tag{179}$$

is the $(m'+m'')$-ary relation obtained by simply concatenating their tuples. And if $\sigma = i_1 \ldots i_s$ is any sequence of integers, $1 \le i_j \le m'$, the projection

$$\Pi_\sigma R' \;=\; \{a_{i_1} \ldots a_{i_s} \mid a_1 \ldots a_{m'} \in R'\} \tag{180}$$

is an s-ary relation obtained by selecting and possibly duplicating specified elements of the tuples. (Notice that s can be larger than m'.)

For example, if $R' = \{001, 120\}$ and $R'' = \{10, 11, 22\}$ we have

$$\begin{aligned} R''' = R' \otimes R'' \;&=\; \{00110, 00111, 00122, 12010, 12011, 12022\};\\ R^{\mathrm{iv}} = \Pi_{3141413} R''' \;&=\; \{0111110, 0121210, 1010101, 1020201\};\\ R^{\mathrm{v}} = \Sigma_{37} R^{\mathrm{iv}} \;&=\; \{1010101\}. \end{aligned}$$

This "algebra of relations" provides mechanisms that suffice to solve indicator problems, so it allows us to form all possible gadgets from a given set Γ. It is familiar in the theory of databases, where the "join" operation $R' \bowtie R''$ is a combination of the $\otimes$, Σ, and Π operators.

***The dichotomy theorem.** We've seen that some families Γ of relations lead to CSPs whose satisfiability or unsatisfiability is guaranteed to be tractable — that is, decidable by a known polynomial-time algorithm — while others can yield CSPs whose decision problem is NP-complete.

relational clone
co-clone, see relational clone
closure
gadgets
indicator problems
empty set $\emptyset$
product $R' \otimes R''$
projection $\Pi_\sigma R'$
equality selection relation
selection relation
diagonalization, see equality selection
concatenating
algebra of relations
databases
join
SPJ-algebra: select, project, join
dichotomy theorem–
decision problem
NP-complete

Richard Ladner [*JACM* **22** (1975), 155–171] showed the existence of "NP-intermediate" problems, which lie *between* those two extremes, if P $\neq$ NP. For example, the question "Does N have a prime factor less than M?" might well be an NP-intermediate problem, as far as anyone knows. Thus it's natural to wonder whether or not some relation sets Γ might be NP-intermediate.

Thomas J. Schaefer [*STOC* **10** (1978), 216–226] proved that there's a strict dichotomy between tractability and NP-completeness when the domain is $\{0,1\}$. Andrei A. Bulatov established it also for the ternary domain $\{0,1,2\}$; the tractable sets Γ in that domain are considerably more complicated, but their clones $\mathrm{Pol}(\Gamma)$ are known [*JACM* **53** (2006), 66–120].

We observed above that all four of the main tractable families can be characterized by the existence of certain ternary polymorphisms. In general, the existence of certain k-ary polymorphisms, for arbitrary k, turns out to be the key to tractability. For example, a k-ary function $f(x_1,\ldots,x_k)$ on domain D is said to be *idempotent* if we have

$$f(x,x,\ldots,x) \;=\; x \qquad \text{for all } x \in D. \tag{181}$$

And we call it a *near unanimity* operator, or NU for short, if $k > 2$ and

$$f(y,x,x,\ldots,x) = f(x,y,x,\ldots,x) = \cdots = f(x,x,\ldots,x,y) = x \tag{182}$$

for all $x,y \in D$; the presence of $k-1$ equal elements determines the outcome. (In particular, an NU operator is always idempotent, because y might equal x.) It's a *weak near unanimity* operator (WNU) if it is idempotent and satisfies

$$f(y,x,x,\ldots,x) = f(x,y,x,\ldots,x) = \cdots = f(x,x,\ldots,x,y), \tag{183}$$

which is like (182) except for the '$= x$' at the end. If $D = \{0,1,2\}$, for example, a ternary WNU operator satisfies $f(1,0,0) = f(0,1,0) = f(0,0,1)$, but all three of those function values might be either 0, 1, or 2. Finally, even less constrained is a *lax weak near unanimity* operator (LWNU), which satisfies (183) but needn't satisfy (181). Notice that WNU and LWNU make sense also when $k = 2$, because (183) is simply the commutative law in that case.

Theorem D. *Let Γ be a finite set of relations on a finite domain D. The decision problem for every CSP whose constraints all belong to Γ is solvable by a known polynomial-time algorithm if Γ has an LWNU polymorphism. Otherwise that set of decision problems is NP-complete.*

Proof. The proof of this theorem is far beyond the scope of this book; references to the literature appear in the history section below. The NP-complete half follows from advanced theories of universal algebra; the polynomial-time half requires the construction of a suitable algorithm. Two such algorithms were published in 2017: one by Andrei A. Bulatov, the other by Dmitriy N. Zhuk. ∎

Many other conditions are known to be equivalent to the existence of an LWNU polymorphism. The most general of these is probably the existence of a

Ladner
NP-intermediate
P $\neq$ NP
prime factor
Schaefer
Bulatov
idempotent
near unanimity
NU
weak near unanimity
WNU
lax weak near unanimity
LWNU
commutative law
NP-complete
Bulatov
Zhuk

Taylor term
lax Maróti pair
Maróti pair
indicator problem
Bulatov
Zhuk
domain
unary relations
idempotent

lax "Taylor term," namely a k-ary polymorphism f that satisfies k identities

$$f(u_{i1},\ldots,u_{ik}) = f(v_{i1},\ldots,v_{ik}) \text{ and } u_{ii} \neq v_{ii}, \qquad \text{for } 1 \le i \le k, \tag{184}$$

where each u_{ij} and v_{ij} is either x or y. The identities needn't be distinct. When $k = 4$, for example, $f(x_1,\ldots,x_4)$ is a lax Taylor term if it satisfies

$$f(y,x,y,x) \;=\; f(x,y,x,x) \;=\; f(y,x,x,y), \tag{185}$$

because the first equality covers $i \neq 4$ in (184) and the second covers $i \neq 3$.

Amazingly, *an LWNU polymorphism exists if and only if there are polymorphisms $p(x,y,z)$ and $q(x,y,z)$ that satisfy the three identities*

$$p(x,y,x) = q(x,y,x); \quad p(x,y,y) = q(x,x,y) = q(y,x,x); \tag{186}$$

p and q are called a *lax Maróti pair*. The tractability of Γ depends only on its 3-ary polymorphisms! (See exercises 442, 447, and the references in their answers.)

As an example of Theorem D, consider the following relation on $D = \{0,1,2,3\}$:

$$G \;=\; \{12, 21, 13, 23, 30\}. \tag{187}$$

The indicator problem $\mathcal{I}_3(\{G\})$ shows that it has 126760 ternary polymorphisms, including four that are LWNU. One of the latter is

$$f(x,y,z) \;=\; \bigl(0 \in \{x,y,z\}?\ 0\colon 3 \in \{x,y,z\}?\ 3\colon \langle xyz\rangle\bigr). \tag{188}$$

Hence we can use either Bulatov's algorithm or Zhuk's algorithm to solve any CSP whose relations all have this polymorphism, in polynomial time. Notice that (188) satisfies (182) when $(x,y) = (0,1)$, $(0,2)$, $(0,3)$, $(1,2)$, $(2,1)$, $(3,1)$, $(3,2)$, but not when $(x,y) = (1,0)$, $(2,0)$, $(3,0)$, $(1,3)$, $(2,3)$.

Another interesting relation on the same domain is

$$Z \;=\; \{12, 23, 31, 20, 30\}. \tag{189}$$

This one has a huge number of k-ary polymorphisms — in fact, $k \cdot 4^{4^{k-1}-3^{k-1}}$ — yet *none* of them are LWNU! Therefore CSPs that use Z can be NP-complete. Indeed, a gadget for the relation $\{002, 020, 200\}$ can be constructed from Z, and we know that that relation is NP-complete. (See exercises 400, 452, and 453.)

There's a sense in which our discussion of tractable CSPs doesn't jibe with the way we've defined things. According to the definition of a CSP at the beginning of this section, every variable v has its own specified domain D_v; but our discussion of polymorphisms and clones deals only with a *single*, all-encompassing domain D. The theory can fortunately be brought into alignment with practice by simply including a few unary relations into the set Γ, namely the sets $D \setminus \{a\}$, one for each element $a \in D$. With these relations a user can restrict any variable v to any desired subset of D. Some tractable Γ become NP-complete; but they aren't of practical importance. The net effect is to decrease the number of polymorphisms in some cases, but usually in a harmless way (see exercise 441).

In particular, this extension of Γ makes every polymorphism idempotent (see (181)). So a lax WNU becomes a WNU. Similarly, a lax Taylor term becomes an ordinary Taylor term; Taylor's classic concept includes (181) together with (184).

***Exploiting a polymorphism.** By definition, a polymorphism is an operation that preserves relations. But it's also more: We can use it for data compression! For example, if we know that an $m \times n$ binary matrix R defines a max-closed relation, we don't need a table of $\Omega(mn)$ entries to list its elements; it suffices to know only $m+n$ quantities:

$$\begin{aligned}\lambda(x) &= \min\{0 \le y < n \mid x\,R\,y\}, \text{ for } 0 \le x < m;\\ \rho(y) &= \min\{0 \le x < m \mid x\,R\,y\}, \text{ for } 0 \le y < n.\end{aligned} \qquad (190)$$

They suffice because $x\,R\,y$ if and only if $x \ge \rho(y)$ and $y \ge \lambda(x)$. (Here either $\lambda(x)$ or $\rho(y)$ might be ∞, if the min operation is applied to an empty set.)

In general the law (144) that defines a polymorphism tells us that if certain tuples belong to a relation R, then certain other tuples also belong to R. If we know enough "seed" tuples of R, we can learn them all by using (144) repeatedly.

We shall see that this idea leads to an instructive algorithm, called Algorithm N, that's different from all of the methods that we've seen so far. Given any NU (near unanimity) polymorphism $f(x_1,\ldots,x_k)$ on a domain D, Algorithm N solves any n-variable CSP whose constraints are all preserved by f, and it does this "online" by incorporating the constraints one by one. Furthermore, when the arity k of f is fixed, Algorithm N's runtime is polynomial, in n and the length of the constraints, assuming that the constraints are specified by lists of m-tuples.

(However, we'll soon see that Algorithm N is by no means suitable for practical use. It's a "thought experiment," intended to clarify why exponential time can be avoided, but it's not intended for actual implementation. We'll keep it simple, with no attempts at optimization.)

The main point is that the set of all *solutions* to a CSP with n variables is a *relation* R on those variables. And if f preserves every constraint of the CSP, then it also preserves R. Algorithm N is able to know everything about the entire set of solutions, because it maintains a polynomial-size subset of R from which the entire relation could in principle be recovered by repeatedly applying (144).

The given NU polymorphism, $f(x_1,\ldots,x_k)$, is k-ary, where $k \ge 3$. We don't assume that f has any special mathematical properties, except that it satisfies (182). As far as we're concerned, it's an otherwise arbitrary table of d^k values in $D=\{0,1,\ldots,d-1\}$, one for each choice of $(x_1,\ldots,x_k)$ not limited by (182).

As usual, some notation will be helpful. Let R be a relation and let f be a k-ary operation on the same domain. Then $\langle R\rangle_f$ denotes the smallest relation that contains R and is closed under f. We write simply '$\langle R\rangle$' when f is understood.

Let $\binom{[m]}{k}$ be the set of $\binom{m}{k}$ sequences $i_1\ldots i_k$ where $1 \le i_1 < \cdots < i_k \le m$. The *$k$-wise projections* of an m-ary relation are the relations $\Pi_\sigma\,R$, for $\sigma \in \binom{[m]}{k}$.

If $R' \subseteq R$ and f is a polymorphism for R, we say that R' is a *seed* of R with respect to f if $\langle R'\rangle_f = R$. NU polymorphisms have nicely identifiable seeds:

Lemma N. *If R has a k-ary NU polymorphism f, then R' is a seed of R with respect to f whenever R and R' have the same $(k-1)$-wise projections.*

Proof. Rule (182) takes us from the equality of $(k-1)$-wise projections to the equality of k-wise, …, m-wise projections. (See exercise 455.) ▮

polymorphisms, exploitation of–
data compression
max-closed
min operation
empty set
near unanimity polymorphisms–
online
polynomial time
thought experiment
$\langle R\rangle_f$
$\binom{[m]}{k}$
k-wise projections
seed

(Incidentally, Lemma N proves that every m-ary relation with a k-ary near uniform polymorphism is *decomposable*, as the intersection of its $\binom{m}{k-1}$ projections into $(k-1)$-ary relations. Conversely, any decomposable relational clone of that kind has a k-ary NU polymorphism. See K. Baker and A. Pixley, *Math. Zeitschrift* **143** (1975), 165–174, Theorem 2.1; P. Jeavons, D. Cohen, and M. C. Cooper, *Artificial Intelligence* **101** (1998), 251–265, Theorem 3.5.)

Let's consider a smallish example, with $D = \{0, 1, 2, 3\}$, $k = 3$, $m = 9$, and

$$f(x, y, z) = \bigl(x, y, z \text{ distinct? } 0\text{: } \langle xyz\rangle\bigr). \qquad (191)$$

This function evaluates to zero whenever its value isn't prescribed by (182). The following relation R, with 108 9-tuples, has f as one of its polymorphisms:

000100000	000220000	002100300	003100010	022200002	100220202	122200002	300100330	301212022
000100010	000220002	002100330	003100030	022200030	102200002	122200202	300130310	302100000
000100030	000220300	002200000	003200000	022220000	102200202	122220002	300200000	302100030
000100300	001200000	002200002	003200002	022220002	102210002	122220202	300200002	302100300
000100310	001200002	002200030	003200030	023200000	102220002	123200002	300200030	302100330
000100330	001200300	002200300	011220300	023200002	102220202	123200202	300200300	302100333
000200000	001210002	002200330	020200000	023200030	103200002	203103010	300200330	302200000
000200002	001220000	002210002	020200002	023200031	103200202	300100000	300210002	302200002
000200030	001220002	002220000	020200030	100200002	120200002	300100010	301200000	302200030
000200300	001220300	002220002	020220000	100200202	120200202	300100030	301200002	302200300
000200330	002100000	002220300	020220002	100210002	120220002	300100300	301200300	302200330
000210002	002100030	003100000	022200000	100220002	120220202	300100310	301210002	302210002

In fact, R is seeded by nine tuples of the radix-4 expansion $\pi = (3.02100333\ldots)_4$:

$$R = \langle\{302100333, 122220202, 011220300, 203103010, \ldots, 103200202\}\rangle. \qquad (192)$$

One of R's more understandable seeds is the following subset of 17 tuples inspired by Lemma N,

$$\begin{aligned} R' \;=\; \{&000100010, 000200330, 000210002, 011220300, 020220000,\\ &022220002, 023200031, 100220202, 102210002, 103200202, 122220202,\\ &203103010, 300130310, 301200000, 301212022, 302100000, 302100333\}. \end{aligned} \qquad (193)$$

The original nine tuples of (192) all appear in R', together with eight others.

We call R' a "stub," because its pairwise (2-wise) projections match those of R, and this property makes it easy to deduce R from R'. For example, $\Pi_{12}R' = \{00, 01, 02, 10, 12, 20, 30\} = \Pi_{12}R$; $\Pi_{28}R' = \{01, 03, 00, 10, 20, 23, 02\} = \Pi_{28}R$; and so on for all Π_{ij} with $1 \le i < j \le 9$. According to Lemma N, a given tuple τ belongs to R if and only if each of its 36 projections $\Pi_{ij}\tau$ belongs to $\Pi_{ij}R'$.

Furthermore R' is a *minimal* stub: If we remove any one of those 17, we lose at least one of the projections Π_{ij}. For example, 000100010 is the only tuple τ of R' for which $\Pi_{14}\tau = 01$ or $\Pi_{18}\tau = 01$.

In general, when R is an m-ary relation that's preserved by a k-ary NU polymorphism, a *stub* for R is a subset R' whose $(k-1)$-wise projections agree with those of R. Although R can have up to d^m elements, in a domain of size d, a minimal stub for R always has at most $\binom{m}{k-1}d^{k-1}$ elements. For example, if $d = 10$, $m = 20$, and $k = 4$, any closed relation of size up to 10^{20} can be represented by a minimal stub of size $\le$ 1,140,000.

decomposable
Baker
Pixley
Jeavons
Cohen
Cooper
median operator
majority operator
pi as source
stub
minimal stub

Let $R_0 = \{000100000, \ldots, 023200031\}$ be the 56 elements of R that begin with 0. Then R_0 is preserved by f. We should be able to derive a stub R'_0 for R_0 by looking at R', because R' is a stub for R. What's a good way to do that? witnesses compact stub–

You might think that it simply suffices to let R'_0 be the tuples of R' that begin with 0. If so, you would be wrong. For example, $02 \in \Pi_{25}R_0$; but 100220202 is the only tuple in R' whose projection Π_{25} is 02. Thus R'_0 *can't* be a subset of R'.

One way to compute a valid stub R'_0 is based on the crucial observation that

$$\Pi_\sigma \langle R\rangle_f \;=\; \langle \Pi_\sigma R\rangle_f, \quad \text{for all relations } R \text{ and operations } f. \tag{194}$$

The process of going from R to $\langle R\rangle_f$ works componentwise on the tuples. So when we watch it through a "window" that projects the action onto only some subset σ of the components, any actions that take place beyond what we can see in the window have no influence on what we do see.

By definition, a stub R'_0 must have the same projections Π_{ij} as R_0 does. So we want the union, over $1 \le i < j \le 9$, of enough tuples to achieve this. Consider, for example, the case $ij = 23$. Rule (194) tells us that

$$\langle \Pi_{123}R'\rangle \;=\; \Pi_{123}\langle R'\rangle \;=\; \Pi_{123}R. \tag{195}$$

Therefore, if we carry out the completion process (144) on the 3-tuples $\Pi_{123}R'$, and examine the 3-tuples of $\langle \Pi_{123}R'\rangle$ that begin with 0, we'll see all the projections $\Pi_{23}R_0$. (Think about it.)

Let's look closer. Although R' has seventeen tuples, its projection $\Pi_{123}R'$ has only thirteen: {000, 011, 020, 022, 023, 100, 102, 103, 122, 203, 300, 301, 302}. And the completion $\langle \Pi_{123}R'\rangle$ adds five more: {001, 002, 003, 120, 123}. Hence we know that $\Pi_{23}R_0$ contains 01, 02, and 03.

The completion process implicitly gives us enough information to identify "witnesses" for 01, 02, and 03, namely some elements of R_0 that contribute those projections to the set Π_{23}. For example, the triple 001 was implied by 301, 011, and 023 using (144); and those triples were projections of 301212022 (or 301200000), 011220300, and 023200031, respectively. Therefore, applying f, we know that 001200000 is in R_0; and we can contribute that tuple to R'_0. Similarly, we obtain witnesses for 02 and 03 by contributing 002200300 and 003200000.

Carrying out this process for all i and j yields a suitable stub,

$$\begin{aligned} R'_0 \;=\; \{&000100010, 000100300, 000100310, 000200000, 000200330,\\ &000210002, 000220000, 001200000, 001210002, 002100000, 002200030,\\ &002200300, 002210002, 003100010, 003100030, 003200000, 003200002,\\ &011220300, 020220000, 022220002, 023200031\}. \end{aligned} \tag{196}$$

OK, this stub is valid; but it's not minimal. Indeed, a tedious computation shows that we could omit 000100010, 000100300, 000200000, and 003200000 without losing any of the pairwise projections. Algorithm N uses a weaker criterion that's much easier to maintain: If we regard a stub as an *ordered* list of tuples, it is called *compact* if each item contributes at least one new projection that isn't present in any previous item on the list. Consequently *a compact stub for an m-ary relation has at most $\binom{m}{k-1}d^{k-1}$ elements*; and that's the same bound as we had for minimal stubs. The stub (196) is compact, when ordered as shown.

The main data structure of Algorithm N is a compact stub called S, which represents all of the solutions to the constraints that have been seen so far. The phrase "contribute τ to S," where τ is a tuple, is a shorthand for the following operation: "If all of the $(k-1)$-wise projections of τ appear among the existing $(k-1)$-wise projections of S, do nothing. Otherwise set $S \leftarrow S \cup \{\tau\}$."

Here now is the promised "thought experiment."

data structure
contribute
thought experiment
complete relation
witnesses
decomposable

Algorithm N (*Near unanimity CSP solver*). Let $f(x_1, \ldots, x_k)$ be a k-ary near unanimity operation on the domain $D = \{0, 1, \ldots, d-1\}$, and let $n > k \geq 3$. This online algorithm inputs a sequence of zero or more f-invariant constraints on n variables that all have domain D. It maintains a compact stub S for the n-ary relation T, the set of all solutions to the problem-so-far, and terminates unsuccessfully if those constraints are unsatisfiable.

N1. [Initialize.] Set S empty. Then, for each combination $i_1 \ldots i_{k-1} \in \binom{[n]}{k-1}$, and each d-ary string $d_1 \ldots d_{k-1} \in D^{k-1}$, contribute to S the n-tuple $t_1 \ldots t_n$ that has $t_{i_1} = d_1, \ldots, t_{i_{k-1}} = d_{k-1}$, and 0s elsewhere. (Now S is a compact stub for the complete relation D^n. Exercise 461 has a generalization.)

N2. [Get new constraint.] If there's no more input, terminate with a solution τ, where τ is any element of S. Otherwise input a new constraint R, where $R \subseteq D^r$ is an r-ary relation that constrains variables $j_1, \ldots, j_r$. (We assume that $1 \leq r < k$ and that f is a polymorphism of R.) Let ρ be the string $j_1 \ldots j_r$.

N3. [Prepare to constrain.] Set $S_0 \leftarrow S$, $S \leftarrow \emptyset$. Do step N5 for every $\sigma \in \binom{[n]}{k-1}$.

N4. [Satisfiable?] (Now S is a compact stub for the n-ary relation

$$T \;=\; \langle S_0 \rangle \cap R(j_1, \ldots, j_r), \tag{197}$$

the set of all tuples that satisfy the former constraints as well as the new one.) Terminate unsuccessfully if $S = \emptyset$. Otherwise return to N2.

N5. [Contribute Π_σ.] If $\sigma = i_1 \ldots i_{k-1}$, let $\hat\sigma$ be the string $\sigma\rho = i_1 \ldots i_{k-1} j_1 \ldots j_r$, except omit any j's that occur earlier in the string. (For example, a case like $\sigma = 234789$ and $\rho = 31415$ would imply $\hat\sigma = 23478915$.) Run through all tuples τ of $\langle \Pi_{\hat\sigma} S_0 \rangle$, and contribute to S a tuple of $\langle S_0 \rangle$ that witnesses τ whenever $\Pi_{\hat\rho}\tau$ satisfies R. (See below, where $\hat\rho$ is explained and this process is spelled out. In the example where $\hat\sigma = 23478915$ we have $\hat\rho = 27378$.) ▮

It's important to notice the assumption '$r < k$' in step N2. Any r-ary constraint with $r \geq k$ can be rewritten as a sequence of $(k-1)$-ary constraints, because it is decomposable as mentioned above. So we don't need to deal with large r.

This algorithm basically uses the methodology by which we deduced (196) from (193) while introducing the concept of a stub. But it deals with relations and NU polymorphisms in full generality, instead of with the special case where R is the unary relation '$x_1 = 0$' and f is the particular 3-ary polymorphism (191).

Step N5 deserves to be elaborated carefully. It finds all projections of the relation T in (197) onto $k-1$ variables $\sigma = i_1 \ldots i_{k-1}$, when we're imposing a new constraint R on $r < k$ variables $\rho = j_1 \ldots j_r$, where σ and ρ are given strings.

polynomial time
Maltsev
forking
transitive law
equivalence relation
set partition

To do this, we examine all of the l-wise projections $\Pi_{\hat\sigma}$ of the former relation $T_0 = \langle S_0\rangle$, where $l = |\hat\sigma| = |\{i_1,\ldots,i_{k-1},j_1,\ldots,j_r\}|$. Notice that $k-1 \le l \le k-1+r \le 2k-2$; the actual value of l depends on overlaps between the i and j coordinates. There's a string $\hat\rho$ such that, if $\tau = t_1\ldots t_l$ is an l-tuple of $\Pi_{\hat\sigma}T_0$, the r-tuple that's subject to relation R is $\Pi_{\hat\rho}\,\tau$.

We keep a dictionary H, initially empty, of all the projected l-tuples that we know belong to $\Pi_{\hat\sigma}T_0$. Every l-tuple τ in H will in fact be the projection $\Pi_{\hat\sigma}\tau^*$ of some known n-tuple τ^* in T_0. First we seed H by contributing all of the l-tuples $\Pi_{\hat\sigma}S_0$, one by one, discarding any l-tuples that are already present. If we find that $\Pi_{\hat\rho}\tau \in R$, when a new l-tuple $\tau = \Pi_{\hat\sigma}\tau^*$ enters H from some $\tau^* \in S_0$, we also contribute τ^* to S, the master compact stub.

After this seeding is complete, we apply (144) repeatedly until no more l-tuples can be contributed to H; exercise 463 has the details. If m is the final size of H, we will have applied (144) m^k times, where $m \le d^l$.

Algorithm N clearly runs in polynomial time, as promised. The bottleneck is step N5, which takes a bounded amount of time independent of n; and that step is performed $O(n^{k-1})$ times for each constraint that is input.

The same idea that propels Algorithm N leads to another interesting method, Algorithm M, which exploits a special kind of polymorphism named after Anatoly Maltsev. A *Maltsev polymorphism*, f, is a ternary operation that satisfies two identities:

$$f(x,x,y) \;=\; y \;=\; f(y,x,x). \tag{198}$$

Thus it favors y, the "minority" parameter — in contrast with a near unanimity operator, which opts for the "majority" parameter x. (Compare with (182).)

There are lots more Maltsev operations than ternary NU operations, because (198) places no condition whatsoever on $f(x,y,x)$. However, somewhat paradoxically, the number of constraints preserved Maltsevwise is dramatically smaller than the number of constraints preserved NU-wise. It's a risky business to oppose the majority, even if we do it only 2/3 of the time! For example, 412922 of the 2^{27} ternary relations on $\{0,1,2\}$ have at least one ternary NU polymorphism; but only 2756 of them have at least one that satisfies (198). (See exercise 464.)

Suppose R is an m-ary relation over a domain D. A level-i *forking* of R is a triple (i,a,a'), with $1 \le i \le m$ and $a,a' \in D$, such that R contains tuples $\tau = t_1\ldots t_m$ and $\tau' = t'_1\ldots t'_m$ with equal prefixes of length $i-1$:

$$t_1 = t'_1,\;\ldots,\; t_{i-1} = t'_{i-1},\; t_i = a, \text{ and } t'_i = a'. \tag{199}$$

In particular, R has a forking (i,a,a) with $\tau = \tau'$ for each a in its projection $\Pi_i\,R$.

When R is closed under a Maltsev polymorphism, f, its level-i forkings satisfy the transitive law: *(i,a,a'') is a forking whenever both (i,a,a') and (i,a',a'') are forkings.* For if $x_1\ldots x_{i-1}\,a x_{i+1}\ldots x_m \in R$, $x_1\ldots x_{i-1}a' y_{i+1}\ldots y_m \in R$, and $z_1\ldots z_{i-1}a' z_{i+1}\ldots z_m \in R$, we have $z_1\ldots z_{i-1}a w_{i+1}\ldots w_m \in R$, by (144) and (198), where $w_j = f(x_j,y_j,z_j)$. The level-i forkings therefore define an *equivalence relation*, a partition of the set $\Pi_i\,R$.

trie structure
forest
bitwise parity constraint
parity constraint
data compression
Maltsev stub
compact

Let's imagine that the tuples of R are represented in a trie structure — a forest with multiway branching, such as Fig. 31 in Section 6.3. The roots of this forest are the values $\Pi_1 R$ that can be first in a tuple. The nodes at distance $i-1$ from a root all have branches that correspond to one of the equivalence classes for level-$(i{+}1)$ forking, when $i < m$.

That's a strong structural condition; it explains why comparatively few relations are closed under any of the numerous Maltsev operations. Consider, for example, the following relation P, shown as a trie:

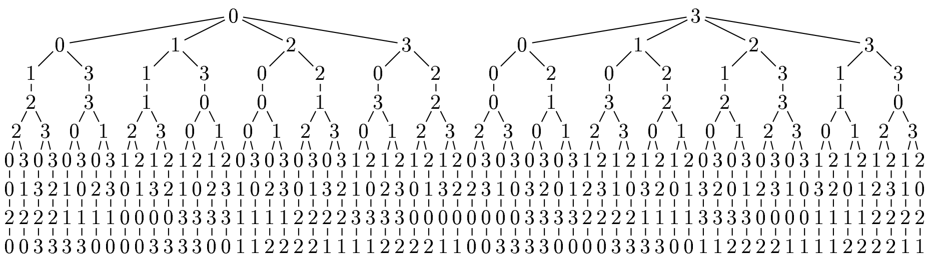


Fig. 120. A 9-ary relation on $\{0,1,2,3\}$, closed under the Maltsev constraint (200).

The 64 tuples of Fig. 120 are preserved by the "bitwise parity constraint"

$$p(x,y,z) \;=\; x \oplus y \oplus z, \tag{200}$$

which obviously satisfies the Maltsev conditions (198). For example, the tuples 001220020, 302100333, and 022130321 all belong to P; and so does 321210032.

In general, Maltsev operations provide excellent data compression, and have seeds that are simpler than those guaranteed for NU operations by Lemma N:

Lemma M. *If R has a Maltsev polymorphism f, then R' is a seed of R with respect to f whenever R and R' have the same forkings.*

Proof. The level-1 forkings tell us that $\Pi_1 R' = \Pi_1 R$. And subsequent forkings imply that $\langle \Pi_{12\ldots i} R'\rangle = \Pi_{12\ldots i} R$ for $i = 2, \ldots, m$. (See exercise 467.) ▮

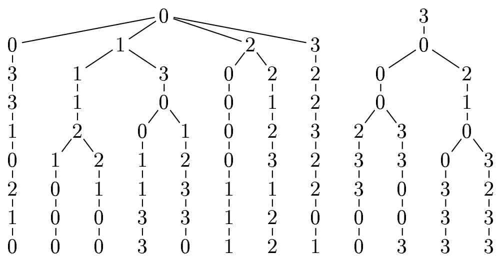
Therefore, for use in Theorem M, we define a *Maltsev stub* of a Maltsev-preserved relation R to be a subset $R' \subseteq R$ whose forkings include all the forkings of R. And as before, we consider R' to be an ordered list of tuples, and call it *compact* if it satisfies a local condition that bounds the list size.

The local condition in this case needs to be slightly trickier than before, because a forking can involve two distinct tuples, while a projection involves only one. Here's the new definition: The tuples $\tau_1, \tau_2, \ldots, \tau_M$ of R are a compact Maltsev stub for R if, for $1 < j \le M$, $\{\tau_1, \ldots, \tau_j\}$ has at least two more forkings than $\{\tau_1, \ldots, \tau_{j-2}\}$. (The two new forkings typically have the form (i, a, a') and (i, a', a), although other cases can arise; see exercise 469. The ordering of the τ's need not be lexicographic.) The good news is that the total number of possible choices of (i, a, a') is quite small. Consequently *a compact Maltsev stub for an m-ary relation on a d-ary domain has at most md^2 elements.*

For example, one of many compact stubs for the relation P of Fig. 120 is

$$\begin{aligned}P' \;=\; \{&003310210, 011121000, 011122100, 013001133,\\ &013012330, 020000111, 022123122, 032232201,\\ &300023300, 300033003, 302100333, 302103233\}. \qquad (201)\end{aligned}$$

It satisfies the compactness condition, but it's not minimal; exercise 468 shows that one of its twelve tuples could be removed without losing any of the forkings. When P' is drawn as a trie, it looks like this:



To get some experience with Maltsev stubs, let's look at how we could use P' to construct a stub for the relation P_3, the set of P's tuples that begin with 3. (The procedure will be analogous to our previous construction of (196) from (191) and (193).) The idea is to examine each forking (i, a, a') of P', and to retain it if it is also a forking of P_3.

Consider, for example, the case $(6, 1, 2)$, which is a forking exhibited by $\tau' = 011121000$ and $\tau'' = 011122100$ in P'. Exercise 470 proves that P_3 will contain this forking if and only if P contains a tuple $\tau = t_1 \ldots t_m$ such that $t_1 = 3$ and $t_6 = 1$. And it's easy to determine $\Pi_{16} P$: We just compute $\langle \Pi_{16} P' \rangle_p$, using (194).

Indeed, $\Pi_{16} P' = \{00, 01, 02, 03, 30, 33\}$. Applying p tells us that $\langle \Pi_{16} P' \rangle_p$ *does* contain 31; for example, $p(30, 00, 01) = 31$. Hence, without projecting,

$$\tau = p(302100333, 003310210, 011121000) = 310331123$$

is a suitable tuple. We've shown that P_3 *does* contain the forking $(6, 1, 2)$ that was present in P. And we can retain it by contributing both τ and $p(\tau, \tau', \tau'') = 310332023$ to our stub for P_3.

A similar calculation on the whole set of forkings yields, for example, the stub

$$\begin{aligned}P'_3 \;=\; \{&300023300, 302100333, 302113130, 310331123, 310332023,\\ &312212110, 323333001, 331101311, 333021222\}. \qquad (202)\end{aligned}$$

Mission accomplished: We've achieved our goal of passing from P to P_3. The next challenge is to use the stub P' to construct a stub for the relation P_{-2}, which is the set of all tuples in P that *end* with 2. Exercise 472 considers how to do that, in general. And armed with that knowledge, a diligent reader is prepared to go the rest of the way and to solve exercise 473, which is to design Algorithm M — the desired partner to Algorithm N.

Trie structures turn out to be convenient for representing not only the stubs but also the tuples of the constraints in the input.

***Visiting all solutions.** We've just seen two classes of CSPs for which it's possible to represent the set of all solutions by a stub of polynomial size, and to do basic operations on stubs in polynomial time. One of those basic operations is to "fix a variable," namely to enforce a unary constraint '$v = a$', where v is a variable and a is an element of v's domain. As a consequence, we're able to visit *every* solution efficiently, one by one, if "efficient" means "polynomial-time delay between visits," using the following generic algorithm:

assignment
fix a variable, see Assignment
unary constraint
polynomial-time delay
generating all solutions, see Visiting
lexicographic order
tractable
core
unary polymorphism
WNU polymorphism
academic

Algorithm V (*Visiting every solution to a CSP*). Given a CSP $\mathcal{P}$ with variables $\{v_1, \ldots, v_n\}$ over the domain $D = \{0, 1, \ldots, d-1\}$, let T_0 be the set of all n-tuples $a_1 \ldots a_n$ such that $(v_1 = a_1, \ldots, v_n = a_n)$ solves $\mathcal{P}$. This algorithm visits the elements of T_0 in lexicographic order, assuming that we have data structures and algorithms with which we can pass from a representation S of any relation T that arises with $\mathcal{P}$ to a representation S' of any relation T' that is obtained from T by fixing a variable. We assume that $S = \emptyset$ if and only if $T = \emptyset$.

V1. [Initialize.] Set $a_1 \ldots a_n \leftarrow 0 \ldots 0$, and set S_j to a representation of

$$T_j \;=\; T_0 \cap (v_1 = a_1) \cap \cdots \cap (v_j = a_j), \tag{203}$$

for $j = 0, 1, \ldots, n$.

V2. [Solution found?] If $S_n \neq \emptyset$, visit the solution $a_1 \ldots a_n$.

V3. [Done?] (At this point we have visited every solution to $\mathcal{P}$ that is lexicographically less than or equal to $a_1 \ldots a_n$.) Let j be maximum such that $j \le n$, $j > 0$, $S_{j-1} \neq \emptyset$, and $a_j \neq d-1$. Terminate if no such j exists.

V4. [Advance.] Set $a_j \leftarrow a_j + 1$ and $a_{j+1} \ldots a_n \leftarrow 0 \ldots 0$. Update the representations $S_j, \ldots, S_n$ so that (203) is again true. Return to V2. ∎

Algorithm V can be used not only with the special CSPs that correspond to Algorithms M and N, but also with *any* CSP whose set Γ of relations is tractable, provided that Γ is a *core* — that is, provided that every unary polymorphism of Γ is a permutation of D. For in that case, exercise 443 shows that Γ has a WNU polymorphism; and a WNU polymorphism is closed under fixing a variable. Thus Theorem D provides the polynomial-time tools that Algorithm V needs.

On the other hand, we can't use Algorithm V to get polynomial-time behavior with an arbitrary CSP that's based on a tractable constraint such as

$$Y \;=\; \{001, 010, 100, 222\}. \tag{204}$$

The set $\Gamma = \{Y\}$ is not a core, because $f(x) = 2$ is one of its unary polymorphisms. And we could solve NP-complete problems with Algorithm V by ignoring the solution $2 \ldots 2$.

The situation gets more interesting when we have a constraint like

$$Y^+ = \{001, 010, 100, 222, 223, 232, 233, 322, 323, 332, 333\}. \tag{205}$$

Then it *is* possible to visit all of the solutions with polynomial-time delay between visits, because the number of solutions is gigantic. (See exercise 475.) Of course this particular relation Y^+ is purely academic, not of practical interest.

A brief history. The notion of "constraint satisfaction problems" was introduced and named by Richard E. Fikes in *Artificial Intelligence* **1** (1970), 27–120, 299. He implemented an elaborate system that generated a sequence of CSPs from a given nondeterministic program in a fairly general language; the goal was to solve one or more of the resulting CSPs. His system included more than a dozen constraint manipulation methods by which it was possible to eliminate variables and/or to reduce their domains and/or to discover contradictions.

historical remarks–
Fikes
Montanari
pixel
identity relation
0s and 1s
Boolean matrix multiplication
path consistency
all shortest paths
shortest paths
Floyd
Gaussian elimination
Huffman
Clowes
line labeling
Waltz
shadows

Before the 1970s, a search for combinatorial patterns was generally specified by prescribing one or more *global* constraints that the variables of a problem were supposed to satisfy. A more nuanced understanding, by which such objectives could often best be regarded as networks of *local* constraints, was then formulated by Ugo Montanari in *Information Sciences* **7** (1974), 95–132.

Montanari limited his discussion to the special case in which all constraints are binary. In other words, he considered n-tuples $(x_1, \ldots, x_n)$ such that $x_j \in D_j$ for $1 \le j \le n$, and such that $(x_i, x_j) \in R_{ij}$ for certain ordered pairs (i, j), where each D_j was a given finite set and each $R_{ij} \subseteq D_i \times D_j$ was a given binary relation. He'd been working with digitized pictures, containing $n \approx 1000$ pixel values x_j, where each domain D_j had roughly 20 values. In such problems he expected most of the constraints to involve geometrically adjacent pixels x_i and x_j, so that only $O(n)$ or $O(n \log n)$ relations would need to be specified. His goal was to reduce the search space by doing some sort of preprocessing to simplify them.

He required each relation R_{ii} between a variable and itself to be a subset of the identity relation $x = y$; but (curiously and unnecessarily) he allowed R_{ij} and R_{ji} to be independent of each other. His main contribution was the following algorithm to refine the given network of relations:

$$\text{For } 1 \le k \le n, \text{ set } R_{ij} \leftarrow R_{ij} \cap R_{ik} R_{kk} R_{kj} \text{ for } 1 \le i, j \le n. \qquad (206)$$

Here each R_{ij} is regarded as a $|D_i| \times |D_j|$ matrix of 0s and 1s, and the matrix multiplication is Boolean (namely ORs of ANDs, not sums of products). If any R_{ij} is changed by this process, the entire computation (206) is supposed to be repeated, until no further changes occur. Finally a form of path consistency will have been achieved (see exercise 479).

Algorithm (206) was inspired by an algorithm for all shortest paths due to R. W. Floyd [*CACM* **5** (1962), 345], which in turn was related to the solution of simultaneous linear equations by Gaussian elimination. It's *not* very efficient; notice, for example, that it may well constrain variables that were initially unconstrained, because R_{ij} might change from $|D_i| \times |D_j|$ to something smaller. But it was a start, and it encouraged other researchers to find improvements.

Meanwhile, as we have seen, D. A. Huffman and M. B. Clowes had independently come up with an interesting system of constraints, both binary and ternary, between adjacent lines in digitized images. Their ideas about line labeling were considerably extended by D. L. Waltz, who showed how to deal not only with the edges of polyhedra but also with the complex *shadows* that are cast by such objects. [See his Ph.D. thesis (MIT report AI-TR-271, November

1972), 349 pages; it was partially summarized in *The Psychology of Computer Vision*, edited by P. Winston (McGraw–Hill, 1975), 19–91.] Waltz found that the propagation of such local constraints led to enormous speedups in the recognition of scenes, and his approach became known as the "Waltz filter."

Another early study that generated considerable interest was R. M. Stallman and G. J. Sussman's investigation of computer-aided circuit analysis via "forward reasoning and dependency-directed backtracking" [*Artificial Intelligence* **9** (1977), 135–196]. Their paper introduced a primitive learning mechanism called NOGOOD assertions — sets of assignments that are incompatible because they lead to a contradiction. But they made little use of such assertions at that time, and didn't foresee any need to discard them when they accumulate. (With hindsight, of course, we now know that a generalized form of clause learning and forgetting is extremely important, as in Algorithm 7.2.2.2C.)

But let's backtrack. Several years before computer scientists had been attaching interesting symbolic labels to lines in scenes, combinatorial mathematicians had been attaching interesting numbers to the vertices of graphs. Alexander Rosa published an influential paper [in *Theory of Graphs* (Paris: Dunod, 1967), 349–355], based on his dissertation written in 1965, which introduced four kinds of labelings called α-valuations, β-valuations, σ-valuations, and ρ-valuations. Every α-valuation was a β-valuation; every β-valuation was a σ-valuation; every σ-valuation was a ρ-valuation; and every ρ-valuation was enough to show that the m edges of the underlying graph could cover all edges of the complete graph K_{2m+1} in rainbow fashion, when rotated cyclically as in Fig. 110(c).

S. W. Golomb began to think about graph labels independently, because he wanted a convenient way to identify the terminals of communication networks and the interconnections between them. He decided to call a graph "graceful" if it had an ideal labeling by his criterion; and of course he told his good friend Martin Gardner about those ideas. Martin wrote about "The graceful graphs of Solomon Golomb, or how to number a graph parsimoniously" in *Scientific American* **226**, 3 (March 1972), 108–112; Golomb's own publication appeared at about the same time in *Graph Theory and Computing* (Academic Press, 1972), 23–37. People soon discovered that Rosa's β-valuations were exactly the same as Golomb's graceful labelings, and interest in the subject began to take off.

Rosa's ρ-valuations eventually became known as "rainbow graceful" — a nice coincidence, because "ρ" stands for both "rainbow" and "Rosa."

The first significant algorithm for subgraph isomorphism was developed by E. H. Sussenguth, Jr., motivated by queries to databases of chemical compounds [*J. Chemical Documentation* **5** (1965), 36–43]. He considered induced subgraphs of labeled structures, and based his method on supplemental labels that he called "properties," such as the length of a shortest cycle (if any) from a vertex to itself. His implementation used bitwise operations to represent the sets of pattern and target vertices that have various combinations of label values. Several years later, J. R. Ullmann independently described bitwise techniques for finding *non*-induced copies of a given pattern in a given target [*JACM* **23** (1976), 31–42].

Winston
propagation
Waltz filter
Stallman
Sussman
circuit analysis
learning
nogood
clause learning
Rosa
Golomb
networks
Gardner
rainbow graceful
subgraph isomorphism
Sussenguth
chemical compounds
induced subgraphs
supplemental labels
bitwise operations
Ullmann

Ullmann obtained domain consistency for binary constraints by repeating the operation

$$\text{revise}(R_{ij}, x_i) = \begin{cases} \text{For each } a \in D_i, \\ \quad \text{if } D_j \,\&\, (\text{row } a \text{ of } R_{ij}) = 0, \\ \quad \text{set } D_i \leftarrow D_i \setminus a, \end{cases} \tag{207}$$

where D_j and the rows of R_{ij} are bit vectors. (Compare with (90) and (206).) Then J. J. McGregor, in *Information Sciences* **19** (1979), 229–250, observed that another procedure is faster when $|D_j| < |D_i|$:

$$\text{revise}(R_{ij}, x_i) = \begin{cases} \text{Set } z \leftarrow 0 \text{ and, for each } b \in D_j, \\ \quad \text{set } z \leftarrow z \mid (\text{column } b \text{ of } R_{ij}); \\ \text{then set } D_i \leftarrow D_i \,\&\, z. \end{cases} \tag{208}$$

The reduction of domains via forward consistency was called "preclusion" by Golomb and Baumert in their classic paper on backtracking [*JACM* **12** (1965), 516–524]. It eventually became prominent under the name "forward checking," following an influential study by Robert M. Haralick and Gordon L. Elliott [*Artificial Intelligence* **14** (1980), 263–313].

The more powerful notion of *domain consistency* was first formulated in general by John Gaschnig [*Proceedings of the Annual Allerton Conference on Circuit and System Theory* **12** (1974), 866–874], inspired by the work of Fikes and Waltz. Gaschnig focused on binary constraints; Alan K. Mackworth extended the theory to k-ary constraints in *IJCAI* **5** (1977), 598–606. (For technical reasons he called it "arc consistency.") Gaschnig made extensive tests, as part of his thesis work at Carnegie-Mellon University [Report CMU-CS-79-124 (1979), Chapter 4], and was disappointed to learn that the n queens problem was *not* solved faster when domain consistency was maintained.

Dozens of algorithms for achieving and maintaining domain consistency have been proposed since then. An excellent survey of those developments, including also a discussion of many stronger notions of consistency, has been prepared by Christian Bessière, in *Handbook of Constraint Programming* (2006), 29–83. Algorithm D, which features time stamps and a queue of variables to check, is based on a procedure by Christophe Lecoutre [*Constraint Networks* (2009), §4.1.2]. Algorithm S incorporates ideas from Bessière's AC6 algorithm [*Artificial Intelligence* **65** (1994), 179–190] and an algorithm that C. Lecoutre and F. Hemery called AC3rm [*IJCAI* **20** (2007), 125–130]. An important algorithm known as AC2001, which lies "between" AC6 and AC3rm, was devised in 2001 by C. Bessière and J.-C. Régin and independently at the same time by R. H. C. Yap and Y. Zhang; it was eventually published by all four authors in *Artificial Intelligence* **165** (2005), 165–185. AC2001 was the first algorithm for domain consistency that maintained supports for each (variable, value) pair.

All of the early programs for CSP solving were essentially based on backtracking with d-way branching. If it became necessary to backtrack after exploring the possibility that $v = a$, for some element a in the current domain of a variable v, the only reasonable next step seemed to be to look at the case $v = a'$, for some other element of v's domain, and so on, until all possible values

domain consistency
binary constraints
McGregor
forward consistency
preclusion
Golomb
Baumert
forward checking
Haralick
Elliott
domain consistency
Gaschnig
Fikes
Waltz
binary constraints
Mackworth
k-ary constraints
arc consistency
n queens problem
consistency
Bessière
time stamps
Lecoutre
AC6
Hemery
AC3rm
AC2001
Bessière
Régin
Yap
Zhang
backtracking
d-way branching

for v had been tried. The first person to realize that '$v \neq a$' might lead to a situation where it's better to branch next on a variable w that's *different* from v, because '$v = a$' had been supporting elements of w's domain in a crucial way, was apparently Daniel Sabin, who mentioned it at a computer conference in 1994 and incorporated it into the design of ILOG Solver. [See page 147 of Jean-Charles Régin's Ph.D. thesis (Université Montpellier II, 1995), vii + 389 pages.]

Two conference papers by Daniel Sabin and Eugene C. Freuder [*European Conference on Artificial Intelligence* **11** (1994), 125–129; *LNCS* **1330** (1997), 167–181], promoting the idea that domain consistency should be maintained throughout the search for solutions, significantly influenced subsequent practice.

The effectiveness of a sparse-set representation for current domains was pointed out in a 12-page note by V. le Clément de Saint-Marcq, P. Schaus, C. Solnon, and C. Lecoutre, presented at a workshop on "Techniques for implementing constraint programming systems" (TRICS) in 2013.

Reversible sparse bitsets were introduced by J. Demeulenaere, R. Hartert, C. Lecoutre, G. Perez, L. Perron, J.-C. Régin, and P. Schaus in *LNCS* **9892** (2016), 207–223, as an important component of the Compact-Table data structure that's discussed in exercise 305.

The tractability of implicational constraints was discovered by L. M. Kirousis [*Artificial Intelligence* **64** (1993), 147–160], and independently (but with a weaker runtime guarantee) by M. C. Cooper, D. A. Cohen, and P. G. Jeavons [*Artificial Intelligence* **65** (1994), 347–361]. Jeavons and Cooper went on to introduce max-closed constraints shortly thereafter [*Artificial Intelligence* **79** (1995), 327–339]. Inspired by a paper that introduced Lemma R [P. van Beek and R. Dechter, *JACM* **42** (1995), 543–561], CRC constraints were introduced by Y. Deville, O. Barette, and P. Van Hentenryck [*Artificial Intelligence* **109** (1999), 243–271].

The indicator problem $\mathcal{I}_k(\Gamma)$ was introduced by P. Jeavons, D. Cohen, and M. Gyssens, *LNCS* **1118** (1996), 267–281. Theorem 16 of that paper is a generalization of Theorem F above.

Theoretical underpinnings of what was to become the complexity classification of CSP languages Γ were begun in the context of "multivalued logic." The study of all operators that preserve a given relation was initiated by S. V. Yablonsky [*Trudy Mat. Inst. Steklov.* **51** (1958), 5–142] and pursued extensively in the Soviet Union. R. A. Bayramov [*Doklady Akad. Nauk Azerbaijan* **24**, 2 (1968), 3–6] called the set of all such operators the "stabilizer of a predicate." V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov, writing in a Ukrainian journal [*Kibernetika* **5**, 3 (1969), 1–11; **5**, 5 (1969), 1–10], gave those operators the Russian name 'полиморфизмы' (polymorphisms), and introduced the associated Galois connection between what they called Пол($\widehat{R}$) and Инв($\mathfrak{F}$). R. Pöschel, in an East German journal [*Zeitschrift für mathematische Logik und Grundlagen der Mathematik* **19** (1973), 37–74], took up the torch and added 'Polymorphismen' to the German language. The same Galois connection had been noted independently in America by D. Geiger [*Pacific J. Math.* **27** (1968), 95–100]; but his work remained comparatively unknown for many years.

Sabin
ILOG Solver
Régin
Freuder
maintaining domain consistency
sparse-set
le Clément de Saint-Marcq
Schaus
Solnon
Lecoutre
Reversible sparse bitsets
Demeulenaere
Hartert
Lecoutre
Perez
Perron
Régin
Schaus
Compact-Table
data structure
tractability
implicational constraints
Kirousis
Cooper
Cohen
Jeavons
max-closed constraints
van Beek
Dechter
CRC constraints
Deville
Barette
Van Hentenryck
indicator problem
Jeavons
Cohen
Gyssens
multivalued logic
Yablonsky
Bayramov
stabilizer of a predicate
Bodnarchuk
Kaluzhnin
Kotov
Romov
polymorphisms
Galois connection
Pol
Inv
Pöschel
Geiger

The idea of making the CSP algebraic, by regarding it as the task of finding homomorphisms in relational structures, was pioneered by T. Feder and M. Vardi [*STOC* **25** (1993), 612–622; *SICOMP* **28** (1998), 57–104]. P. Jeavons developed very similar ideas independently [*Theoretical Comp. Sci.* **200** (1998), 185–204]. The Feder–Vardi dichotomy conjecture was widely regarded as the most important unsolved problem in universal algebra until it was finally established by A. A. Bulatov [*FOCS* **58** (2017), 319–330; arXiv:2007.09099 [cs.CC] (2020), 42 pages], and independently by D. Zhuk [*FOCS* **58** (2017), 331–342; *JACM* **67** (2020), 30:1–30:78]. Zhuk devised a simpler method in arXiv:2404.01080 [cs.CC] (2024), 58 pages. The "NP-complete" half of Theorem D had previously been proved by M. Maróti and R. McKenzie, *Algebra Universalis* **59** (2008), 463–489; see also P. Jeavons, D. Cohen, and M. Gyssens, *JACM* **44** (1997), 527–548; A. Bulatov, P. Jeavons, and A. Krokhin, *SICOMP* **34** (2005), 720–742.

A comprehensive exposition of the theory underlying the dichotomy theorem has been prepared by Z. E. Brady, beginning in 2022 and subsequently updated: See `https://notzeb.com/csp-notes.pdf`.

For an excellent survey of the early work on tractable CSPs, see D. Cohen and P. Jeavons, *Handbook of Constraint Programming* (2006), 245–280.

Algorithm N is a simplification of much deeper ideas presented by V. Dalmau [*Logical Methods in Computer Science* **2** (2006), 1–15]. The notion of "stub" in (193) is equivalent to what N. Alon, D. Moshkovitz, and S. Safra called a $(k-1)$-*restriction problem*, in *ACM Transactions on Algorithms* **2** (2006), 153–177. For Lemma M and the notion of "compact stub," see A. Bulatov, H. Chen, and V. Dalmau, *LNCS* **3244** (2004), 365–379. Dalmau presented Algorithm M in *Electronic Colloq. Comput. Complexity*, Report No. 97 (2004), 9 pages. He also suggested Algorithm V, in correspondence with the author during 2024.

Many other historical notes appear with the answers to particular exercises. They can be located by consulting "Historical notes" in the index.

For further reading. The science of constraint satisfaction itself is unconstrained, and continually growing. We've studied many of its fascinating aspects; but there isn't space in these pages to satisfy every reader's appetite. A wealth of additional information exists, much of which can be found in the *Handbook of Constraint Programming*, edited by F. Rossi, P. van Beek, and T. Walsh (Elsevier, 2006), several chapters of which have already been cited. Major textbooks in the field are *Constraint Processing* by Rina Dechter (Morgan Kaufmann, 2003) and *Constraint Networks* by Christophe Lecoutre (Wiley, 2009).

One good way to understand the substantial progress that has been made is to study the results of carefully crafted international competitions, which began in 2005 and have two major threads. The first of these is based on XCSP, an XML format for CSP instances; see C. Lecoutre, O. Roussel, and M. R. C. van Dongen, *Constraints* **15** (2010), 317–326, and `https://xcsp.org/competitions/`. The other, launched in 2008 and repeated annually thereafter, is based on MINIZINC, a higher-level format that is supported by most of the major CSP solvers; see P. J. Stuckey, R. Becket, and J. Fischer, *Constraints* **15** (2010), 307–316, and `https://www.minizinc.org/challenge/`.

homomorphisms
relational structures
Feder
Vardi
Jeavons
dichotomy conjecture
universal algebra
Bulatov
Zhuk
Maróti
McKenzie
Jeavons
Cohen
Gyssens
Bulatov
Jeavons
Krokhin
Brady
Cohen
Jeavons
Dalmau
stub
Alon
Moshkovitz
Safra
restriction problem
Bulatov
Chen
Dalmau
author
Historical notes
Rossi
van Beek
Walsh
Dechter
Lecoutre
XCSP
Lecoutre
Roussel
van Dongen
MINIZINC
Stuckey
Becket
Fischer

Our discussions in this section have largely focused on "local" constraints of small arity. But an examination of the competitions/challenges shows that *global* constraints — which apply to *all* variables, or at least to a large fraction of them — are also of great importance. We've seen an example of this when we investigated GAD filtering, the "globally all different" constraint. Exercises 185 and 186 show that GAD filtering is sometimes able to prune significantly more values from the domains of variables than could be excluded by purely local means. Other global constraints enable similar improvements. Global constraints significantly simplify the modelling of many combinatorial problems; for example, ten lines of MINIZINC code can replace thousands of SAT clauses.

General-purpose CSP solvers provide a framework for independent subroutines or coroutines called *propagators*, some of which enforce local constraints and some of which enforce global constraints, using appropriate data structures. A propagator comes into action when the domain of any of its variables changes; and it attempts to maintain an appropriate level of consistency (e.g., DC as in Algorithm D). Already in June 2014, more than 400 global propagators had been catalogued at the website `sofdem.github.io/gccat/gccat/`, compiled by N. Beldiceanu, J.-X. Rampon, and M. Carlsson. Exercises 480–485 discuss an approach to global constraints based on an abstract mechanism called a "constraint satisfaction automaton" (CSA).

Propagators for complicated constraints often use a weaker form of domain consistency called *bounds consistency*, in which we pretend that the domain of every variable is an interval of integers. Only the minimum and maximum potential elements of domains are maintained. For example, if $2x + 5y = 3z - 1$ and $D_x = D_y = D_z = [1\,..\,4]$, we can reduce D_y to $[1\,..\,1]$ and D_z to $[3\,..\,4]$; the resulting domains are bounds consistent. Three kinds of bounds consistency were defined by C. W. Choi, W. Harvey, J. H. M. Lee, and P. J. Stuckey in *LNCS* **4304** (2006), 49–58; the variant they called "bounds($\mathcal{Z}$)" is now favored.

When looking for a needle in a haystack, we noticed in Section 7.2.2.2 that SAT solvers can often make use of random restarts. The same is true of CSP solvers, of course; and there's a nice new trick: A CSP solver can restart with random partial assignments in such a way that it never retraces previous steps, if it saves nogoods in a clever way. See J. H. M. Lee, C. Schulte, and Z. Zhu, *Proc. AAAI Conference* **30** (2016), 3426–3433.

Another important idea that carries over from SAT solvers to CSP solvers is the notion of *proof logging.* When no solution to a problem has been found, we'd like to have a tangible certificate of unsatisfiability; such a certificate can then be checked independently, with a comparatively simple algorithm, in order to give us more confidence in the result. Proof logging has had a rather late start in CSP solverland, but the following early papers look very promising: S. Gocht, C. McCreesh, and J. Nordström, *LIPIcs* **235** (2022), 25:1–25:18; M. J. McIlree and C. McCreesh, *LIPIcs* **280** (2023), 26:1–26:17; M. Flippo, K. Sidorov, I. Marijnissen, J. Smits, and E. Demirović, *LIPIcs* **307** (2024), 11:1–11:20.

local versus global constraints
arity
GAD filtering
MINIZINC
CSP solvers
propagators
consistency
DC
Beldiceanu
Rampon
Carlsson
constraint satisfaction automaton
automaton
CSA
domain consistency
bounds consistency
interval
Choi
Harvey
Lee
Stuckey
restarts
nogoods
Lee
Schulte
Zhu
proof logging
certificate of unsatisfiability
Gocht
McCreesh
Nordström
McIlree
McCreesh
Flippo
Sidorov
Marijnissen
Smits
Demirović

EXERCISES

1. [*01*] Find all solutions to the CSP in (1) and (2).

2. [*21*] Every 3SAT problem with m clauses on n Boolean variables can be regarded as a CSP with n variables, binary domains, and m ternary constraints. (See (3).)

a) Instead, represent it with m variables, *ternary* domains, and *binary* constraints.

b) What CSP does your method construct from the 3SAT problem R' in 7.2.2.2–(7)?

c) Reduce the number of binary constraints to $3m$, by adding n binary variables.

d) What CSP do you get from 7.2.2.2–(7) now?

3. [*18*] Express the CSP of (1) and (2) as a SAT problem.

4. [*15*] Express the CSP of (1) and (2) as an XCC problem.

5. [*M05*] The Cartesian product D^0 of 0 copies of a set D consists of a single element, the 0-tuple, denoted by ϵ. Describe all of the possible nullary relations.

▶ **6.** [*M16*] When f is a function from a set A to a set B, textbooks of mathematics traditionally say that A is the "domain" and B is the "range." But when h is the function in a CSP that takes i to x_i, the literature of constraint processing traditionally says that x_i lies in the domain — *not* the range! Discuss.

7. [*15*] True or false: If there's a homomorphism from the cycle graph C_9 to a given graph G, that graph must contain either a 3-cycle or a 9-cycle.

▶ **8.** [*M25*] Is it hard to decide if there's a homomorphism from a given graph to C_5?

▶ **9.** [*25*] Explain why the following problems are special cases of the GCP.

a) Does graph $G = (V, E)$ have an independent set of size k? (Can we choose k distinct vertices in G without selecting any neighbors?)

b) Does graph $G = (V, E)$ have a vertex cover of size k? (Are there k vertices that "hit" every edge of G at least once?)

c) Are graphs $G = (V, E)$ and $G' = (V', E')$ isomorphic? (Is there a one-to-one correspondence between their vertices so that $u \mathrel{-\!\!-} v$ in $G \iff h(u) \mathrel{-\!\!-} h(v)$ in G'?)

d) Does graph $G = (V, E)$ have bandwidth k? (Can its vertices be given distinct integer labels so that $u \mathrel{-\!\!-} v$ implies $|h(u) - h(v)| \le k$?)

e) Does the directed graph $G = (V, A)$ have an Eulerian trail? (Can we "walk" through it, traversing every arc exactly once?)

10. [*20*] (P. Jeavons.) The k-tuple $x_1 \ldots x_k$ is said to be *unlike* the k-tuple $x'_1 \ldots x'_k$ if $x_j \ne x'_j$ for $1 \le j \le k$. It's convenient to write '$x_1 \ldots x_k \parallel x'_1 \ldots x'_k$' when this is true.

Let R be a k-ary relation on a set V. What's a "natural" way to understand the significance of a homomorphism from $(V, \ne)$ to $(R, \parallel)$?

▶ **11.** [*M12*] Why is the general combinatorial problem (GCP) a special case of the CSP?

12. [*HM34*] Let $G(z) = G_N(z) = \sum z^{E(\Sigma)}$ be the generating function for energy, summed over all 2^N one-dimensional Ising configurations Σ, as defined in (9).

a) Find a "closed-form" expression for $G(z)$, when B is (i) 0; (ii) arbitrary.

b) What is the *average* energy per particle, $zG'(z)/(NG(z))$, when $z = e^{-\beta}$?

c) Express those quantities asymptotically as $N \to \infty$.

d) Also evaluate $G_k(z) = \sum \sigma_k z^{E(\Sigma)}$, and the "average magnetization" $\frac{1}{N}\sum_{k=1}^{N} \frac{G_k(z)}{G(z)}$.

▶ **13.** [*20*] Is the all-different constraint *really* necessary, when the crystal maze puzzle (11) already has seventeen constraints like (12)? How about when there are just seven constraints like (15)?

3SAT
SAT as CSP
CSP represented as SAT
SAT representation of CSP
CSP represented as XCC
XCC representation of CSP
Cartesian product
0-tuple
nullary relations
domain
range
homomorphism
cycle graph
independent set
vertex cover
isomorphic
bandwidth
Eulerian trail
Jeavons
unlike
Notations: $\parallel$
general combinatorial problem
GCP
generating function
Ising configurations
partition function
asymptotically
magnetization
all-different constraint
crystal maze puzzle

symmetry
car sequencing problem
MCC problem
(p/q)-string
generating functions
extreme
plane partitions
Szilassi
polyhedron

14. [*21*] Since the graph in (11) is symmetric, every essentially different solution to the CSP models in the text will be found four times. Explain how to exploit symmetry.

▸ **15.** [*20*] Express (11) as an exact cover problem with primary items $\{1,\ldots,8,\mathtt{A},\ldots,\mathtt{H}\}$.

16. [*22*] Express (11) as a CSP with only 7 variables. *Hint:* Use edges, not vertices.

17. [*20*] Solve the car sequencing problem of Fig. 100 and (16).

18. [*15*] Why can the solution to exercise 17 assume $f < 5$, in the text's formulation?

▸ **19.** [*M25*] The redundant constraints in (18) are asymmetrical: They all apply at the *left* of the sequence, because they involve f_{0k}. We could generalize them, and require

$$f_{(l'q_k)k} + f_{(l'q_k+1)k} + \cdots + f_{(t-l''q_k-1)k} \ge r_k - (l'+l'')p_k$$

in the "middle" of the sequence, where $l'+l'' < \lceil r_k/p_k \rceil$. Would that be a good idea?

▸ **20.** [*21*] Express the car sequencing problem as an MCC problem without using colors.

21. [*21*] Improve the previous answer by incorporating the redundant constraints (18).

▸ **22.** [*20*] Extend (16) to two new types of car: Model G has premium audio and heated seats only; Model H is "loaded" with every feature *except* heated seats. Then the 30 cars $\{7\cdot\text{A}, 2\cdot\text{B}, 5\cdot\text{C}, 4\cdot\text{D}, 4\cdot\text{E}, 2\cdot\text{F}, 4\cdot\text{G}, 2\cdot\text{H}\}$ have overall requirements $(r_0,\ldots,r_4) = (15, 20, 10, 12, 6)$, which are the maximum that could conceivably be installed in 30 cars.

Does that "tight" car sequencing problem have a solution? Answer this question by applying Algorithm 7.2.2.1M to the MCC encoding of (a) exercise 20; (b) exercise 21.

23. [*21*] If we double all the requirements of exercise 22, we get a 60-car problem. Unfortunately that problem has no solution. Is there, however, a solution to the 61-car problem in which we manufacture one extra "Model 0" car (with *no* optional features)?

24. [*M25*] Inspired by the car sequencing problem, let's say that a "(p/q)-string," where $1 \le p < q$, is a binary string in which no q (or fewer) consecutive bits contain more than p 1s.

a) How many strings of length 10 are (1/2)-strings? (1/3)-strings? (2/3)-strings?
b) What is the maximum number of 1s in a (p/q)-string of length n?
c) Find the generating functions $G_{pq}(z) = \sum_{n\ge 0} C_{pqn} z^n$ for $1 \le p < q \le 5$, where C_{pqn} is the number of (p/q)-strings of length n.

▸ **25.** [*M35*] A (p/q)-string with the maximum number of 1s is called *extreme*.

a) Let $e_{pq}(m)$ be the number of (p/q)-strings of length qm that contain exactly pm 1s. Prove that $e_{pq}(m)$ is the number of plane partitions that fit in a $p \times (q-p) \times m$ box (see answer 7.2.2.1–262). *Hint:* Find a one-to-one correspondence.
b) Let c_{pqn} be the number of extreme (p/q)-sequences of length n. Express c_{pqn} in terms of the numbers in part (a).

26. [*M21*] (L. Szilassi, 1986.) Regard each of the following 14 triples ijk of digits

$$023, 134, 245, 356, 460, 501, 612,\ \ 054, 165, 206, 310, 421, 532, 643$$

as a cycle that contains the pairs ij, jk, and ki. Then every pair of distinct digits $i \ne j$ with $0 \le i, j < 7$ occurs exactly once. Show that those triples can be assigned to points (x, y, z) in such a way that every triple containing digit j belongs to plane j, where plane 0 is '$z = 0$'; plane 1 is '$4y + z = 200$'; plane 2 is '$2x + z = -280$'; plane 3 is '$5x - 5y + 7z = -700$'; plane 4 is '$-5x + 5y + 7z = -700$'; plane 5 is '$-2x + z = -280$'; plane 6 is '$-4y + z = 200$'. Furthermore, the six triples containing j form the boundary of a polygon that defines the face of a polyhedron, for $0 \le j < 7$.

▸ **27.** [*M28*] Three-dimensional space can be discretized into little "cubies," where cubie (i,j,k) consists of all points (x,y,z) with $i \le x \le i+1$, $j \le y \le j+1$, and $k \le z \le k+1$. (Each cubie therefore shares a common face with 6 adjacent cubies, a common edge with 12 diagonally adjacent cubies, and a common vertex with 8 corner-adjacent cubies.)

Given an $m \times n$ matrix (a_{ij}) for $0 \le i < m$ and $0 \le j < n$, its *histoscape* is the set of cubies (i,j,k) for $0 \le k < a_{ij}$. (For example, Fig. 101(d) is the histoscape for $\binom{4\,3}{1\,2}$.)

How many 2×2 matrices with $0 \le a_{ij} < 10$ have a histoscape that's a 3VP?

▸ **28.** [*M27*] Continuing exercise 27, how many of the 10^{64} 8×8 matrices whose entries satisfy $0 \le a_{ij} < 10$ for $0 \le i,j < 8$ have a histoscape that's a 3VP? *Hint:* Formulate this question as a constraint satisfaction problem.

29. [*24*] Extend the algorithm of the previous exercise so that it will find the kth $m \times n$ histoscape whose entries satisfy $0 \le a_{ij} < t$, given k, m, n, and t, when those histoscapes are listed in some convenient order. Then, by choosing k at random, use your method to find a uniformly random solution to the 8×8 problem.

▸ **30.** [*M26*] Given an $m \times n$ matrix whose histoscape is a 3VP, what are its vertices, and what polygons define its faces? (Design an algorithm that answers these questions.)

▸ **31.** [*M21*] (*Whirlpool permutations.*) An $m \times n$ matrix has $(m-1)(n-1)$ submatrices of size 2×2 in adjacent rows and columns. An $m \times n$ "whirlpool permutation" is an $m \times n$ matrix containing mn distinct numbers, in which the relative order of the elements in each of those submatrices is a "vortex" — that is, it travels a cyclic path from smallest to largest, either clockwise or counterclockwise.

Thus there are eight 2×2 whirlpool permutations of $\{1,2,3,4\}$:

$$\begin{pmatrix}1&2\\4&3\end{pmatrix} \quad \begin{pmatrix}1&4\\2&3\end{pmatrix} \quad \begin{pmatrix}2&1\\3&4\end{pmatrix} \quad \begin{pmatrix}2&3\\1&4\end{pmatrix} \quad \begin{pmatrix}3&2\\4&1\end{pmatrix} \quad \begin{pmatrix}3&4\\2&1\end{pmatrix} \quad \begin{pmatrix}4&1\\3&2\end{pmatrix} \quad \begin{pmatrix}4&3\\1&2\end{pmatrix}.$$

$$\begin{pmatrix}16&3&2&13\\9&7&8&10\\5&6&12&11\\4&14&15&1\end{pmatrix}$$

a) The 4×4 matrix at the right is not quite a whirlpool permutation. Fix the problem by interchanging two rookwise adjacent elements.
b) Prove that if any two rookwise adjacent elements of a whirlpool permutation are interchanged, the result is *not* a whirlpool.
c) What is the lexicographically smallest $m \times n$ whirlpool permutation of $\{1,\ldots,mn\}$?
d) True or false: The histoscape of an $m \times n$ matrix with distinct elements is a 3VP if and only if that matrix is a whirlpool permutation. (See Fig. 101(d).)
e) If M exceeds the difference between the largest and smallest elements of a whirlpool permutation, and if x is any number, prove that the matrix obtained after replacing each element a_{ij} by $(a_{ij}+x) \bmod M$ is also a whirlpool permutation.

▸ **32.** [*M30*] How many 5×5 matrices are whirlpool permutations of $\{0,1,\ldots,24\}$? *Hint:* An algorithm similar to that of exercise 28 can be used to count them.

▸ **33.** [*HM35*] An *up-up-or-down-down permutation* of $2n-1$ elements is a permutation $a_1a_2\ldots a_{2n-1}$ for which $a_{2k-1} < a_{2k}$ if and only if $a_{2k} < a_{2k+1}$, for $0 < k < n$. Let U_n be the number of such permutations; for example, $(U_1,\ldots,U_5) = (1,2,14,204,5104)$.
a) Prove that $U_{n+1} = \sum_k \binom{2n}{2k} Q_k Q_{n-k}$, where $Q_k = (k=0?\ 1\!: kU_k)$.
b) Find the exponential generating function $U(z) = U_1 z/1! + U_2 z^3/3! + U_3 z^5/5! + \cdots$.
c) What is the asymptotic behavior of U_n, correct to relative error $(1+O(1/4^n))$?
d) The number of $2 \times n$ whirlpool permutations is $2nU_n$. Prove this by establishing a one-to-one correspondence between up-up-or-down-down permutations and $2 \times n$ whirlpool permutations of $\{0,\ldots,2n-1\}$ with first element 0.

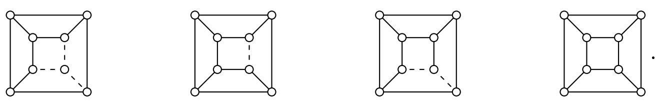
cubies
histoscape
constraint satisfaction problem
uniformly random solution
Whirlpool permutations
vortex
Dürer
lexicographically smallest
up-up-or-down-down permutation
exponential generating function
generating function
whirlpool permutations

34. [*21*] Which of the following partially filled 5×5 matrices can be completed to a whirlpool permutation of $\{1, 2, \ldots, 25\}$ in exactly one way?

$$\text{(i)}\ \begin{array}{|c|c|c|c|c|}\hline 1&3&5&7&9\\\hline 17&&&&\\\hline &&25&&\\\hline &&&&\\\hline 2&4&6&8&10\\\hline\end{array}\ ;\quad \text{(ii)}\ \begin{array}{|c|c|c|c|c|}\hline 3&14&15&9&2\\\hline &6&&5&\\\hline &&&&\\\hline &&&&\\\hline &&&&\\\hline\end{array}\ ;\quad \text{(iii)}\ \begin{array}{|c|c|c|c|c|}\hline 3&14&15&&\\\hline &9&2&6&\\\hline 5&&&&\\\hline &1&25&22&\\\hline &&11&21&19\\\hline\end{array}\ ;\quad \text{(iv)}\ \begin{array}{|c|c|c|c|c|}\hline 3&&14&&15\\\hline 9&&2&&6\\\hline &&5&&\\\hline 1&&21&&25\\\hline 4&&18&&22\\\hline\end{array}\ .$$

▸ **35.** [*M25*] The *skeleton* of a polyhedron is the graph formed by its vertices and edges. Hence the skeleton of a 3VP is a cubic graph. Make sketches of four 3VPs, each of which has the same skeleton as the 3-cube, but they differ in the number of concave edges.

36. [*M20*] The *signed skeleton* of a polyhedron is like its skeleton, but each edge is also identified as being either concave or convex. In illustrations we can indicate a convex edge by a solid line and a concave edge by a dashed line; for example, the signed skeletons of the objects in answer 35 are



What is the signed skeleton of the Szilassi polyhedron?

37. [*HM46*] Is there an algorithm to decide whether or not a given signed cubic graph can be realized as the signed skeleton of some 3VP?

38. [*HM20*] Let v_0 be a vertex of X, where X is a 3VP. Let the three neighbors of v_0 in the skeleton of X be $\{v_1, v_2, v_3\}$, and let each v_i have Cartesian coordinates (x_i, y_i, z_i).

a) Show that we can always choose the subscripts in such a way that

$$D(v_0, v_1, v_2, v_3) > 0, \qquad \text{where } D(v_0, v_1, v_2, v_3) = \det \begin{pmatrix} x_0 & y_0 & z_0 & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{pmatrix}.$$

b) Let p_{12} be the plane that contains v_0, v_1, and v_2. What equation defines the set of all vectors $v = (x, y, z)$ that lie in p_{12}?

c) What inequality characterizes all $v = (x, y, z)$ that lie on the same side of p_{12} as v_3?

d) Define p_{23} and p_{31} by analogy with p_{12}. Then the three planes p_{12}, p_{23}, p_{31} divide three-dimensional space into eight "octants": Every point v lies on one side or the other of each plane, unless it belongs to that plane. Devise a computer-friendly way to number the octants *0* to *7* in octal notation.

e) Using your numbering scheme, what octant contains the "three-famous-constants" point (π, ϕ, γ) when $v_0 = (0,0,0)$, $v_1 = (1,0,0)$, $v_2 = (0,1,0)$, $v_3 = (0,0,1)$?

f) Same as (e), but $v_0 = (0,0,0)$, $v_1 = (1,1,0)$, $v_2 = (0,1,1)$, $v_3 = (1,0,1)$.

▸ **39.** [*HM25*] Continuing exercise 38, let $\epsilon > 0$ be smaller than the distance from v_0 to any other vertex of X, and let X_ϵ be the interior of the closed set $X \cap S_\epsilon(v_0)$, where

$$S_\epsilon(v_0) = \{v \mid \|v - v_0\| \le \epsilon\} = \bigl\{(x, y, z) \mid (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 \le \epsilon^2\bigr\}.$$

a) Explain how to decide precisely which of the edges from v_0 to v_1, v_2, and v_3 are concave and which are convex, if told which of the octants are intersected by X_ϵ.

b) Explain how to compute the angles between the pairs of planes that meet at v_0.

pi as random
puzzle
skeleton
3-cube
concave edges
signed skeleton
convex edge
Szilassi polyhedron
realized
coordinates
octants
octal notation
pi as source
phi as source
gamma as source

40. [*HM25*] Using Cartesian coordinates (x, y, z), state quantitative conditions for the notion of "general position," under which we can be sure that a given 3VP X has a well-defined HC picture after projection to the (x, y)-plane.

41. [*M29*] Derive Table 1 by considering the $2^8 = 256$ different ways that up to eight cubies can be placed into a $2 \times 2 \times 2$ box.

a) Show that exactly 64 of those placements make a 3VP in which the center of the box is a vertex.

b) Furthermore, if that 3VP is in general position, we'll be able to *see* its central vertex in exactly 32 cases.

c) Draw those 32 pictures, and verify that the different possibilities for V, W, and Y junctions are precisely those shown in Table 1.

d) Also explain why Table 1 is correct for T junctions.

42. [*10*] If an HC network has respectively (t, v, w, y) junctions of types T, V, W, and Y, how many variables does the corresponding CSP have? How many constraints?

▸ **43.** [*18*] The line labeling problem has also been modeled as a CSP in quite a different way from (21) and (22): Instead of having one variable for each line, let there be one variable for each junction. The domain of variable j is then either $\{1, 2, 3, 4\}$ or $\{1, 2, 3, 4, 5, 6\}$ or $\{1, 2, 3\}$ or $\{1, 2, 3, 4, 5\}$, depending on whether j has type T, V, W, or Y; and j's value represents the index of the legal labeling in Table 1. There's one constraint for each line between junctions.

a) What is the constraint for line ab of (20) in this scheme?

b) How about the lines np and op?

c) What's the answer to exercise 42, with respect to *this* model?

d) Which model do you think is better?

44. [*15*] Translate the line labeling problem (22) into an XCC problem.

45. [*15*] What standard labeling of Szilassi's polyhedron differs from Fig. 104(b)?

46. [*M20*] If H is the HC network that corresponds to an HC picture, explain how to construct the HC network H^R that corresponds to the mirror image of that picture, when H and H^R both have the same junctions and the same oriented lines. Find a simple relation between the line labeling problems for H and H^R.

▸ **47.** [*M25*] An HC network is called *realizable* if it corresponds to at least one actual HC picture. Furthermore, that HC picture must not have a T junction whose collinear lines both lie on the outer boundary. (Such a T cannot be the image of a 3VP in general position. Notice that the line labeling problem for H is well defined regardless of whether or not H can be physically realized.)

a) What is the smallest unrealizable HC network? *Hint:* It has three junctions.

b) Characterize all realizable HC networks whose junctions all have type V.

c) Find an HC network, consisting entirely of type W junctions, that is unrealizable because it doesn't define a planar graph.

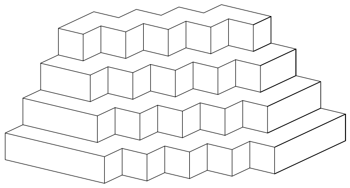
d) Prove that every realizable HC network contains at least three junctions of type V or W. *Hint:* Consider the boundary cycle of any connected component.

e) True or false: If the junction $T(\mathrm{a}, \mathrm{b}, \mathrm{c})$ in a realizable network is changed to either $W(\mathrm{c}, \mathrm{b}, \mathrm{a})$ or $Y(\mathrm{a}, \mathrm{b}, \mathrm{c})$, the resulting network is still realizable.

48. [*M46*] Is there an algorithm to decide whether a given HC network is realizable?

general position
HC picture
projection
HC network
XCC problem
Szilassi
mirror image
reflection of an HC network
realizable
planar graph
boundary cycle
connected component
decision problem

49. [*22*] Cover up the boundary of the HC picture



and watch the disconnected interior images as they jump in and out, before your eyes.

a) Show that this picture has only one standard labeling.

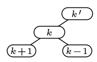
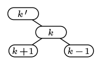
b) In how many ways can the boundary junctions be labeled consistently, without regard to any of the interior junctions?

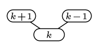
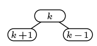
c) How many labelings are possible altogether, standard or not?

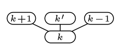
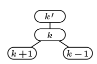
▸ **50.** [*M30*] Let $(j_0 j_1 \ldots j_{q-1})$ be the boundary cycle of a realizable HC network.

a) For $0 \le k < q$, show that there are only six possible ways to define j_k:

- $j_k = T(j_{k+1}, j_{k-1}, j'_k)$, called case $\boldsymbol{L}$;
- $j_k = T(j'_k, j_{k+1}, j_{k-1})$, called case $\boldsymbol{R}$;
- $j_k = V(j_{k+1}, j_{k-1})$, called case $\boldsymbol{V}$;
- $j_k = V(j_{k-1}, j_{k+1})$, called case $\boldsymbol{\Lambda}$;
- $j_k = W(j_{k+1}, j'_k, j_{k-1})$, called case $\boldsymbol{W}$;
- $j_k = Y(j_{k+1}, j'_k, j_{k-1})$, called case $\boldsymbol{Y}$.

$\boldsymbol{L} \leftrightarrow$  ; $\boldsymbol{R} \leftrightarrow$  ;

$\boldsymbol{V} \leftrightarrow$  ; $\boldsymbol{\Lambda} \leftrightarrow$  ;

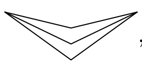
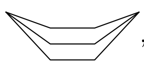
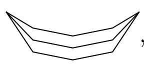
$\boldsymbol{W} \leftrightarrow$  ; $\boldsymbol{Y} \leftrightarrow$  .

(The subscripts in '$j_{k\pm 1}$' are to be understood mod q. The line j_k — j'_k in cases $\boldsymbol{L}$, $\boldsymbol{R}$, $\boldsymbol{W}$, and $\boldsymbol{Y}$ is called an "inner line," although j'_k might lie on the boundary.)

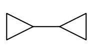
b) What combinations of line labels for $j_{k-1}j_k$, $j_k j_{k+1}$, $j_k j'_k$ can occur in each case?

c) Design an efficient way to test whether any inner line label can be assigned more than one value, when only the q constraints of the boundary cycle are imposed.

51. [*M23*] The "smile of order n" is a realizable HC network S_n with $3n+2$ junctions:

$S_1 =$  , $S_2 =$  , $S_3 =$  , $\ldots$.

How many line labelings does S_n have? How many of them are standard?

52. [*16*] In how many ways can the "bow tie"  be labeled?

53. [*M22*] Does a biconnected realizable HC network have a unique boundary cycle?

54. [*22*] Construct a realizable HC network that has a unique line labeling, although it doesn't have a *standard* labeling.

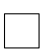
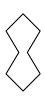
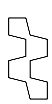
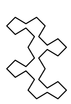
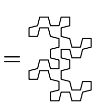
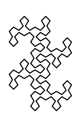
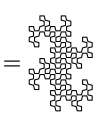
55. [*HM39*] Suppose each junction j_k of a boundary cycle $(j_0 j_1 \ldots j_{q-1})$ is $\boldsymbol{V}$ or $\boldsymbol{\Lambda}$.

a) Let $M_k = A$ if $j_k = \boldsymbol{V}$ and $M_k = B$ if $j_k = \boldsymbol{\Lambda}$, where $A = \left(\begin{smallmatrix}1&1\\1&0\end{smallmatrix}\right)$ and $B = \left(\begin{smallmatrix}1&-1\\-1&0\end{smallmatrix}\right)$ are 2×2 matrices. Prove that the number of ways to label the boundary cycle $(j_0 j_1 \ldots j_{q-1})$ is $\operatorname{trace}(M_0 M_1 \ldots M_{q-1}) + L_q$, where L_q is a Lucas number.

b) Show that $2F_q \le \operatorname{trace}(A^p B^{q-p}) + L_q \le 2L_q$ for $0 \le p \le q$. What p gives equality?

c) In fact, the number of labelings is between $2F_q$ and $2L_q$ in all cases.

56. [*HM21*] The *twindragon fractal* (see Fig. 1 in Chapter 4) can be approximated by a sequence of polygonal paths T_n for $n \ge 2$, where T_n has 2^n junctions:

$T_2 =$ , $T_3 =$ , $T_4 =$ , $T_5 =$ , $T_6 =$ , $T_7 =$ , $T_8 =$ , $\ldots$.

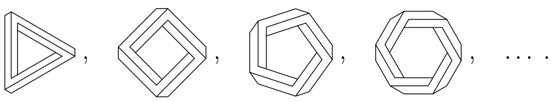
standard labeling
smile of order n
standard
bow tie
biconnected
standard
Lucas number
Fibonacci numbers
twindragon fractal
fractal

The clockwise path T_n turns left or right at step k and at step $k + 2^{n-1}$ according as the Jacobi symbol $\left(\frac{-1}{k}\right)$ is $= -1$ or $+1$, for $1 \le k \le 2^{n-1}$. (See exercise 4.5.4–23.)

In how many ways can T_n be labeled? *Hint:* Use exercise 55.

57. [*20*] Combine a V junction, a W junction, and a Y junction in such a way that the resulting subpicture cannot be labeled. (See (24) and (25).)

▸ **58.** [*M25*] The Penrose triangle, Penrose square, Penrose pentagon, Penrose hexagon, . . . , are

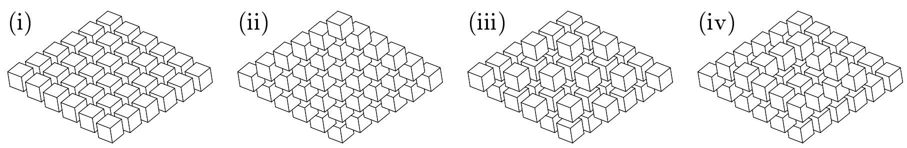


a) What is the HC network for the Penrose n-gon?
b) In how many ways can the Penrose n-gon be labeled consistently?
c) Is the Penrose n-gon weakly realizable for any $n \ge 3$?

59. [*20*] Explain how to obtain (32) as the projection of nine "squashed" cubes.

60. [*M22*] In how many ways can Reutersvärd's (32) be labeled (standard or not)?

61. [*24*] We can extend the idea in (32) to larger arrays of partially overlapping boxes:



(This is essentially a hexagonal grid, because each box can potentially overlap with six neighbors.) How many standard labelings are possible for (i), (ii), (iii), and (iv)?

62. [*23*] The 36 boxes in the 6×6 hexagonal arrays of exercise 61 involve 85 pairs (A, B) of adjacent boxes: $30 = 6 \cdot 5$ pairs in direction $\nearrow$; $30 = 5 \cdot 6$ pairs in direction $\nwarrow$; and $25 = 5 \cdot 5$ pairs in direction $\uparrow$. In every case we're allowed to specify either $A < B$ or $A > B$, meaning that A lies behind or in front of B in the image. Example (iv) illustrates the fact that this relation need not be transitive.

Thus those 36 boxes might be depicted in 2^{85} different ways. However, it turns out that the boxes are too close together to allow all possibilities: When boxes A, B, and C are mutually adjacent, we cannot simultaneously specify $A < B$, $B < C$, and $C < A$.

a) In how many ways can those 85 relations be specified, without any such non-transitive triplets? *Hint:* This is a CSP.
b) Generalize to $m \times n$ hexagonal arrays of boxes, for $1 \le m \le n \le 10$.

▸ **63.** [*M30*] Let H be a labeled HC picture, whose junctions have known (x, y) coordinates. Explain how to construct a system of linear equations and linear inequalities that have a solution whenever H is the projection of some 3VP X in general position.

64. [*22*] Is the following HC picture impossible? (It uses the right half of (26), twice.)

65. [*M25*] (L. Kirousis and C. Papadimitriou, 1988.) Prove that it's NP-complete to decide whether or not a realizable HC picture can be labeled.

66. [*HM46*] Is it decidable whether or not a given HC network is weakly realizable?

Jacobi symbol
Penrose n-gon
squashed
Reutersvärd
hexagonal grid
linear equations and linear inequalities
impossible
Kirousis
Papadimitriou
NP-complete
decision problem

67. [*15*] If we change `6` to `7` in Fig. 105(b), we get *another* graceful labeling, since the edge labels $10-6=4$ and $6-3=3$ become $10-7=3$ and $7-3=4$. Show that further graceful labelings can be obtained by changing only the labels of vertices `13` and `14`.

68. [*M21*] True or false: If graph G has k automorphisms, every graceful labeling of G is equivalent to $2k-1$ others, under symmetry and complementation.

▸ **69.** [*21*] To model the graceful labeling problem of Fig. 105 as an XCC problem, we can introduce 18 primary items $\{1,\ldots,18\}$ for the edge labels, 18 primary items $\{\texttt{NH-MA},\ldots,\texttt{GA-SC}\}$ for the edges, 13 secondary items $\{\texttt{NH},\ldots,\texttt{SC}\}$ for the colonies, and 19 secondary items $\{h_0,\ldots,h_{18}\}$ for the holders of vertex labels. These items are to be governed by $18\cdot 19\cdot 18 = 6156$ options, such as

$$\text{‘6 }\texttt{PA-DE PA}{:}3\ \texttt{DE}{:}9\ h_3{:}\texttt{PA}\ h_9{:}\texttt{DE}\text{’},$$

namely one for each edge label d, each edge, and each way to assign labels j and k with $0\le j<k=j+d\le 18$ to the endpoints of that edge. (The example shown covers edge label 6 and edge `PA-DE` when `PA` is labeled 3 and `DE` is labeled 9.) Given those options, Algorithm 7.2.2.1C needs about 90 gigamems to find the 641952 solutions.

a) Modify the model so that only the 160488 essentially different solutions are found.
b) Modify the model so that it solves the puzzle of Fig. 105(d).

70. [*M21*] The arrays `LO`, `FIRST`, `NEXTL`, `NEXTH`, `NAME` in (35) correspond to the labeling in Fig. 105(b). What arrays `LO`$'$, …, `NAME`$'$ correspond to its complement, Fig. 105(c)?

71. [*M20*] (S. Golomb, 1972.) Complete the proof that K_n is ungraceful when $n\ge 5$.

▸ **72.** [*25*] Design a backtrack algorithm to find all the graceful labelings of P_n as in (38).

73. [*26*] The search tree for graceful labelings of P_{10}, analogous to (38), contains 206 nodes, two of which are labeled 1738092 and 1809372. Those two nodes have *identical* subtrees, because they both represent a partial path between 1 and 2 that lacks the elements $\{4,5,6\}$. Modify the algorithm of exercise 72 so that it avoids such redundant computations, by identifying nodes that are obviously equivalent. (Think of ZDDs.)

74. [*M25*] (M. Adamaszek, 2013.) Consider n points that all lie on a straight line L.

a) What's the length of the longest path within L that doesn't hit any point twice?
b) Prove that if $p_1\ldots p_{2m}$ is a graceful permutation of $\{1,\ldots,2m\}$ with $p_{2m}=p_1+m$, then $p_{2k}>m$ for $1\le k\le m$.
c) Conversely, if $p_1\ldots p_{2m}$ is graceful and $p_{2k}>m$ for $1\le k\le m$, then $p_{2m}=p_1+m$.

▸ **75.** [*M30*] Determine all of the essentially different graceful labelings of $K_{1,1,n}$.

76. [*M16*] Prove that exactly one of the $4n!$ equivalent matrices (x_{ij}) that gracefully label a KP graph $K_n\,\square\,P_r$ has $0 \mathbin{—} (m-1)$ and satisfies (40).

77. [*16*] Study Fig. 107. Why doesn't $\boxed{\begin{smallmatrix}29\\80\end{smallmatrix}}$ appear in level 3 of that tree?

▸ **78.** [*21*] If $n>5$, one of the branches in the search tree analogous to Fig. 107 will set $x_{12}=m$ and $x_{22}=0$ at level 1, $x_{32}=m-1$ at level 2, $x_{42}=2$ at level 4 (and level 3), and $x_{52}=m-4$ at level 5. What are the immediate descendants of that level-5 node, if (a) $r=2$? (b) $r=3$?

79. [*M25*] Explain why the exhaustive search for graceful labelings of $K_n\,\square\,P_2$, illustrated for $n=3$ in Fig. 107, performs essentially identical calculations for all sufficiently large values of n, never finding a solution.

▸ **80.** [*20*] Draw levels 0, 1, and 2 of the search tree for $K_3\,\square\,P_3$, analogous to Fig. 107.

graceful labeling
automorphisms
symmetry
complementation
XCC problem
symmetry breaking
puzzle
complement
Golomb
K_n
ZDDs
Adamaszek
longest path
graceful permutation
KP

81. [*46*] Determine the number of graceful labelings of $K_n \,\square\, P_4$ for all n.

▸ **82.** [*20*] Is it possible to prove that $K_3 \square P_{17}$ is graceful by constructing a 3×17 matrix whose first row contains the first 34 digits of π?

83. [*M24*] Prove that $K_3 \,\square\, P_r$ is graceful for all $r \geq 1$, by constructing an appropriate $3 \times r$ matrix whose top row is $(0, m-2, 4, m-6, 8, \ldots)$.

84. [*46*] Is $K_4 \,\square\, P_r$ graceful for all $r \geq 1$?

85. [*M11*] How many symmetries does a KC graph have?

86. [*M18*] For what $n > 2$ and $r > 2$ does Lemma O prove that $K_n \,\square\, C_r$ isn't graceful?

▸ **87.** [*20*] Does Lemma O tell us anything useful about KP graphs?

88. [*20*] A graceful square: Show that $K_4 \,\square\, K_4$ is graceful(!).

89. [*12*] Is every graph with four edges graceful?

▸ **90.** [*M24*] A "random graceful graph" G_m^π can be based on π using the factorial series

$$\pi = 3 + \sum_{k=1}^{\infty} \frac{a_k}{(k+1)!}, \quad \text{where } 0 \leq a_k \leq k.$$

The vertices are $\{0, \ldots, m\}$; the edges are $0 — m$ and $a_k — a_k + m - k$, for $1 \leq k < m$.

a) Show that these integers a_k are unique, and compute them for $k \leq 20$.
b) How many isolated vertices does G_m^π have, for $m \leq 20$? How many components?
c) Determine the chromatic numbers $\chi(G_1^\pi)$, ..., $\chi(G_{20}^\pi)$.

91. [*22*] Among the 16! graceful labelings with 16 edges, how many of them define an n-vertex graph, for each n, after removing isolated vertices? How many are connected?

92. [*22*] Repeat exercise 91, but restrict the counts to *bipartite* graphs.

▸ **93.** [*22*] Explain how to compute all possible graceful labelings of r-regular graphs with m edges, given m and r. What are the smallest such labelings when $2 \leq r \leq 8$?

94. [*22*] Continuing exercise 93, make a complete survey of all graceful labelings of 2-regular graphs with ≤ 16 edges. How many such graphs are graceful?

95. [*32*] Continuing exercise 93, make a complete survey of all graceful labelings of 3-regular graphs (cubic graphs) with ≤ 14 *vertices*. How many such graphs are graceful?

96. [*46*] Is every connected cubic graph graceful?

▸ **97.** [*40*] Fun fact: Exactly 12345 different graphs have at most 8 nonisolated vertices. Study their gracefulness: How many of them are graceful? Which of them are *uniquely* graceful? Which of them are *maximally* graceful — graceful in the most different ways?

▸ **98.** [*28*] A graceful labeling is called *rooted* if every edge has a vertex in common with a longer edge, except edge m itself. For example, the first three graceful permutations in (38) are rooted; but the other three are not, because edge $1 — 3$ doesn't touch any of the longer edges $2 — 5$, $0 — 4$, $0 — 5$.

a) Is the 13-colonies labeling in Fig. 105(b) rooted?
b) How many of the 160488 graceful labelings of that graph are rooted?
c) How many of the 16! labelings in exercise 91 are rooted?
d) Compute the number of rooted graceful labelings of P_n, for $n \leq 16$.

99. [*30*] Find a connected graceful graph that has no *rooted* graceful labeling.

▸ **100.** [*35*] Design an algorithm that finds all of the ways to label a given graph gracefully. Try to choose data structures that are as efficient as possible.

101. [*29*] (S. Golomb, 1972.) In how many essentially different ways can the vertices and edges of (a) an icosahedron or (b) a dodecahedron be labeled gracefully?

π
KC graph
random graceful graph
G_m^π
π
factorial series
isolated vertices
components
chromatic numbers
isolated vertices
bipartite
r-regular graphs
2-regular graphs
3-regular graphs
cubic graphs
trivalent graphs
Fun fact
graphs, small
uniquely graceful
rooted
graceful permutations
13-colonies
P_n
data structures
efficient
Golomb
icosahedron
dodecahedron

▶ **102.** [*28*] Design a randomized algorithm that's able (with a little bit of luck) to discover "miraculous" graceful labelings of a largish graph, such as the one in Fig. 106.

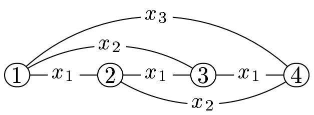
103. [*24*] Find all of the essentially distinct graceful labelings of (41).

104. [*M30*] (D. Anick, 2016.) To backtrack through all graceful labelings of free trees on the vertices $\{0,\ldots,m\}$, we can successively choose `LO[`k`]` for $k=m,\ m-1,\ \ldots,\ 1$, in such a way that the edge `LO[`k`]` —— `LO[`k`]` $+\,k$ doesn't produce a cycle in the graph-so-far. We shall prove that the number of choices is superexponential, by showing that there always are at least t_k choices for `LO[`k`]`, where t_k is suitably large.

At the moment we choose `LO[`k`]`, the current graph has exactly $k+1$ connected components (possibly singletons). Let's write $x \asymp y$ if vertices x and y belong to the same component; also $x \equiv y$ if $x \bmod k = y \bmod k$. Call r a "residue" if $0 \le r < k$, and call it "bad" if $x \equiv y \equiv r$ implies $x \asymp y$. Say also that x is bad if $x \bmod k$ is bad; a component is bad if all its vertices are bad. Furthermore, "good" means "not bad."

a) Show that there's always at least one good residue.
b) If there are g good residues, then $t_k \ge g$.
c) If there are G good components, then $t_k \ge G - g$.
d) If $k < m/2$, a bad component contains at least two different bad residues.
e) Hence we may let $t_k = \lfloor (k+4)/3 \rfloor$ when $k < m/2$.
f) When $k \ge m/2$ we may let $t_k = 2 + \lfloor (m-k)/2 \rfloor$. *Hint:* Prove that if $x \asymp x+k$, there are vertices $y < x$ and $z > x+k$ such that $y \asymp x$ and $z \asymp x+k$.

▶ **105.** [*HM25*] (N. Elkies, 2002.) In the complete graph on vertices $\{1,\ldots,n\}$, assign the weight x_d to edge k —— $(k{+}d)$, for $1 \le k \le n-d$ and $1 \le d < n$, as illustrated here for $n=4$. This graph has n^{n-2} spanning trees in general, by exercise 2.3.4.4–22; and we can form the sum $S(x_1,\ldots,x_{n-1})$ of the products of all edge weights, over each of those trees. For example, when $n=4$ we have



$$S(x_1,x_2,x_3) \;=\; x_1^3 + 4x_1^2x_2 + 3x_1x_2^2 + 3x_1^2x_3 + 4x_1x_2x_3 + x_2^2x_3,$$

because there's one spanning tree that uses all three x_1's, and four that use two x_1's and an x_2, etc. Notice that $[x_1x_2x_3]\,S(x_1,x_2,x_3) = 4$ is twice the total number of graceful labelings of 4-vertex trees, since a labeling and its complement are both counted.

a) Express $S(x_1,\ldots,x_{n-1})$ as a determinant. *Hint:* See exercise 2.3.4.2–20.
b) Explain how to compute $\tau(n{-}1) = [x_1 \ldots x_{n-1}]\,S(x_1,\ldots,x_{n-1})$ in $O(2^n n^3)$ steps.

106. [*HM46*] Determine the asymptotic value of the function $\tau(n)$ in exercise 105.

▶ **107.** [*21*] The *binomial tree* T_n has 2^n nodes $\{0,1,\ldots,2^n-1\}$, rooted at 0, where the parent of node $x \ne 0$ is node $x \,\&\, (x-1)$. (See 7.2.1.3–(21).) If $x = (x_{n-1}\ldots x_1x_0)_2$, let $l(x) = (l_{n-1}\ldots l_1l_0)_2$, where $l_k = x_0 \oplus \cdots \oplus x_k$. Show that these labels make T_n graceful.

108. [*24*] Continuing exercise 107, determine the exact number of essentially different graceful labelings of T_3 and T_4. Also estimate that number for T_5 and T_6.

109. [*M23*] Prove that the n-cube is graceful by means of the following labeling based on Gray code and an auxiliary sequence $0 = a_0 < a_1 < a_2 < \cdots$: Let $g(2k)$ and $g(2k+1)$ be labeled a_k and $m-k-a_k$, respectively, where $m = n2^{n-1}$. For example,

$$\begin{array}{rcllllllll} v &=& 000 & 001 & 011 & 010 & 110 & 111 & 101 & 100 \\ l(v) &=& a_0 & 12{-}a_0 & a_1 & 11{-}a_1 & a_2 & 10{-}a_2 & a_3 & 9{-}a_3 \end{array}$$

when $n=3$. (See 7.2.1.1–(4).) Assume that $a_{2^n+r} = a_{2^n} + a_r$ for $0 \le r < 2^n$.

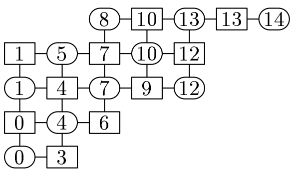
randomized algorithm
miraculous
Anick
free trees
superexponential
components
Elkies
complete graph
spanning trees
complement
determinant
binomial tree
n-cube
Gray code
binary recurrence

a) Let V_j be the vertices of the form $j\alpha$, and let L_j be the labels of the edges in $G\,|\,V_j$, for $0 \le j \le 1$. (For example, when $n = 3$ we have $V_0 = \{000, 001, 010, 011\}$ and $L_0 = \{12 - 2a_0, 12 - a_0 - a_1, 11 - 2a_1, 11 - a_1 - a_0\}$.) Express L_1 in terms of L_0.

b) What values of a_1, a_2, a_4, a_8, $\ldots$ make the labeling graceful?

▸ **110.** [*M25*] A *parallomino graph* (see exercise 7.2.2.1–303) has vertices (x, y) for integers $0 \le x \le r$ and $s_x \le y \le t_x$, where $0 = s_0 \le s_1 \le \cdots \le s_r$, $t_0 \le t_1 \le \cdots \le t_r$, and $s_{k+1} \le t_k$ for $0 \le k < r$; edges go from (x, y) to $(x+1, y)$ and $(x, y+1)$ when possible.

For example, the parallomino graph with $r = 6$, $(s_0, t_0) = (s_1, t_1) = (0, 3)$, $(s_2, t_2) = (1, 4)$, $(s_3, t_3) = (s_4, t_4) = (2, 4)$, and $(s_5, t_5) = (s_6, t_6) = (4, 4)$ can be decorated with labels in two closely related ways:

(i) 

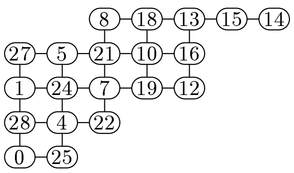
(ii) 

Illustration (ii) is in fact a remarkable *graceful* labeling, where the edges whose labels are 1, 2, $\ldots$, 28 appear in strict order, from right to left and top to bottom!

a) How many vertices and edges does a parallomino graph have, in general?

b) Decipher the rule that connects illustration (i) with illustration (ii).

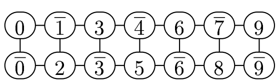
c) Reverse-engineer the rule by which illustration (i) was labeled.

d) Can *every* parallomino be gracefully labeled, using these rules?

▸ **111.** [*M25*] Let $\overline{l}$ denote $m - l$. A graph is *α-graceful* if its edges can be written

$$u_0 \,\frac{\quad}{}\, \overline{v_0},\ u_1 \,\frac{\quad}{}\, \overline{v_1},\ \ldots,\ u_{m-1} \,\frac{\quad}{}\, \overline{v_{m-1}},$$
$$\text{where } u_k + v_k = k,\ 0 \le u_k < l, \text{ and } 0 \le v_k < m{+}1{-}l, \text{ for some } l.$$

Here u_k and $\overline{v_k}$ are labels of vertices in the graph. For example, the labels



show that $K_2 \square P_7$ is α-graceful; and a similar construction works for $K_2 \square P_r$ in general.

a) Prove that an α-graceful graph is graceful and bipartite.

b) For which n is the cycle C_n α-graceful?

c) Prove that every α-graceful labeling has an "edge complement" in which edge k becomes edge $m + 1 - k$, for $1 \le k \le m$.

d) Find a tree with seven nodes that's not α-graceful.

112. [*23*] A bipartite graph with parts U and V has an *ordered* graceful labeling if it has a graceful labeling such that $l(u) < l(v)$ for every edge $u \,\frac{\quad}{}\, v$ with $u \in U$, $v \in V$.

a) Show that every α-graceful graph has an ordered graceful labeling.

b) Show that the non-α-graceful tree of answer 111(d) *also* has such a labeling.

c) Let G have m edges and an ordered graceful labeling. Prove that m copies of G can be perfectly packed into the complete bipartite graph $K_{m,m}$.

d) A bipartite graph G with m edges $u \,\frac{\quad}{}\, v$ between parts U and V leads naturally to a bipartite graph $G^{(t)}$ with tm edges $u \,\frac{\quad}{}\, v_i$ between parts U and $V_1 \cup \cdots \cup V_t$. If G has an ordered graceful labeling, show that $G^{(t)}$ does too.

113. [*M21*] Continuing exercises 111 and 112, how many (a) α-graceful labelings (b) ordered graceful labelings have m edges? (Compare with Theorem S.)

parallomino graph
grid
skeleton
α-graceful
bipartite
cycle C_n
complement
ordered graceful labeling
near α-labeling, see ordered graceful labeling

114. [*M20*] The direct product of bipartite graphs always has at least two components.

a) Prove this, by determining the components of $K_{a,b} \otimes K_{c,d}$.

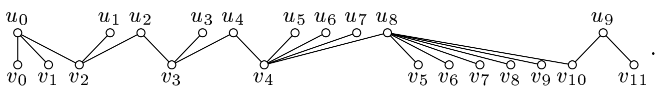
b) When G and H are bipartite with parts (U,V) and (X,Y), let $(G\otimes H)'$ and $(G\otimes H)''$ be $G\otimes H$ restricted respectively to parts $(U\times X, V\times Y)$ and $(U\times Y, V\times X)$. Describe (i) $(P_{2m}\otimes P_{2n})'$ and $(P_{2m}\otimes P_{2n})''$; (ii) $(P_{2m}\otimes P_{2n+1})'$ and $(P_{2m}\otimes P_{2n+1})''$; (iii) $(P_{2m+1}\otimes P_{2n+1})'$ and $(P_{2m+1}\otimes P_{2n+1})''$; (iv) $(C_{2m}\otimes C_{2n})'$ and $(C_{2m}\otimes C_{2n})''$; (v) $(Q_m\otimes Q_n)'$ and $(Q_m\otimes Q_n)''$, where Q_n is the n-cube.

c) If G and H each have an ordered graceful labeling, prove that $(G\otimes H)'$ and $(G\otimes H)''$ do too.

▸ **115.** [*M28*] (*Caterpillar nets.*) A "caterpillar" is a graph with at least two vertices that becomes a path (or empty) when you remove all of its vertices of degree 1. More precisely, an (s,t)-*caterpillar* is a bipartite graph with vertices $\{u_0,\ldots,u_s; v_0,\ldots,v_t\}$ and edges defined by a binary vector $e = e_1\ldots e_{s+t}$ that has s 0s and t 1s:

$$u_{s_i} \;\text{—}\; v_{t_i} \quad \text{for } 0\le i\le s+t, \quad \text{where } s_i = \bar e_1+\cdots+\bar e_i \text{ and } t_i = e_1+\cdots+e_i.$$

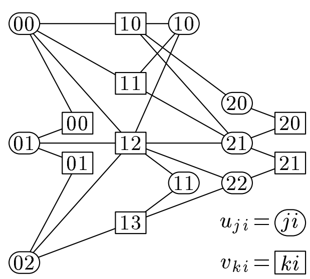
For example, here's the $(9,11)$-caterpillar whose edge vector is 11001001000011111101:



a) Draw the eight (s,t)-caterpillars for which $s+t=3$.

b) Prove that every (s,t)-caterpillar is α-graceful.

c) Given an (s,t)-caterpillar, a "caterpillar net" is a graph obtained when we replace the vertices u_j and v_k by disjoint sets of vertices $U_j = \{u_{j0},\ldots,u_{jp_j}\}$ and $V_k = \{v_{k0},\ldots,v_{kq_k}\}$, for $0\le j\le s$ and $0\le k\le t$. The edges are (p_{s_i},q_{t_i})-caterpillars between U_{s_i} and V_{t_i}, for $0\le i\le s+t$. For example, a caterpillar net with $e=1001$, $p_0=p_2=2$, $p_1=q_0=q_2=1$, and $q_1=3$ is illustrated here. How many edges does a caterpillar net have?

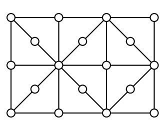
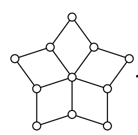


d) Prove that every caterpillar net is α-graceful.

e) Prove that the complete bipartite graph $K_{n,r}$ is a caterpillar net.

f) Prove that the grid $P_n \,\square\, P_r$ is a caterpillar net.

g) Are either of the following graphs caterpillar nets?


;

.

116. [*23*] The grid graph $P_2\,\square\,P_6$ is the "skeleton" of a pentomino (showing the outlines of its five cells). Prove that the skeletons of all twelve pentominoes are α-graceful.

117. [*HM36*] Exercise 115(e) proved that $K_{n,r}$ is α-graceful. Let $A(n,r)$ be the exact number of different α-labelings that $K_{n,r}$ has, times 2 if $n=r>1$. (We know that $K_{2,2}=C_4$ has a unique graceful labeling; but $A(2,2)=2$ because the edges can be written either as $0 \text{—} \bar0$, $1 \text{—} \bar0$, $0 \text{—} \bar2$, $1 \text{—} \bar2$ or $0 \text{—} \bar0$, $0 \text{—} \bar1$, $2 \text{—} \bar0$, $2 \text{—} \bar1$ in the notation of exercise 111.)

a) Prove that $A(n,r)$ is the number of ways to write the polynomial $F_m(x) = 1 + x + \cdots + x^{m-1}$ as a product $G(x)H(x)$, where $m=nr$, $G(1)=n$, $H(1)=r$, and all coefficients of G and H are either 0 or 1. (For example, $A(2,2)=2$ because $F_4(x) = (1+x)(1+x^2) = (1+x^2)(1+x)$; $A(6,2)=4$ because $F_{12}(x) =$

direct product
tensor product, see direct product
n-cube
grid
torus
Path P_n
cycle C_n
ordered graceful labeling
Caterpillar nets
(s,t)-caterpillar
bipartite graph
pi, as random example
complete bipartite graph
$K_{n,r}$
grid
grid graph
skeleton
pentominoes
$K_{n,r}$
complete bigraph
polynomial

$(1+x+x^2+x^3+x^4+x^5)(1+x^6) = (1+x+x^2+x^6+x^7+x^8)(1+x^3) = (1+x+x^4+x^5+x^8+x^9)(1+x^2) = (1+x^2+x^4+x^6+x^8+x^{10})(1+x)$.)

b) Prove that if $F_m(x) = G(x)H(x)$ and the coefficients of G and H are real, both G and H are *palindromials* (palindromic polynomials): Their coefficients are the same when read in either direction. (That is, $G(x) = x^{\deg(G)}G(1/x)$.)

c) Furthermore if all coefficients of G and H are between 0 and 1, they're all 0 or 1.

d) Furthermore, if $n > 1$ and $r > 1$, either $G(x)$ or $H(x)$ has the special form $F_k(x)T(x)$, where $1 < k < m$ and all coefficients of T are 0 or 1.

e) Furthermore, $G(x) = F_k(x)T(x)$ implies that H and T are polynomials in x^k.

f) Conclude that $A(p,q) = 2$ whenever p and q are prime. What is $A(p^e, q^f)$?

g) What is $A(p_1p_2, q_1q_2)$, when p_1, p_2, q_1, q_2 are prime and $p_1 \neq p_2$, $q_1 \neq q_2$?

h) Use trace theory (Theorem 7.2.2.2F) to prove that $A(p_1^{e_1}\ldots p_s^{e_s}, q_1^{f_1}\ldots q_t^{f_t}) = [p_1^{e_1}\ldots p_s^{e_s}q_1^{f_1}\ldots q_t^{f_t}]\,1/\big((1-p_1)\ldots(1-p_s)+(1-q_1)\ldots(1-q_t)-1\big)$.

i) In particular, $A(p^e, q_1^{f_1}\ldots q_t^{f_t}) = \binom{e+f_1}{e}\ldots\binom{e+f_t}{e}$.

118. [*M22*] Show that $K_{n,r}$ sometimes has graceful labelings that are *not* α-graceful:

a) If $r = 2$ and $2n+1 = pq$ with $p,q > 1$, use labels $\{2n, 2n-p\}$ in the second part, with labels $\bigcup_{k=0}^{\lfloor q/2\rfloor-1}[2kp \,.\,.\, 2kp+p)$ and $\lfloor p/2\rfloor$ others in the first part.

b) If $n = 3k+1$, use labels $[0\,.\,.\,2k] \cup [nr-k\,.\,.\,nr-1]$ in the first part.

119. [*M46*] Does $K_{n,r}$ have graceful labelings besides those of exercises 117 and 118?

120. [*20*] Given a simple digraph D without loops, construct an XCC problem whose solutions are the graceful labelings of D. *Hint:* Modify the construction in exercise 69.

121. [*13*] For what a and b does $x \mapsto (ax+b) \bmod 13$ take Fig. 109(e) into Fig. 109(f)?

122. [*22*] Find all of the essentially different graceful labelings of Fig. 109(a).

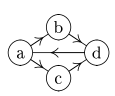
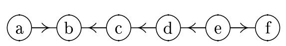
▶ **123.** [*M28*] Two graceful labelings l and l' of a digraph D with $q-1$ arcs are called *affinely equivalent* if $l'(v) = \big(a(l(v)-b)\big) \bmod q$ for all vertices v, where a and b are integers with $a \perp q$. (This notion matches transformations (i) and (ii) discussed in the text.)

a) Let v and w be distinct vertices of D. Show that every graceful labeling l is affinely equivalent to a graceful labeling l' for which $l'(v) = 0$ and $l'(w) = d$ for some $d\backslash q$.

b) Exactly how many such labelings l' exist, given d and q?

c) Now explain how to take the labelings found in (a) and find all of the "essentially different" ones, by taking account of D's symmetries and antisymmetries.

124. [*19*] What are the essentially different ways to label these digraphs gracefully?

(a)  ; (b)  .

125. [*16*] Design an algorithm to create the `FIRST` and `NEXT` arrays of a graceful digraph, given its `LO` array.

▶ **126.** [*M25*] Let l be a graceful labeling of D, and let `LO`, `FIRST`, `NEXT`, and `NAME` be the corresponding representation as in (44). A labeling l' equivalent to l will then correspond to certain arrays `LO`$'$, `FIRST`$'$, `NEXT`$'$, and `NAME`$'$. (Compare with exercise 70.)

a) Compute them when $l'(v) = \big(a(l(v)-b)\big) \bmod q$, given a and b with $a \perp q$.

b) Compute them when $l'(v) = l(v\alpha)$, given an automorphism α of D.

c) Compute them when $l'(v) = l(v\alpha)$, given an antiautomorphism α of D.

▶ **127.** [*M24*] The digraph D in Fig. 109 doesn't fully represent set inclusion in a 3-element universe because it isn't transitive. Let D^* be the digraph obtained when

real
palindromials
trace theory
digraph
XCC problem
essentially different graceful labelings
affinely equivalent
essentially different
symmetries
antisymmetries
`FIRST`
`NEXT`
`LO`
digraph representation
automorphism
antiautomorphism
set inclusion
transitive
Boolean lattice

the arcs $000 \longrightarrow 011$, $000 \longrightarrow 101$, $000 \longrightarrow 110$, $000 \longrightarrow 111$, $001 \longrightarrow 111$, $010 \longrightarrow 111$, $100 \longrightarrow 111$ are added to D. What are its classes of equivalent graceful labelings?

128. [*22*] When the digraph in Fig. 109 is extended to a 4-element universe, it has 16 vertices and 32 arcs. Is it still graceful?

▸ **129.** [*HM35*] Let $\mathcal{D}_m$ be the set of m-tuples $x = x_1 \ldots x_m$ with $0 \le x_l \le m$ for $1 \le l \le m$. If $x \in \mathcal{D}_m$, the digraph $aD(x)+b$ has $m+1$ vertices $\{0, \ldots, m\}$ and m arcs, $(ax_l + b) \bmod q \longrightarrow (ax_l + al + b) \bmod q$, where $q = m+1$. Furthermore, say that $x \equiv x'$ in $\mathcal{D}_m$ if $aD(x)+b$ equals $D(x')$ or its converse $D(x')^T$, for some a and b with $a \perp q$.

a) What are the equivalence classes of $\mathcal{D}_2$ and $\mathcal{D}_3$?
b) What's a good way to visit each equivalence class of $\mathcal{D}_m$, when m isn't too large?
c) What's a good way to count the number of equivalence classes, when m is larger?

130. [*25*] Let $l_k = l(v_k)$ be the kth vertex label in a path or cycle $v_0 \longrightarrow \cdots \longrightarrow v_m$.

a) Show that $l_{2k} = r-1-k$ and $l_{2k+1} = r+k$ gracefully label the oriented path $\vec{P_{2r}}$.
b) Find a somewhat similar pattern of graceful labels for $\vec{C_{2r}}$. *Hint:* Use vertex labels $< r$ and arc labels $\equiv r-1$ (modulo 2) in the first half of the cycle.

131. [*20*] (G. S. Bloom and D. F. Hsu.) If D is a graceful digraph with m arcs and $m+1$ vertices, prove that $D \longrightarrow \overline{K_n}$ is also graceful. (It has $mn+m+n$ arcs, $m+n+1$ vertices.)

132. [*22*] Find an ungraceful digraph D with 2 arcs and 3 vertices such that $D \longrightarrow \overline{K_n}$ is graceful for all $n > 0$.

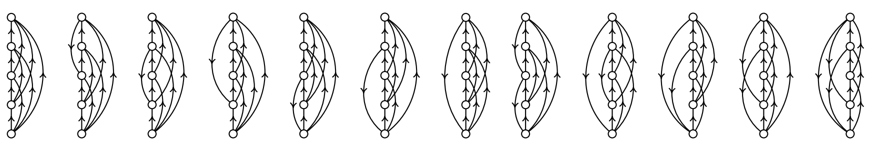
133. [*16*] Is the oriented complete bipartite graph $\vec{K_{m,n}} = \overline{K_m} \longrightarrow \overline{K_n}$ graceful?

134. [*41*] Investigate all of the graceful digraphs that have at most 6 nonisolated vertices. (Compare with exercise 97; the number rises from 12345 to 1540943.)

135. [*M20*] (C. Delorme.) Let D be a digraph with m arcs for which the total degree $d^+(v)+d^-(v)$ is *even* at every vertex v. Prove that D cannot be graceful if $m \bmod 4 = 1$.

136. [*20*] (G. S. Bloom and D. F. Hsu.) Show that the m edges of a digraceful graph can always be oriented in at least $2^{\lfloor m/2 \rfloor}$ graceful ways.

▸ **137.** [*M30*] A *tournament* is a digraph in which either $u \longrightarrow v$ or $v \longrightarrow u$ for every pair of vertices u and v (see exercise 7–59). There are twelve unlabeled tournaments of order 5:



A B C D E F G H I J K L

a) What are the *converses* of A, B, …, L? (For example, $A^T = A$.)
b) How many essentially distinct graceful labelings does each of them have?
c) What are the graceful tournaments of orders 3 and 4?
d) A *cyclic* (v, k, λ)-*difference set* is a set $\{a_1, \ldots, a_k\} \subseteq \{0, 1, \ldots, v-1\}$ such that the $k(k-1)$ differences $(a_j - a_k) \bmod v$ for $j \ne k$ contain each nonzero residue exactly λ times. For example, $\{0, 1, 3\}$ is a cyclic $(4, 3, 2)$-difference set because

$$0 \ominus 1 = 3,\ 0 \ominus 3 = 1,\ 1 \ominus 0 = 1,\ 1 \ominus 3 = 2,\ 3 \ominus 0 = 3,\ 3 \ominus 1 = 2,$$

writing '$x \ominus y$' for $(x-y) \bmod v$. Prove that there exists a graceful n-vertex tournament if and only if there exists a cyclic $(\binom{n}{2}+1, n, 2)$-difference set.
e) Show that $\{1, 7, 7^2 \bmod 37, \ldots, 7^8 \bmod 37\}$ is a cyclic $(37, 9, 2)$-difference set.

Boolean lattice
set inclusion
converse
affine equivalence
oriented path
Bloom
Hsu
oriented complete bipartite graph
graphs, small
digraphs, small
Delorme
parity
$d^+(v)$ (out-degree)
Bloom
Hsu
digraceful graph
tournament
converses
essentially distinct
cyclic (v, k, λ)-difference set

138. [*46*] An undirected graph with m edges can be converted to a directed graph in 3^m ways, because each edge $u \relbar\!\relbar v$ can become $u \longrightarrow v$ or $u \longleftarrow v$ or both. Is *every* graph "weakly digraceful," in the sense that at least one of those 3^m possibilities is graceful?

139. [*20*] (M. Buratti and A. Del Fra.) Show that (47) gracefully labels $C_n^{\leftrightarrow}$.

▶ **140.** [*23*] (R. Montgomery, A. Pokrovskiy, and B. Sudakov.) Prove that every tree T with m edges and a vertex v adjacent to at least $2m/3$ leaves is rainbow graceful.

141. [*24*] Find rainbow graceful labelings of (i) $\overline{K_1 \oplus 3K_2}$; (ii) $\overline{4K_1 \oplus C_3}$; (iii) $\overline{2C_4}$.

142. [*30*] Which of the 12345 graphs of exercise 97 are *rainbow* graceful?

143. [*23*] Is every digraceful graph also rainbow graceful?

144. [*HM20*] A *projective plane of order n* has n^2+n+1 points and n^2+n+1 lines, where every line contains exactly $n+1$ points and every point belongs to exactly $n+1$ lines. Furthermore, every two points belong to exactly one line, and every two lines intersect in exactly one point. The following construction defines such a plane whenever F is a finite field of n elements (see exercise 4.6.2–16): Each point is a nonzero triple (a_1, a_2, a_3), and each line is a nonzero triple $[b_1, b_2, b_3]$, where the a's and b's belong to F. Two triples are considered equal if one is a multiple of the other; for example, $(a_1, a_2, a_3) = (2a_1, 2a_2, 2a_3)$ in the field of three elements. Point (a_1, a_2, a_3) lies on line $[b_1, b_2, b_3]$ if and only if $a_1b_1 + a_2b_2 + a_3b_3 = 0$ in F.

a) Explain why this construction gives n^2+n+1 points and n^2+n+1 lines.
b) Which points belong to the line $[1, 0, 2]$ when $n = 3$?
c) Why do two lines intersect in a unique point?

▶ **145.** [*HM27*] (J. Singer, 1938.) Suppose K_{n+1} has graceful rainbow labels $\{l_0, \ldots, l_n\}$.

a) Show that they're a cyclic $(n^2+n+1, n+1, 1)$-difference set (see exercise 137(d)).
b) If $n = p$ is prime, let $f(x) = x^3 - c_1x^2 - c_2x - c_3$ be a primitive polynomial modulo p for the field F of p^3 elements (see 3.2.2–(9)). Consequently the nonzero elements of F are $\{1, \pi, \pi^2, \ldots, \pi^{p^3-2}\}$, where π is a root of f in F. What are the other two roots of f? *Hint:* $(x+y)^p \equiv x^p + y^p$ (modulo p).
c) Continuing (b), find a transformation $(a_1, a_2, a_3)\alpha = (a_1', a_2', a_3')$ of triples with the property that $\pi^k = a_1\pi^2 + a_2\pi + a_3$ implies $\pi^{k+1} = a_1'\pi^2 + a_2'\pi + a_3'$.
d) Find a transformation $[b_1, b_2, b_3]\alpha = [b_1', b_2', b_3']$ of triples, to go with the transformation in (c), with the property that $a_1b_1 + a_2b_2 + a_3b_3 = a_1'b_1' + a_2'b_2' + a_3'b_3'$.
e) As a consequence of (c), there are triples (a_{k1}, a_{k2}, a_{k3}) of integers mod p for which we have $\pi^k = a_{k1}\pi^2 + a_{k2}\pi + a_{k3}$, for $0 \le k < p^3 - 1$. List those triples in the special case when $p = 5$ and $f(x) = x^3 - 4x^2 - 3$. (You can stop at $k = 31$.)
f) Construct a projective plane of order p as in exercise 144, and show that we may take the points to be (a_{k1}, a_{k2}, a_{k3}) for $0 \le k < p^2 + p + 1$. Furthermore, $L = \{k \mid a_{k1} = 0 \text{ and } 0 \le k < p^2+p+1\}$ is a set of graceful rainbow labels for K_{p+1}.
g) Extend the ideas of (b)–(f) to the case when $n = p^e$ is an arbitrary power of the prime p, and work out the details when $n = 8$.

146. [*HM33*] Let $\mathcal{R}_m$ be the set of m-tuples $x = x_1 \ldots x_m$ with $0 \le x_l \le 2m$ for $1 \le l \le m$. If $x \in \mathcal{R}_m$, the graph $aG(x) + b$ has $2m+1$ vertices $\{0, \ldots, 2m\}$ and m edges, $(ax_l + b) \bmod q \relbar\!\relbar (ax_l + al + b) \bmod q$, where $q = 2m+1$. Furthermore, say that $x \equiv x'$ in $\mathcal{R}_m$ if $aG(x) + b$ equals $G(x')$, for some a and b with $a \perp q$.

a) What are the equivalence classes of $\mathcal{R}_2$ and $\mathcal{R}_3$? (Compare with exercise 129.)
b) What's a good way to visit each equivalence class of $\mathcal{R}_m$, when m isn't too large?
c) What's a good way to count the number of equivalence classes, when m is larger?

weakly digraceful
Buratti
Del Fra
Montgomery
Pokrovskiy
Sudakov
rainbow graceful labelings
projective plane
uniform hypergraph
finite field
Singer
primitive polynomial modulo p
projective plane
affine equivalence

147. [*46*] Is every *forest* rainbow graceful?

148. [*15*] True or false: A *subgraph* of H is any graph that we obtain from H by removing zero or more edges, then removing zero or more isolated vertices. An *induced subgraph* of H is any graph that we obtain from H by removing zero or more vertices, then removing every edge that touched at least one of those vertices.

149. [*16*] Find, by hand, an induced C_7 of common English words, including `chord`.

150. [*17*] Is `cords` — `colds` — `colts` — `costs` — `casts` — `carts` — `cards` — `cords` an isometric embedding of C_7 into `WORDS`(5757)?

151. [*M21*] A *Hamming graph* is a graph of the form $K_{n_1} \,\square\, K_{n_2} \,\square \cdots \square\, K_{n_r}$. Thus it has $n_1 \ldots n_r$ vertices $x_1x_2\ldots x_r$, where $0 \le x_k < n_k$ for $1 \le k \le r$; and we have $x_1x_2\ldots x_r \mathrel{—} y_1y_2\ldots y_r$ if and only if $x_k \ne y_k$ for exactly one index k.

a) How many edges does $K_{n_1} \,\square\, K_{n_2} \,\square \cdots \square\, K_{n_r}$ have?
b) How many automorphisms does $K_{n_1} \,\square\, K_{n_2} \,\square \cdots \square\, K_{n_r}$ have?
c) Compute the distance between 141421 and 271828 in a Hamming graph.
d) If a clique G is embedded in $K_{n_1} \square K_{n_2} \square \cdots \square K_{n_r}$, prove that its image is constant in all but one of the constituents K_{n_k}.
e) What 4-vertex graph G can't be strictly embedded in a Hamming graph?
f) Prove that the five-cycle C_5 can't be strictly embedded into a Hamming graph.

152. [*27*] Exactly how many induced seven-cycles are present in `WORDS`(5757)? How many of them are isometrically embedded?

153. [*22*] A strict embedding into a Hamming graph is called a *Hamming embedding.* More precisely, if G is a graph with vertices $\{v_0, v_1, \ldots, v_{n-1}\}$, a Hamming embedding of G is a function $f(v_i) = x_{i1}\ldots x_{ir}$ with the property that, for $0 \le i < j < n$, we have $x_{i1}\ldots x_{ir} \mathrel{—} x_{j1}\ldots x_{jr}$ in a Hamming graph if and only if $v_i \mathrel{—} v_j$ in G.

a) Assume that G is connected, and that each vertex v_i for $i > 0$ has a "parent vertex" $v_{i'}$ with $i' < i$ and $v_{i'} \mathrel{—} v_i$. Show that every Hamming embedding of G can be "normalized" so that (i) $x_{0k}x_{1k}\ldots x_{(n-1)k}$ is a restricted growth string, as defined in 7.2.1.5–(4), for $1 \le k \le r$; and (ii) $x_{i(k+1)} > 0$ for $i > 0$ implies that $x_{jk} > 0$ for some $j < i$. (Condition (ii) means that we don't "invade" coordinate $k+1$ until coordinate k has been used. In particular, a normalized embedding always has $x_{01}x_{02}\ldots x_{0r} = 00\ldots0$ and $x_{11}x_{12}\ldots x_{1r} = 10\ldots0$.)
b) Design an algorithm that visits every normalized Hamming embedding of G.

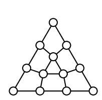
154. [*18*] A graph G is called *minimal non-Hamming* (MNH) when its induced subgraphs G' are Hamming embeddable if and only if $G' \ne G$.

a) Is G Hamming embeddable if and only if it has no induced MNH subgraph?
b) Prove that an MNH subgraph is connected.
c) True or false: If G is connected and not Hamming embeddable and not MNH, one of its subgraphs $G \setminus v$ is connected and not Hamming embeddable.

155. [*24*] Find all MNH graphs that have at most nine vertices.

156. [*25*] (P. M. Winkler, 1984.) If graph G satisfies the conditions of exercise 153(a), prove that it has at most one normalized *isometric* embedding into a Hamming graph. Also design a polynomial-time algorithm that discovers the embedding, if it exists.

157. [*M25*] (P. M. Winkler, 1984.) Let $(u \mathrel{—} v) \bowtie (u' \mathrel{—} v')$ be the relation $d(u,u') - d(u,v') \ne d(v,u') - d(v,v')$, when $u \mathrel{—} v$ and $u' \mathrel{—} v'$ are edges of a graph and $d(u,v)$ denotes shortest distance in that graph.



a) Determine the $\bowtie$ relation between the 18 edges of the graph shown.

subgraph
induced subgraph
isometric embedding
`WORDS`(5757)
Hamming graph
Cartesian product
automorphisms
isometrically embedded
strict embedding
Hamming embedding
parent vertex
restricted growth string
minimal non-Hamming
MNH
induced subgraphs
Winkler
isometric
Winkler
relation

b) True or false: In a complete graph, $(u \mathbin{-\!-} v) \bowtie (u' \mathbin{-\!-} v') \iff \{u,v\} \cap \{u',v'\} \neq \emptyset$.

c) A *ternary Hamming graph* is a graph of the form $K_3 \,\square \cdots \square\, K_3$, "a Cartesian product of triangles." If G can be isometrically embedded in a ternary Hamming graph, prove that the $\bowtie$ relation in G is *transitive* (so it's an equivalence relation).

d) Conversely, if $\bowtie$ is transitive in G, there's an isometric ternary embedding of G.

158. [*24*] Find the smallest graph that (i) can be embedded as an induced subgraph, but not isometrically; (ii) can be embedded isometrically, but has an induced subgraph that cannot. How many graphs of n vertices, for $1 \le n \le 9$, can be isometrically embedded in a Hamming graph? (See exercise 155.)

▶ **159.** [*M37*] (*Subcube labels.*) A string of 0s, 1s, and $*$s conventionally represents a subcube of a cube, where each $*$ is a "wild card" that stands for either 0 or 1. For example, $0{*}1{*}$ represents $\{0010, 0011, 0110, 0111\}$, which is a subcube of $*{*}{*}{*}$.

It's easy to work with subcubes inside a computer, using the asterisks-and-bits representation of exercise 7.1.1–30. For example, $0{*}1{*}$ is represented by the two bitstrings $a = 0101$ and $b = 0010$, showing respectively the $*$s and the 1s.

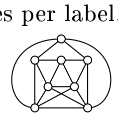
The vertices of a connected graph can always be labeled with subcubes in such a way that *the distance between any two vertices is exactly equal to the distance between their labels*(!). One such labeling of the five-cycle $0 \mathbin{-\!-} 1 \mathbin{-\!-} 2 \mathbin{-\!-} 3 \mathbin{-\!-} 4 \mathbin{-\!-} 0$ is

$$l(0) = 0000,\ l(1) = 1000,\ l(2) = 11{*}0,\ l(3) = {*}{*}11,\ l(4) = 0{*}01;$$

for example, the distance $d(1,4)$ from 1 to 4 is 2; so is the distance from 1000 to $0{*}01$.

a) Give a formula for the distance between subcubes represented by (a,b) and (a',b').

b) Find all of the subcube representations of C_5 that have 4 coordinates per label.

c) Show that the eight-vertex graph illustrated here has a subcube representation, with 4 coordinates per label, in which the vertices of the induced five-cycle have the same labels as shown above. 

d) Let T be a tree with n vertices, rooted at r. Assign labels with $n-1$ coordinates to each vertex v of T, with one coordinate v_w for each $w \neq r$, defined by the rule

$$v_w \;=\; [w \text{ is an inclusive ancestor of } v] \;=\; [d(v,w) + d(w,r) = d(v,r)].$$

Exactly $d(v,r)$ coordinates of $l(v)$ are 1. Show that these are valid subcube labels.

e) Given any graph G on n vertices, let T be a spanning tree rooted at r, with every vertex v at level $d(r,v)$ of that tree. Construct labels as in (d), with $v_w = 1$ if w is an inclusive ancestor of v; but otherwise $v_w = (0, ?, *)$ if $d(v,w) - d(v,w') = (1, 0, -1)$, respectively. Here '?' is a special value that contributes $\frac{1}{2}$ to the distance when matched with 1, but 0 when matched with 0 or $*$ or ?. For example, if $G = C_5$ and $r = 0$, and if T has all edges but $2 \mathbin{-\!-} 3$, we get

$$l(0) = 0000;\ l(1) = 10?0;\ l(2) = 11{*}?;\ l(3) = ?{*}11;\ l(4) = 0?01.$$

Now the "distance" between, say, $l(2)$ and $l(4)$ is $1 + \frac{1}{2} + 0 + \frac{1}{2} = 2$. Prove that, in general, the "distance" between $l(u)$ and $l(v)$ is $d(u,v)$, for any graph G. Also exhibit the labels when G is the Petersen graph, $\mathit{subsets}(2, 1, -4, 0, 0, 0, {}^{\#}1, 0)$.

f) In order to obtain a subcube labeling, we need to find a rule that changes each '?' to either '0' or '$*$', like flipping a coin but smarter. Show that there is such a rule. *Hint:* Thinking of T as an ordered tree, v_w can depend on whether v precedes or follows w in preorder, as well as on the parity of the distances of v and w from r.

160. [*M16*] Suppose G_1, G_2, H_1, and H_2 are connected graphs, with $G_1 \oplus G_2 \subseteq H_1 \oplus H_2$. True or false: Either $G_1 \subseteq H_1$ and $G_2 \subseteq H_2$ or $G_1 \subseteq H_2$ and $G_2 \subseteq H_1$.

ternary Hamming graph
Cartesian product
transitive
equivalence relation
Subcube labels
cube
a-code, see Asterisk codes for subcubes
Asterisk codes for subcubes
tree
ancestor
inclusive ancestor
subsets graphs (SGB)
Petersen graph
preorder
connected graphs

161. [*M15*] Which of the following potential "transitive laws" are true in general?

i) $G \subseteq G' \subseteq G''$ implies $G \subseteq G''$.
ii) $G \subseteq G' \subseteq G''$ implies $G \sqsubseteq G''$.
iii) $G \subseteq G' \sqsubseteq G''$ implies $G \subseteq G''$.
iv) $G \subseteq G' \sqsubseteq G''$ implies $G \sqsubseteq G''$.
v) $G \sqsubseteq G' \subseteq G''$ implies $G \subseteq G''$.
vi) $G \sqsubseteq G' \subseteq G''$ implies $G \sqsubseteq G''$.
vii) $G \sqsubseteq G' \sqsubseteq G''$ implies $G \subseteq G''$.
viii) $G \sqsubseteq G' \sqsubseteq G''$ implies $G \sqsubseteq G''$.

162. [*M17*] True or false: If $G \subseteq H$, G is connected, and H is a forest, then $G \sqsubseteq H$.

▶ **163.** [*20*] Let G be the pattern graph $K_{1,m} \oplus P_{a_1} \oplus \cdots \oplus P_{a_t}$, where $A = \{a_1, \ldots, a_t\}$ is a multiset of positive integers. Let T be the tree with root r and $mn+m$ additional vertices x_{jk} for $1 \le j \le m$, $0 \le k \le n$, whose edges are $r \mathrel{-\!\!-} x_{j0}$ and $x_{jk} \mathrel{-\!\!-} x_{j(k+1)}$. Prove that $G \subseteq T$ if and only if A can be partitioned into m multisets whose sums are each $\le n$. (And special cases of this partitioning problem are known to be NP-complete.)

▶ **164.** [*M23*] If G is a graph on vertices V, let $q(G)$ be the graph whose vertices are pairs (v,k) with $v \in V$ and $0 \le k < 5$, and whose edges $(v,k) \mathrel{-\!\!-} (v',k')$ are of three kinds: (i) $v = v'$ and $\{k,k'\} \in \{\{0,1\},\{1,2\},\{2,3\},\{3,4\},\{4,0\},\{2,4\}\}$; (ii) $v \mathrel{-\!\!-} v'$ and $\{k,k'\} = \{0,1\}$; (iii) $v \ne v'$, $v \not\mathrel{-\!\!-} v'$, and $\{k,k'\} = \{0,3\}$.

a) If G has n vertices, how many vertices does $q(G)$ have? How many edges?
b) Prove that G can be strictly embedded in H if and only if $q(G)$ can be embedded in $q(H)$. (Thus unlabeled ISIP is a special case of unlabeled SIP.)

165. [*M25*] Continuing exercise 164, reduce the unlabeled SIP to the unlabeled ISIP.

166. [*M20*] (*Labeled graph embedding.*) A SIP often has side constraints in practice. For example, when graphs represent molecules, each vertex might represent a particular kind of atom (carbon, hydrogen, etc.), and each edge might be labeled strong or weak.

In general, a *labeled subgraph isomorphism problem* is defined by a pattern graph G and a target graph H, where every vertex has zero or more labels l_i and every edge has zero or more labels L_j. Relations of compatibility are also defined between the pattern and target labels. The problem is to find every function f from the vertices of G to the vertices of H that satisfies four conditions: (i) If $v \ne w$ then $f(v) \ne f(w)$. (ii) If $v \mathrel{-\!\!-} w$ in G then $f(v) \mathrel{-\!\!-} f(w)$ in H. (iii) $l_i(v)$ is compatible with $l_i(f(v))$, for all i. (iv) If $v \mathrel{-\!\!-} w$ in G then $L_j(v,w)$ is compatible with $L_j(f(v),f(w))$, for all j.

a) Prove that every ISIP, possibly labeled, is a labeled SIP.
b) Given a labeled SIP, a vertex u of G, and a vertex $\hat{u}$ of H, show that the problem of finding all solutions with $f(u) = \hat{u}$ is a labeled SIP on the graphs $G \setminus u$ and $H \setminus \hat{u}$.

167. [*M30*] Show that the problem of testing $G \sqsubseteq H$ is NP-complete, even when G is a (free) tree and all vertices of G and H have degree ≤ 3. *Hint:* Reduce from 3SAT.

168. [*20*] If G is a graph with n vertices and m edges, let $\widehat{G}$ be the directed acyclic graph with $m+n$ vertices and $2m$ arcs obtained by replacing each edge $u \mathrel{-\!\!-} v$ by $u \longrightarrow uv \longleftarrow v$. Prove or disprove: (a) $G \subseteq H \iff \widehat{G} \subseteq \widehat{H}$; (b) $G \sqsubseteq H \iff \widehat{G} \sqsubseteq \widehat{H}$.

169. [*21*] Given an integer $M \ge 3$ and a graph H, is it hard to test if $C_M \sqsubseteq H$?

170. [*20*] A suitably small SIP problem can be solved as an exact cover problem using the options (53). Can an ISIP problem be solved in a similar way?

171. [*20*] Encode SIP and ISIP problems for *directed* graphs as exact cover problems.

172. [*HM30*] (S. Chatterjee and P. Diaconis, 2023.) Let $\mathcal{G}_N$ be a random graph on N vertices; each of the $\binom{N}{2}$ potential edges is independently present with probability $1/2$.

a) Prove that $\mathcal{G}_{\lfloor 2\lg n+2+\delta\rfloor} \not\sqsubseteq \mathcal{G}_n$ a.s., for fixed $\delta > 0$ as $n \to \infty$.
b) Prove that $\mathcal{G}_{\lceil 2\lg n-\delta\rceil} \sqsubseteq \mathcal{G}_n$ a.s., for fixed $\delta > 0$ as $n \to \infty$.

transitive laws
forest
partitioned
NP-complete
strictly embedded
Labeled graph embedding
chemistry
molecules
atom
compatibility
bounded degree
NP-complete
tree
3SAT
directed acyclic graph
exact cover
ISIP problem
directed graphs
Chatterjee
Diaconis
random graph
a.s.: Asymptotically almost surely

▶ **173.** [*21*] The reduced target graph $\widehat{H}$ obtained from `BRAIN83`(250) has 11 vertices in the left brain and 11 vertices in the right brain, with only two edges between them. Why does that make $G \subseteq \widehat{H}$ impossible, when G is Chvátal's graph (52)?

174. [*23*] Embed Chvátal's graph (52) into `BRAIN83` with 6 vertices in each half-brain.

175. [*24*] When $k \geq 12$ is a multiple of 6, Chvátal's graph of order k has k vertices $\{0, 0+, 1-, 1, 1+, \ldots, ((k/3)-1)+, 0-\}$ and $2k$ edges $j \mathrel{—} (j+1)$, $j \mathrel{—} j+$, $j \mathrel{—} j-$, $j+ \mathrel{—} (j+1)-$, $j+ \mathrel{—} (j+k/6)+$, $j+ \mathrel{—} (j+k/6)-$ (modulo k) for $0 \leq j < k/3$. (Thus (52) is the case of order 12.) Can his 18-vertex graph be embedded in `BRAIN83`?

176. [*23*] Is the flower snark graph J_5 (exercise 7.2.2.2–176) embeddable into `BRAIN83`?

▶ **177.** [*20*] Constrain the embeddings of (52) so that only the essentially different solutions are found (thus only 1/8 of the total number).

178. [*M22*] If A and B are multisets of integers, say that A *surpasses* B if A's kth largest element is greater than or equal to B's kth largest element, for $1 \leq k \leq |B| \leq |A|$.

a) Given a vertex v of a graph G, let $s(v) = \{\deg(u) \mid u \mathrel{—} v\}$ be the multiset of its neighbors' degrees. Prove that, whenever $G \subseteq H$ with an embedding function f, the multiset $s(f(v))$ surpasses $s(v)$, for all vertices v of G.

b) The obvious way to test whether or not $s(w)$ surpasses $s(v)$ is to sort the neighbors of w and v by their degrees, then to do a pairwise comparison of the sorted elements. But sorting might introduce a logarithmic factor into the running time. Explain how to perform that test in only $O(p + \deg(w))$ steps, where p is the maximum degree of any pattern vertex.

179. [*21*] Explain why LAD filtering from (58) forces `02` $\mapsto$ `LA`, after which further assignments to `01` and `03` and their neighbors get into trouble.

180. [*23*] What two solutions to the embedding problem (54) differ from Fig. 112?

181. [*24*] What's the largest n for which (a) $P_2 \,\square\, P_n \subseteq$ `USA`? (b) $P_3 \,\square\, P_n \subseteq$ `USA`?

182. [*15*] Do exercise 181 with $\sqsubseteq$ in place of $\subseteq$.

183. [*20*] If possible, embed half of a dodecahedron (namely, a pentagon surrounded by five other pentagons) into the `USA` graph.

184. [*21*] Explore the embedding of *simplex* graphs (triangular grids) into `USA`.

▶ **185.** [*M25*] (*Globally All Different filtering.*) When variables $x_1, \ldots, x_m$ are subject to an all-different constraint, the domains $D_1, \ldots, D_m \subseteq \{1, \ldots, n\}$ are said to be *feasible* if there's a matching of size m in the bipartite graph on vertices $\{x_1, \ldots, x_m\}$ and $\{y_1, \ldots, y_n\}$ whose edges are $x_i \mathrel{—} y_j$ when $j \in D_i$. A value $j \in D_i$ is said to be *removable* if $x_i \mathrel{—} y_j$ isn't in any feasible matching.

Let $x_1 \mathrel{—} y_{j_1}, \ldots, x_m \mathrel{—} y_{j_m}$ be a matching, and construct the following tripartite digraph T on $\{x_1, \ldots, x_m\}$, $\{y_1, \ldots, y_n\}$, and $\{\perp\}$: $x_i \longrightarrow y_{j_i}$ and $y_{j_i} \longrightarrow \perp$, for $1 \leq i \leq m$; $x_i \longleftarrow y_j$, if $j \in D_i$ and $j \neq j_i$, for $1 \leq i \leq m$; $y_j \longleftarrow \perp$, if $j \notin \{j_1, \ldots, j_m\}$. Prove that $j \neq j_i$ is removable from D_i if and only if x_i, y_j, and $\perp$ belong to different strong components of T.

▶ **186.** [*M26*] Continuing exercise 185, further theory elucidates the situation.

a) If $I \subseteq \{1, \ldots, m\}$, let $D(I) = \bigcup\{D_i \mid i \in I\}$. Prove that the domains are feasible if and only if $|D(I)| \geq |I|$ for all subsets I. *Hint:* Use Algorithm 7.5.1H (see page x).

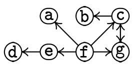
b) A subset I for which $|D(I)| = |I|$ is called a "Hall set." Prove that if a feasible family of domains has no nonempty Hall sets, it has no removable values.

c) In particular, nothing is removable if $|D_i| \geq m$ for $1 \leq i \leq m$.

BRAIN83
Chvátal's graph
Chvátal's graph
flower snark
essentially different solutions
multisets
comparison of multisets
sorting
LAD filtering
dodecahedron
simplex graphs
triangular grids
Globally All Different filtering
GAD
all-different
matching
bipartite graph
removable
tripartite digraph
strong components
Hopcroft–Karp algorithm
Hall set
critical block, see Hall set

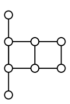
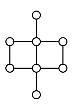
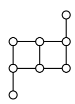
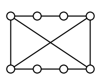
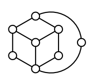
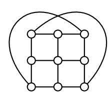
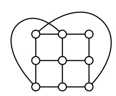
d) If I is a Hall set, explain why we can remove $D(I)$ from all domains D_j for $j \notin I$.
e) Prove that Hall sets of feasible domains are closed under union and intersection.
f) Prove that a feasible family of domains has no removable elements if and only if there's a partition of $\{1, \ldots, m\}$ into disjoint sets I_0, I_1, $\ldots$, I_r with disjoint domains $D(I_0)$, $D(I_1)$, $\ldots$, $D(I_r)$ such that the Hall sets are precisely the 2^r sets obtainable by unions of $\{I_1, \ldots, I_r\}$. (GAD filtering always yields such a family.)
g) Relate the partition of (f) to the tripartite digraph T of exercise 185.

▸ **187.** [*21*] When $m = n$ in exercise 185, every solution $x_1 \ldots x_n$ will be a *permutation* of $\{1, \ldots, n\}$. Improve the GAD filtering algorithm in that case.

188. [*29*] Find all (a) embeddings (b) strict embeddings of the digraph  into Agatha Christie's "Orient Express digraph" (Fig. 3 near the beginning of Chapter 7). As in the text's solution of (54), determine the initial domains; then repeatedly branch on a variable with smallest domain, using LAD and GAD filtering.

189. [*22*] Is the Petersen graph, minus two edges, embeddable in Chvátal's graph?

190. [*25*] Which of the following graphs are *strictly* embeddable in Chvátal's graph?

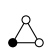
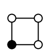
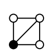
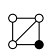
(i)  (ii)  (iii)  (iv)  (v)  (vi)  (vii) 

191. [*15*] True or false: $G \sqsubseteq H$ implies $G^{\le 2} \sqsubseteq H^{\le 2}$.

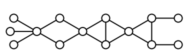
192. [*20*] Compute the vertex degrees of $G^{\le 2}$ and $H^{\le 2}$ when $G = P_4 \,\square\, P_5$ and $H = \texttt{USA}$. What do those statistics imply about the domain of G's "middle vertex" `12`?

193. [*M18*] Explain informally the meaning of the supplemental label d_G^S when S is the path P_{k+1} of length k, placing the designated vertex s at one end. Show that the degree of vertex v in $G^{\le 2}$ can be expressed in terms of $d_G^{P_2}(v)$ and $d_G^{P_3}(v)$.

194. [*23*] Consider the following motif graphs S, with designated vertex $s =$ '•':

(i) (ii) (iii) (iv) (v) (vi) (vii) (viii) (ix)

Compute the supplemental vertex labels $d_G^S(v)$, for each $v \in G =$ .

195. [*21*] Compute supplemental edge labels for each edge u — v of that same graph, using each of the motifs $S =$ •—■—○, •—■—○—○, ○—•—■—○. (Here $\bullet = s$, $\blacksquare = t$.)

196. [*20*] Draw the supplemental graphs $G^{S,k}$, for the graph G of exercise 194, when (i) $S =$ •—■—○ and $k = 1$; (ii) $S =$ •—■—○—○ and $k = 2$.

197. [*20*] Consider supplemental pair labels based on the motif $S = C_4$, with s and t at distance 2. Show that, in problem (54) of embedding $P_4 \,\square\, P_5$ into `USA`, such labels tell us that we can't map both `00` $\mapsto$ `MN` and `11` $\mapsto$ `MO`.

▸ **198.** [*24*] Using the supplemental graph $G^{S,2}$, where $S = P_2 \,\square\, P_3$ and its vertices of degree 3 are s and t, show that the initial domains for all six interior vertices of $P_4 \,\square\, P_5$ in the `USA` problem can be reduced to size 13 — less than half of what we had without it!

▸ **199.** [*20*] Restate the rules for LAD filtering in the presence of supplemental edge labels, pair labels, and graphs: Precisely what bipartite graph is required to have a matching of size $\deg(u)$ when we're trying to ascertain whether $u \mapsto v$ is locally feasible?

permutation
GAD filtering
strict embeddings
Christie
initial domains
LAD
GAD
Petersen graph
Chvátal's graph
supplemental label
supplemental vertex labels
supplemental edge labels
supplemental graphs
supplemental pair labels
`USA`
LAD filtering
bipartite graph

▶ **200.** [*20*] Extend the concept of supplemental labels and graphs to strict embeddings: Show that it's possible to construct functions $d_G(v)$, $\ell_G(v,w)$, and digraphs G^Σ such that $G \sqsubseteq H$ implies $d_G(v) \le d_H(f(v))$, $\ell_G(v,w) \le \ell_H(f(v),f(w))$, and $G^\Sigma \sqsubseteq H^\Sigma$, by analogy with (62), (64), and (65).

▶ **201.** [*22*] Many graph embedding problems are simple enough to be solved efficiently without maintaining a separate domain for each pattern variable. Instead, it suffices to keep track of the vertices adjacent to the ones already assigned. Say that an unassigned vertex is *near* if it has at least one assigned neighbor; otherwise it's *far*.

a) Show that the pattern vertices can be prearranged into a fixed (static) order $p_1p_2\ldots p_m$ so that, at level l of the search, vertices $\{p_1,\ldots,p_l\}$ have been assigned and $\{p_{l+1},\ldots,p_{r_l}\}$ are near, for some $r_l \ge l$. Furthermore $r_l > l$ for $0 < l < m$ if and only if the pattern is connected.

b) Explain how to maintain a permutation $t_1t_2\ldots t_n$ of the target vertices (dynamically) so that, at level l of the search, the current assignments are $t_j = f(p_j)$ for $1 \le j \le l$, and the vertices $\{t_{l+1},\ldots,t_{s_l}\}$ are near, for some $s_l \ge r_l$.

c) If p_{l+1} has q near neighbors, must $f(p_{l+1})$ have at least q near neighbors?

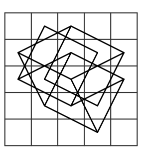
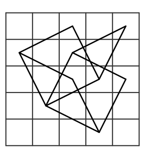
d) If p_{l+1} has q far neighbors, must $f(p_{l+1})$ have at least q far neighbors?

202. [*20*] Let D_1, …, D_m be domains $\subseteq \{1,\ldots,n\}$, with $|D_1| \le \cdots \le |D_m|$. In practice, much of the benefit of GAD filtering (exercise 185) can be achieved more cheaply: "Set $H \leftarrow U \leftarrow \emptyset$, and do the following for $1 \le j \le m$: Set $D_j \leftarrow D_j \setminus H$ and $U \leftarrow U \cup D_j$; then if $D_j = \emptyset$ or $|U| < j$, the domains aren't feasible; otherwise if $|U| = j$, set $H \leftarrow H \cup U$." Show that all values removed from D_j were indeed removable.

▶ **203.** [*25*] One of the main subtasks of a SIP solver is to assign a target value v' to a pattern vertex v, and to update all domains appropriately. Suggest appropriate data structures for making such assignments, when GAD filtering is relaxed as in exercise 202. Consider also the use of supplemental graphs. How can your structures efficiently propagate the constraints until all remaining domains have size 2 or more?

204. [*22*] Write an `MMIX` program for the algorithm of exercise 202, assuming that $n \le 64$ and that each domain is represented bitwise. Process the domains in order of increasing size, *without* assuming that $|D_1| \le \cdots \le |D_m|$, and show that the running time for the entire computation is only $O(m)$. *Hint:* Sort into $m+1$ buckets.

205. [*22*] (*Knight's grids.*) The graphs $P_2 \,\square\, P_7$ and $P_3 \,\square\, P_3$ can be seen as knight moves

 and 

within a 5×5 board; in other words, $P_2 \,\square\, P_7 \subseteq N_5$ and $P_3 \,\square\, P_3 \subseteq N_5$, where N_n is the $n\times n$ knight graph. (This scenario generalizes the classic notion of a "knight's tour.")

a) Find the largest n with $P_m \,\square\, P_n \subseteq N_8$ when $m = 2, 3, 4, 5, 6$.

b) Find the largest n with $P_m \,\square\, P_n \sqsubseteq N_8$ when $m = 2, 3, 4, 5, 6$.

c) Find the largest n with $P_2 \,\square\, C_n \subseteq N_8$.

d) Find the largest n with $P_2 \,\square\, C_n \sqsubseteq N_8$.

e) Find the largest n with $P_3 \,\square\, C_n \subseteq N_8$.

f) Find the largest n with $P_3 \,\square\, P_3 \,\square\, P_n \subseteq N_8$.

206. [*40*] Continuing exercise 205, let $f_m(t)$ be the largest n such that $P_m \,\square\, P_n \subseteq N_t$, and let $\overline{f}_m(t)$ be the largest n such that $P_m \,\square\, P_n \sqsubseteq N_t$. Compute $f_m(t)$ and $\overline{f}_m(t)$ for as

strict embeddings
domain
near vertices
far vertices
connected
GAD filtering
approximate GAD filtering
supplemental graphs
`MMIX`
bitwise
Sort
buckets
Knight's grids
knight moves
chessboard
knight graph

many values of $t \geq 3$ as you can, when $m = 2$, 3, and 4. [These problems make interesting benchmark tests for SIP and ISIP solvers — and the results are attractive too.]

▶ **207.** [*30*] (*Knights and queens.*) Hundreds of benchmarks for use in comparing and improving SIP and ISIP solvers have been proposed by J. Larrosa and G. Valiente [*Math. Structures in Comp. Sci.* **12** (2002), 403–422], who selected a wide variety of graphs from the Stanford GraphBase and proceeded to test all pairs. The smallest SIP instance that couldn't be solved within a reasonable time limit, according to C. Solnon's survey in 2018, turned out to be, "Is $N_8 \subseteq Q_8$?" In other words, are the knight moves on a chessboard isomorphic to a subset of the queen moves? Investigate this problem.

208. [*40*] Continuing exercise 207, study other values of $n \geq 3$ for which $N_n \subseteq Q_n$.

209. [*M25*] Is the $n \times n$ knight graph embeddable into the $n \times n$ rook graph for any n?

210. [*30*] Continuing exercise 207, the smallest ISIP instance that resisted solution in 2018 was quite weird: "Is $\mathit{book}(\texttt{"jean"}, 0, 5, 0, 178, 1, 0, 0) \sqsubseteq \mathit{games}(0, 0, 0, 0, 0, 0, 0, 0)$?" (The pattern graph has 75 vertices; the target graph has 120.) Investigate this problem.

▶ **211.** [*30*] (*Universal graphs.*) A five-vertex graph called the "bull" () is *3-universal*, in the sense that it contains every 3-vertex graph at least once as an induced subgraph.

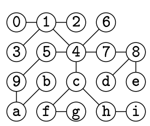
a) Find a 4-universal eight-vertex graph in which every vertex has degree 3 or 6.

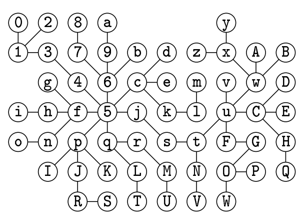
b) Find a 5-universal ten-vertex graph that contains an induced 4-universal graph with eight vertices.

212. [*27*] Find a "revolving-door Gray code for 4-vertex graphs" by finding 4-vertex subsets $V_1, V_2, \ldots, V_{11}$ of the graph H in exercise 211(a) such that the induced subgraphs $H \mid V_1, H \mid V_2, \ldots, H \mid V_{11}$ are the eleven possible graphs on four vertices. Each V_{j+1} should share three vertices with V_j.

▶ **213.** [*34*] (*Subtree isomorphism.*) Let S and T be free trees, having m nodes and n nodes, respectively. A remarkably efficient algorithm, due to D. W. Matula, is able to decide whether or not $S \subseteq T$ (and $S \sqsubseteq T$) in only $O(mn\sqrt{s})$ steps, where s is the maximum inner degree of any node in S (the number of nonleaf neighbors).

a) Get ready to understand Matula's algorithm by solving the problem by hand when

S = 

($m = 19$, $s = 4$); T = 

($n = 59$).

b) In general, let the nodes of S be $\{0, 1, \ldots, m-1\}$, where $\deg(0) = 1$. We think of 0 as S's *root*; every other node r has a *parent*, $p(r)$, which is the first node on the path from r to 0. Similarly, the nodes of T are $\{0, 1, \ldots, n-1\}$; but instead of regarding T as rooted, we consider it to have $2(n-1)$ directed arcs $u \longrightarrow v$, one for each edge $u — v$ of T. This arc e is denoted for convenience by $e = {}^{u}_{v}$. Let S_r be the subtree of S consisting of all nodes whose path to 0 passes through r. Similarly, when $e = {}^{u}_{v}$, let T_e be the subtree of T consisting of all nodes whose path to u passes through v. Is $S_r \subseteq T_e$ in (a), when $r = 7$, $u = \texttt{u}$, and $v = \texttt{w}$?

c) Let $\{r_1, \ldots, r_k\}$ be the children of r in S, let $e = {}^{u}_{v}$, and let $\{w_1, \ldots, w_l\}$ be the children of v in T. Under what conditions is it possible to embed S_r into T_e, with $r \mapsto v$, based on the embeddability of smaller subtrees?

benchmark tests
Knights and queens
queens
benchmarks
Larrosa
Valiente
Stanford GraphBase
Solnon
rook graph
book graphs
games graphs
benchmarks
Universal graphs
bull
4-vertex graphs
5-vertex graphs
revolving-door Gray code for 4-vertex graphs
Gray code for 4-vertex graphs
Subtree isomorphism.
free trees
Matula
inner degree
root
parent
subtree

d) Let `sol[`r`][`e`]` $= [S_r \subseteq T_e$ with $r \mapsto \mathrm{root}(T_e)]$, for $0 < r < m$ and $0 \le e < 2n-2$. Explain how to compute all elements of this $(m-1)\times(2n-2)$ matrix by solving $O(mn)$ maximum bipartite matching problems.

e) Furthermore, if v has $l+1$ neighbors in T, the $l+1$ matching problems with $\mathrm{root}(T_e) = v$ are almost the same and they can be solved simultaneously.

f) Sketch the details of a complete implementation, using Algorithm 7.5.1H (the Hopcroft–Karp algorithm) for matching. What's the `sol` matrix for problem (a)?

214. [*29*] Evaluate Matula's algorithm (exercise 213) empirically by applying it to several classes of free trees:

a) Let S run through all 551 free trees with $m = 12$, and let T run through all 19320 free trees with $n = 16$.

b) Let S and T be uniformly random free trees with $m = 25$ and $n = 1000$.

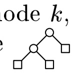
c) Let T be a random free tree with $n = 1000$; obtain S by repeatedly removing a random leaf, 100 times.

▸ **215.** [*20*] The *feedback vertex set* problem asks whether a given digraph D has a set of k vertices that cover every directed cycle. Show that it's a special case of ISIP.

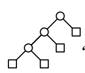
▸ **216.** [*23*] Exercise 4 illustrates how any finite CSP can be encoded as an XCC problem by listing its positive table constraints — the tuples that satisfy the given relations. Show that any finite *binary* CSP can be encoded as an XC problem by listing its *negative* table constraints — the ordered pairs that do *not* satisfy the given relations.

Illustrate your method by explaining how to find all *radio colorings* of a given graph, using the colors $\{0, 1, \ldots, d-1\}$. (See exercise 7.2.2.2–36.)

217. [*21*] Apply exercise 216 to enumerate all optimum radio colorings of (a) $P_3 \,\square\, P_3$; (b) Petersen's graph; (c) Chvátal's graph; (d) Mycielski's graph M_4.

218. [*20*] Any extended binary tree with d leaves and height h defines an h-bit *prefix code* for a d-element domain: The representation of k is the path to external node k, using 0 for a left branch and 1 for a right branch. For example, the binary tree  defines the 2-bit codewords $(00, 01, 1*)$ for $k = (0, 1, 2)$.

a) Is this the same as Table 2's "prefix encoding"?

b) What's the prefix code for the extended binary tree  ?

c) Relate that code to the "weakened encoding" of Table 2.

219. [*20*] Reverse-engineer Table 2's "reduced encoding." What makes it tick?

220. [*20*] How many variables, clauses, and total literals are generated by each of the encodings in Table 2, when the given graph has V vertices and E edges?

221. [*17*] Why is the Sierpiński gasket graph $S_n^{(3)}$ uniquely 3-colorable?

222. [*20*] True or false: The graph $S_n^{(3)}$ minus any edge is *not* uniquely 3-colorable.

▸ **223.** [*M25*] Since $S_n^{(3)}$ is a subgraph of the triangular grid, we can also name its edges and vertices by using the barycentric even/odd coordinate system of answer 7.2.2.1–124. Give formulas for the barycentric coordinates of triangle $a_1 \ldots a_{n-1}$ and its vertices, assuming that vertex $12\ldots2 = 21\ldots1 \mapsto (0,0,0)$. What are the coordinates of $0\ldots0$, $1\ldots1$, and $2\ldots2$? *Hint:* Show that every odd number between -2^n and $+2^n$ has a unique *binary representation* $(b_1 \ldots b_n)_2$ in which every digit b_j is ±1.

224. [*18*] What clauses can be used with Table 2 to ensure that vertices u, v, and w will have the respective colors 0, 1, and 2?

225. [*29*] Apply the encodings of Table 2 to the problem of 3-coloring $\widehat{S}_n^{(3)}$ for small n. How well do they work with Algorithms 7.2.2.2L and 7.2.2.2C?

maximum bipartite matching
bipartite matching
matching
Hopcroft
Karp
Matula
feedback vertex set
cover
directed cycle
positive table constraints
table constraints
CSP represented as XCC
XCC representation of CSP
CSP as XC
negative table constraints
radio colorings
L(2,1) labeling, see radio coloring
Petersen's graph
Chvátal's graph
Mycielski's graph M_4
extended binary tree
binary tree
prefix code
reduced encoding
Sierpiński gasket graph
triangular grid
barycentric even/odd coordinate system
even/odd coordinate system
binary representation

▸ **226.** [*M30*] Find a simple formula for the size of the backtrack tree that arises when proving that $\widehat{S}_n^{(3)}$ cannot be 3-colored. Each node should branch on a vertex with fewest available colors, breaking ties by choosing the lexicographically smallest.

227. [*40*] The pinched Sierpiński gasket $\widehat{S}_4^{(3)}$ remains uncolorable with three colors even if we remove the edges $0000 —\!\!— 0001$, $0101 —\!\!— 0111$, $0222 —\!\!— 2002$, $2202 —\!\!— 2222$, $2212 —\!\!— 2222$. What's the largest number of edges that can be removed from $\widehat{S}_n^{(3)}$ before it becomes 3-colorable?

▸ **228.** [*21*] What clique hints, analogous to (69), are most appropriate for the (a) log, (b) weakened, (c) reduced, and (d) prefix encodings?

229. [*24*] How could a SAT solver learn '$(0202_2 \vee 0222_2)$' from the prefix-encoded clauses for 3-coloring $\widehat{S}_4^{(3)}$? (See (70); assume that the clique hints have been given.)

▸ **230.** [*25*] Exercise 7.2.2.1–117 shows that graph coloring is an XC problem. Empirically, how long does it take Algorithm 7.2.2.1X to show that $\widehat{S}_n^{(3)}$ cannot be 3-colored?

231. [*M46*] Can an exponential lower bound be proved on the refutation length of the clauses for 3-uncolorability of $\widehat{S}_n^{(3)}$? (See Theorem 7.2.2.2B.)

232. [*24*] Repeat exercise 225, but test flower snark line graphs $L(J_q)$ instead of $\widehat{S}_n^{(3)}$.

233. [*40*] The flower snark line graph $L(J_q)$ for odd q actually remains 3-uncolorable even if we remove any one of its $12q$ edges. What's the largest number of edges that can be removed before it becomes 3-colorable?

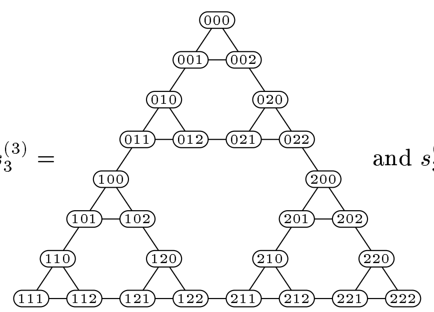
234. [*16*] The graph $S_3^{(4)}$ in Fig. 114 has $(4^3+4)/2 = 34$ vertices, but only 27 of them are visible. What are the names of the seven hidden vertices? (Give both names.)

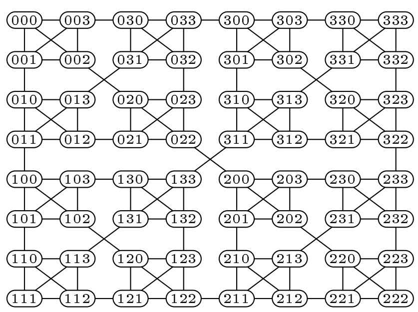
235. [*10*] What's a simpler name for the Sierpiński simplex graph $S_n^{(d)}$ when $d = 2$?

236. [*M15*] True or false: $S_n^{(d)}$ is an induced subgraph of $S_n^{(d')}$ when $d \le d'$.

237. [*16*] Almost every vertex of $S_n^{(d)}$, $\widehat{S}_n^{(d)}$, and $\overline{S}_n^{(d)}$ has degree $2d-2$. What vertices are the exceptions?

▸ **238.** [*M17*] The "proper" *Sierpiński graphs* $s_n^{(d)}$, exemplified by

$s_3^{(3)} =$ 

and $s_3^{(4)} =$ 

,

are different from but strongly related to the Sierpiński simplex graphs $S_n^{(d)}$. In general, $s_n^{(d)}$ has d^n vertices $a_1 \ldots a_n$, for $0 \le a_j < d$, and two kinds of edges:

i) clique edges $a_1 \ldots a_{n-1}j —\!\!— a_1 \ldots a_{n-1}k$, for $0 \le j < k < d$;

ii) nonclique edges $a_1 \ldots a_i jk \ldots k —\!\!— a_1 \ldots a_i kj \ldots j$, for all $0 \le i < n-1$ and $0 \le j < k < d$.

Notice that almost every vertex has degree d; this property is akin to exercise 237.

a) Give a formula for the total number of edges in $s_n^{(d)}$.

search tree size
analysis of algorithms
MRV heuristic
clique hints
log
weakened
reduced
prefix encoding
XC problem
exact covering problem
lower bounds for resolution
refutation length
flower snark line graphs
Sierpiński simplex graph
induced subgraph
subgraph
Sierpiński graphs

b) What's an intuitive way to obtain $S_n^{(d)}$ from $s_{n+1}^{(d)}$?

c) What's an intuitive way to obtain $S_n^{(d)}$ from $s_{n-1}^{(d)}$?

239. [*M25*] Show that, in every d-coloring of $S_n^{(d)}$, for $n > 1$, the number of pure vertices having a given color is congruent to d (modulo 2).

▶ **240.** [*22*] Generalize the encodings of '$u \neq v$' in Table 2 from ternary to d-ary.

▶ **241.** [*23*] Generalize the clique hints of exercise 228 to d-ary. Illustrate the case $d = 5$.

242. [*20*] Apply exercise 241 to the problem of 8-coloring the 8×8 queen graph, using the direct encoding. (See test problem K1 in Table 7.2.2.2–6.)

243. [*M26*] When we try to prove that $\overline{S}_n^{(4)}$ isn't 4-colorable, we can assume without loss of generality that vertices $0\ldots00$, $0\ldots01$, $0\ldots02$, $0\ldots03$ have the respective colors 0, 1, 2, 3. Show that the remaining problem still has 6-fold symmetry. How could that symmetry be exploited?

244. [*24*] Repeat exercise 225, but test $\overline{S}_n^{(4)}$ instead of $\widehat{S}_n^{(3)}$. (Use clique hints.)

245. [*24*] Repeat exercise 225, but test $\widehat{S}_n^{(5)}$ instead of $\widehat{S}_n^{(3)}$. (Use clique hints.)

246. [*34*] Apply a state-of-the-art SAT solver to the clauses for $\widehat{S}_n^{(3)}$, $\overline{S}_n^{(4)}$, $\widehat{S}_n^{(5)}$, and $L(J_q)$ for various encodings, and compare the results to those obtained with Algorithm 7.2.2.2C in exercises 225, 232, 244, and 245.

▶ **247.** [*24*] (*The haystack problem.*) Consider n^2 variables x_{ij} for $0 \le i, j < n$, each with domain $\{0, 1, \ldots, n-1\}$, subject to the following constraints: (i) $x_{ij} \neq x_{ij'}$ when $j \neq j'$. (ii) $x_{i0} + x_{ij} > 1$ when $0 < i, j < n$. (iii) $x_{i0} = x_{0i}$ when $0 < i < n$.

a) Prove that this CSP is unsatisfiable.

b) Formulate it as an exact cover problem, and try it with algorithms of §7.2.2.1.

c) Formulate it as a satisfiability problem, and try it with algorithms of §7.2.2.2.

248. [*25*] Explain how to generate SAT clauses that efficiently encode the relation '$u \le v - t$', when variables u and v are represented with the log encoding and t is constant. Illustrate your construction in the cases '$u \le v + 1$' and '$u \le v - 2$', assuming that $u = (u_8u_4u_2u_1)_2$ and $v = (v_8v_4v_2v_1)_2$.

249. [*20*] Shorten the direct encoding of (78) by simplifying (79). (For example, $(\bar{u}_0 \vee \bar{v}_1 \vee \bar{w}_1) \wedge (\bar{u}_0 \vee \bar{v}_2 \vee \bar{w}_1)$ can be replaced by $(\bar{u}_0 \vee v_0 \vee \bar{w}_1)$.)

250. [*17*] What are the direct and support encodings of '$uv \in \{00, 01, 12, 20\}$'?

▶ **251.** [*20*] If the binary relation of exercise 250 is treated as a k-ary relation with $k = 2$ and "binarized" by the general strategy of (77), what support clauses do we get?

252. [*11*] Derive $\overline{R}_{001}$, $\overline{R}_{010}$, $\ldots$, $\overline{R}_{211}$ from (80)–(82) and (R_{000}) by unit propagation.

253. [*M16*] Let $R(v_1, \ldots, v_k)$ be a k-ary relation, where variable v_j has domain $[0\,.\,.\,d_j)$ for $1 \le j \le k$. If R contains exactly G tuples, how many total literals are in the (a) preclusion (b) support clauses, when R is encoded for SAT?

254. [*M20*] Prove that the direct encoding doesn't need the at-most-one clauses.

▶ **255.** [*M22*] Use resolution to derive the clauses for $b \in D_v$ in (76) from the clauses for $a \in D_u$. (Thus half of the support clauses for R are redundant.)

256. [*22*] How many of the 2^{27} ternary relations on variables whose domain size is 3 can be expressed as the conjunction of *binary* relations on those variables?

pure vertices
queen graph
direct encoding
symmetry
augmented Sierpinski tetrahedron
clique hints
pinched Sierpinski simplex
clique hints
haystack problem
exact cover problem
satisfiability problem
encode
log encoding
direct encoding
support encoding
unit propagation
at-most-one clauses
resolution
ternary relations
binary relations
decomposable constraints

257. [*23*] Two of the 2^{27} ternary relations on ternary domains are equivalent to each other if they differ only with respect to permuting the elements of the domains or permuting the order of the variables (or both). Thus, an equivalence class might contain as many as $3!^4 = 1296$ different relations. How many equivalence classes are there? How many of them satisfy the special condition of exercise 256? How many "come close"?

258. [*20*] When variables u, v, and w all have the domain $[0\,..\,d)$, let $R(u,v,w)$ be the median-fixing relation '$\langle uvw\rangle = c$'. Is R the conjunction of its three binary projections?

▶ **259.** [*20*] Let $R(a,b,c,d,e)$ be the quinary relation whose tuples are `WORDS`(1000), the most common 1000 five-letter words of English: `which`, `there`, ..., `ditch`. What tuples are not in R, but are in all of its projections $R_a(b,c,d,e)$, $R_b(a,c,d,e)$, ..., $R_e(a,b,c,d)$?

▶ **260.** [*21*] One way to perform unit propagation is to (i) delete any clause that contains a true literal; (ii) remove all false literals from all clauses; (iii) regard a unit clause as a true literal; (iv) regard an empty clause as a contradiction. If this process has been applied to the support encoding S for some nonempty relation $R(v_1,\ldots,v_k)$, prove:

a) There will be no contradiction.
b) If no clauses remain, R is satisfied by the true literals v_{1a_1}, ..., v_{ka_k}.
c) Otherwise the remaining clauses are the support encoding for some relation R'.
d) If literal v_a remains, there's a solution with v_a true and another with v_a false.
e) If literal v_a remains, statements (a), (b), and (c) hold also for the clauses $S \wedge (v_a)$.
f) If literal v_a remains, statements (a), (b), and (c) hold also for the clauses $S \wedge (\bar{v}_a)$.

▶ **261.** [*20*] Formulate the CSP (87) as an exact cover problem with primary items w, x, y, z, and with three options for each primary item (one for each domain element).

262. [*20*] As an alternative to exercise 261, formulate (87) as an XCC problem, in the style of the answer to exercise 4.

263. [*18*] Test your knowledge of "corner cases" in basic definitions by determining which of the following statements (if any) are true and which of them (if any) are false.

a) The domain of every inactive variable in a partially solved CSP has size 1.
b) The domain of every active variable in a partially solved CSP has size > 1.
c) The domain of every active variable in a partially solved CSP has size > 0, if we have forward consistency.
d) Same as (c), but with domain consistency instead of forward consistency.
e) If all variables are active, there is forward consistency.

The remaining statements refer to a simple CSP that has four variables $\{w,x,y,z\}$, a single constraint '$w+x < y+z$', and domains $D_w = D_x = D_y = \{1\}$; $D_z = \{0,1,2,3\}$:

f) If w, x, y, and z are active, there is forward consistency.
g) If w is inactive, but x, y, and z are active, there is forward consistency.
h) If w and x are inactive, but y and z are active, there is forward consistency.
i) If w, x, and y are inactive, but z is active, there is forward consistency.
j) Same as (g), (h), (i), but with domain consistency instead of forward consistency.

264. [*20*] Show that forward consistency and domain consistency are almost equivalent, when the CSP being solved is a coloring problem (all constraints are '$\neq$'), assuming that we branch on a variable of domain size 1 whenever possible.

▶ **265.** [*28*] Prove that, when reducing domains while solving the n queens problem, domain consistency will yield no improvement over forward consistency until at least $\lceil n/3\rceil - 1$ queens have been placed. But find a placement of five queens on a 16×16 board for which DC reduces more domains than FC does.

ternary relations
equivalent
median
`WORDS`(1000)
five-letter words
unit propagation
support encoding
exact cover problem
CSP represented as XC
XCC problem
inactive variable
active variable
active variable
forward consistency
domain consistency
domain consistency
coloring problem
n queens problem
domain consistency
forward consistency

266. [*25*] If four nonattacking queens are placed on a 16×16 board, can a solution to the 16 queens problem always be obtained by placing twelve more queens?

267. [*25*] Modify step D4 of Algorithm D so that the case $w = v$ can often be omitted.

268. [*23*] Design a domain filtering algorithm that applies to any CSP in which all constraints are binary, by adapting Algorithm 7.1.1C (the "Horn core algorithm") to the present context. Your algorithm should either establish domain consistency or conclude that the problem is unsatisfiable.

269. [*27*] Extend exercise 268 to nonbinary constraints.

▸ **270.** [*20*] (*Transforming k-ary constraints to binary.*) Show that any CSP $\mathcal{P}$ with n variables and m constraints, of arities $k_1, \ldots, k_m$, is equivalent to a CSP $\mathcal{P}^*$ with $m + n$ variables and $k_1 + \cdots + k_m$ *binary* constraints. Furthermore, $\mathcal{P}$ is domain consistent if and only if $\mathcal{P}^*$ is domain consistent. *Hint:* See (77).

▸ **271.** [*25*] (*The dual of a CSP.*) Continuing exercise 270, show that $\mathcal{P}$ is also equivalent to a "dual" CSP $\mathcal{P}^D$ that has binary constraints on only m variables. Does domain consistency in $\mathcal{P}$ imply domain consistency in $\mathcal{P}^D$?

272. [*23*] Exercise 43 discusses a junction-oriented way to model the line labeling problem as a CSP, in contrast to the line-oriented approach that has been followed in the text and illustrated in (21) and (22). (In fact, the junction-oriented model is precisely what exercise 271 calls the *dual* of the line-oriented model.)

Compare the results of junction-oriented domain filtering, when applied to the histoscape example (20), with the results of line-oriented filtering in (91).

273. [*21*] Describe the top levels of the search tree for the CSP $\mathcal{P}$ of (21) and (22), when the MRV heuristic is used to select a variable for d-way branching, and when domains are reduced by forward consistency only. Initially all domains are $\{+, -, >, <\}$.

▸ **274.** [*21*] Do exercise 273 but with the binary CSP $\mathcal{P}^*$ of exercise 270 instead of $\mathcal{P}$.

275. [*18*] By exercise 265, the constraints of the 4 queens problem are domain consistent. Show that *singleton domain consistency* will reduce each domain size from 4 to 2.

276. [*M22*] Suppose there's a binary constraint R_{uv} for every pair of variables u and v, where $R_{vv} = \{aa \mid a \in D_v\}$ and $R_{vu} = \{ba \mid ab \in R_{uv}\}$. These constraints are called *path consistent* if u and v are consistent with w for all variables u, v, w, in the sense that

$$ab \in R_{uv} \text{ implies that at least one } c \in D_w \text{ satisfies } ac \in R_{uw} \text{ and } bc \in R_{vw}.$$

(Notice that this condition, with $u = v$, implies domain consistency.)

Consider, for example, the 5 queens problem with variables $\{r_1, \ldots, r_5\}$, where $r_i = j$ means that there's a queen in row i, column j. Let $R_{ii'}$ denote $R_{r_i r_{i'}}$. Initially

$$R_{ii'} = \{jj' \mid (i = i' \wedge j = j') \vee (i \neq i' \wedge j \neq j' \wedge |i - i'| \neq |j - j'|)\};$$

but these constraints aren't path consistent: We must remove 25 from R_{13} because $r_1 = 2$ and $r_3 = 5$ wipes out r_2. Then we must remove 21 from R_{15}, to avoid wiping out r_3. And then we must remove 24 from R_{14}, lest r_5 be wiped out.

What path-consistent relations $R_{ii'}$ remain, after we've done all such removals?

▸ **277.** [*M30*] Consider the $d \times d'$ matrix (r_{ij}), where $r_{ij} = [ij \in R]$ characterizes a binary relation R. When doing domain filtering, we want to know the support vectors $s_i = [\text{row } i \text{ of } r \text{ is nonzero}]$ and $s'_j = [\text{column } j \text{ of } r \text{ is nonzero}]$, for $0 \leq i < d$ and $0 \leq j < d'$. It's easy to compute s_i and s'_j by simply scanning row i or column j until we see a 1. But let's suppose that it's *expensive* to access the array r (that is, to decide whether or not $ij \in R$); so we want to avoid checking r_{ij} whenever possible.

16 queens problem
domain filtering
binary constraints
Horn core algorithm
k-ary constraints to binary
arities
binary constraints
domain consistent
dual of a CSP
line labeling problem
histoscape
MRV heuristic
d-way branching
forward consistency
4 queens problem
singleton domain consistency
binary constraint
composition of binary relations
path consistent
domain consistency
5 queens problem
wiped out
domain filtering
support vectors

The following two-pass procedure has been suggested, using an auxiliary $d \times d'$ Boolean matrix m to remember where we've already looked in r. Initially m, s, and s' are zero. "Pass 1. For $0 \le i < d$ do this: For $0 \le j < d'$, set $m_{ij} \leftarrow 1$; if $r_{ij} = 1$, set $s_i \leftarrow 1$, $s'_j \leftarrow 1$, and break out of the loop on j. Pass 2. For $0 \le j < d'$ with $s'_j = 0$ do this: For $0 \le i < d$ with $m_{ij} = 0$, if $r_{ij} = 1$, set $s'_j \leftarrow 1$, and break out of the loop on i."

a) Analyze that algorithm, assuming that each entry of the matrix is independently random, with $\Pr(r_{ij} = 1) = p$ for all i and j. Given i and j, what is the probability that r_{ij} will be examined in Pass 1? In Pass 2?

b) Improve Pass 1. *Hint:* We can often avoid looking at r_{ij} if we know that $s'_j = 1$.

c) Experiment with the improved algorithm when, say, $d = d' = 100$.

278. [*M41*] Does the algorithm of exercise 277(b) have minimum expected cost, over all support-finding algorithms for random $d \times d'$ matrices of density p?

▸ **279.** [*M25*] The *chain problem* is a CSP with n variables $x_1, \ldots, x_n$, of which x_1 through x_m are "sources" and x_n is a "sink." All variables have domain $\{0, 1, 2\}$. There are m binary constraints, '$x_i \ne x_n$' for $1 \le i \le m$; also $n - m$ ternary constraints,

$$\text{'either } x_i = x_{j(i)} \text{ or } x_i = x_{k(i)}\text{'} \qquad \text{for } m < i \le n,$$

where two indices with $0 < j(i) < k(i) < i$ are prescribed for every such i. (Notice the similarity with addition chains, Boolean chains, resolution chains, etc.)

a) Explain why every chain CSP is unsatisfiable.

b) Express any given chain CSP as an XCC problem with $\le 15n$ options.

c) Exactly how many chain CSPs are possible, given m and n with $1 \le m \le n$?

d) Experiment with XCC solvers on uniformly random chain CSPs that have been formulated as in (b), when $m = 24$ and n varies.

e) Exhibit supports that establish domain consistency for every chain CSP. But show that exercise 268 will find a contradiction just after x_n is assigned a value.

280. [*M28*] Analyze the problems of exercise 279: Let $P_{m,n}$ be a random chain problem, where every possible choice of the pairs $(j(i), k(i))$ for $i > m$ is equally likely.

a) Let $S_{m,n}$ be the expected total number of sinks in $P_{m,n}$. (A sink is a variable x_i that isn't in $\{j(i+1), k(i+1), \ldots, j(n), k(n)\}$.) Find a simple formula for $S_{m,n}$.

b) A sink, x_i, for which $i < n$, is not connected to x_n. Neither is a variable that's constrained only by unconnected variables. Find a recurrence by which we can compute $C_{m,n}$, the expected number of variables of $P_{m,n}$ that *are* connected to x_n. (For example, $C_{3,5} = 66/18$.) What is $C_{24,64}$?

c) Find a recurrence by which we can compute $c_{m,n}$, the probability that all variables of $P_{m,n}$ are connected to x_n. (For example, $c_{3,5} = 3/18$.) What is $c_{24,64}$?

281. [*HM41*] What's the asymptotic behavior of $C_{m,n}$, for fixed m and large n?

282. [*M21*] How many solutions does the (d, n)-modstep problem have? (See (93).)

▸ **283.** [*M22*] Analyze the behavior of a backtrack search for all solutions to the (d, n)-modstep problem when $d \ge n - 1$ and $n \to \infty$, using MRV and assuming that filtering is done by maintaining (a) forward consistency (only); (b) domain consistency.

284. [*M20*] Exactly how many permutations of $\{1, 2, \ldots, n\}$ have $p_{j+1} < p_j + d$, for $1 \le j < n$, given a number d with $1 \le d \le n$?

285. [*M21*] For every subset $S \subseteq \{1, \ldots, n-1\}$, prove that exactly one slow growth permutation of $\{1, 2, \ldots, n\}$ has the property "$p_{j+1} > p_j$ if and only if $j \in S$."

286. [*M20*] True or false: The inverse of a slow growth permutation has slow growth.

Analyze
random
support-finding algorithms
chain problem
sources
supports
domain consistency
chain problem
sink
recurrence
asymptotic behavior
modstep problem
MRV
forward consistency
slow growth permutation
inverse

▸ **287.** [*23*] Construct an exact cover problem whose solutions are the 2^{n-1} slow growth permutations of $\{1, 2, \ldots, n\}$. There should be n^2 options, each containing $O(\log n)$ items. *Hint:* Use the pairwise ordering trick of exercise 7.2.2.1–20.

▸ **288.** [*21*] Use exercise 286 to solve exercise 287 with more restrictive options.

▸ **289.** [*20*] (*Fillomino.*) A "fillomino pattern" is a labeling of grid cells with positive integers in such a way that every cell labeled d is rookwise connected to exactly d cells that have the same label. (Equivalently, it's a way to pack a shape with polyominoes, where no two d-ominoes have an edge in common.) For example, a more-or-less random fillomino pattern is shown at the right.

```
33877722
31888777
44488179
33418899
13225999
88855926
81859926
88856666
```

A "fillomino puzzle" is a labeling with positive integers and blanks, for which exactly one fillomino pattern can be obtained by filling in the blanks.

If, for instance, we want to solve puzzle (i) below, it's clear that the upper left corner cell must be labeled 2, and that there must be a 3 at the lower left.

(i)
```
␣14␣
2␣␣␣
1␣␣2
␣3␣␣
```
; (ii)
```
214␣
2␣␣␣
1␣␣2
33␣␣
```
; (iii)
```
214␣
24␣␣
1␣␣2
33␣␣
```
.

So (ii) is forced; and with a bit of thought we see that the blank below the upper 1 can't be 3 or more than 4. Hence we reach (iii), and ultimately a unique solution.

Show that one of the six clues in puzzle (i) is actually redundant. But none of the other five can be removed, without spoiling the puzzle by allowing additional patterns.

290. [*M24*] Compute the exact number of $2 \times n$ fillomino patterns for $n = 1, 2, 3, \ldots,$ until reaching an n for which that number exceeds 10^{100}.

291. [*21*] The "fillomino problem" is to find every fillomino pattern that's consistent with a given partial labeling. Formulate it as an exact cover problem.

292. [*22*] Try your luck with the following selected fillomino puzzles:

(a)
```
2␣1␣2␣
1␣␣␣12
␣1␣3␣␣
␣␣␣␣␣␣
␣␣3␣1␣
21␣␣␣1
␣2␣1␣2
```
; (b)
```
33␣␣␣␣␣␣␣
31415926␣
␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣
535897932
␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣
␣38462643
␣␣␣␣␣␣␣44
```
; (c)
```
␣␣␣␣␣2␣␣␣␣
␣24␣␣8␣24␣
␣68␣␣6␣68␣
␣␣␣␣␣4␣␣␣␣
2684␣␣␣␣␣␣
␣␣␣␣␣␣6482
␣␣␣␣4␣␣␣␣␣
␣24␣8␣␣24␣
␣68␣6␣␣68␣
␣␣␣␣2␣␣␣␣␣
```
; (d)
```
1␣341412␣␣
␣␣␣␣␣␣␣␣␣4
2␣␣3␣2␣␣3␣
␣2␣3␣33␣1␣
␣1␣␣␣␣␣␣␣2
2␣␣␣␣␣␣␣4␣
␣4␣44␣3␣2␣
␣3␣␣1␣3␣␣1
1␣␣␣␣␣␣␣␣␣
␣␣342242␣3
```
; (e)
```
␣24␣␣␣␣␣␣␣
␣15␣␣␣␣␣13
␣␣␣36␣␣␣45
␣␣␣46␣63␣␣
␣␣␣␣␣␣31␣␣
␣␣65␣␣␣␣␣␣
␣␣52␣42␣␣␣
34␣␣␣13␣␣␣
23␣␣␣␣␣41␣
␣␣␣␣␣␣␣24␣
```
.

293. [*24*] There are 59,951 4×4 fillomino patterns Φ whose labels don't exceed 5. Exhaustively study them all, finding every valid puzzle without redundant clues for which Φ is the solution. What interesting statistics and extremal examples lurk among them?

▸ **294.** [*M28*] Prove that the solution to a fillomino puzzle whose maximum clue is s cannot include a d-omino with $d > 4s+2$. Can you construct such puzzles with $d = 4s+2$?

▸ **295.** [*M30*] Characterize all valid $m \times n$ fillomino puzzles whose clues are all 1s.

296. [*HM40*] Let $\#_d(\Phi)$ be the number of cells labeled d in the fillomino pattern Φ, and let $\delta_d = \limsup_{n\to\infty} \#_d(\Phi_n)/n^2$ be the maximum density of d's in any infinite sequence of $n \times n$ patterns Φ_n. Determine δ_d for as many d as you can, and show that $\delta_d = 1 - \Theta(1/\sqrt{d}\,)$ as $d \to \infty$.

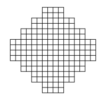
297. [*41*] Construct a valid 9×9 fillomino puzzle that is also a valid sudoku puzzle.

exact cover problem
pairwise ordering trick
fillomino
rookwise connected
polyominoes
pi, "random" example
e, "random" example
unique solution
googol=10^{100}
exact cover problem
pi, random
density
sudoku

298. [*23*] An octomino that contains a 2×3 rectangle is called a "chunky-oct." There are three kinds: Type S, symmetrical (e.g.,); Type D, asymmetrical, $2\times 3+1\times 2$ (e.g.,); Type M, asymmetrical, $2\times 3+1\times 1+1\times 1$ (e.g., ).

a) How many chunky-octs are of Type S? Type D? Type M?

b) Pack them into the scaled-up Aztec diamond shown, in such a way that all chunky-octs of the same type are kingwise connected.



299. [*21*] The trail at the right of Fig. 117 includes many unnecessary entries; for example, '$\genfrac{}{}{0pt}{}{x\ \ x}{8\ \ 3}$' has the same effect as '$\genfrac{}{}{0pt}{}{x}{8}$'. One idea for avoiding them is to set $\sigma\leftarrow\sigma-1$ when backtracking, instead of advancing σ twice per node by always setting $\sigma\leftarrow\sigma+1$.

a) Demolish that idea.

b) Find a correct way to do stamping that advances σ only once per node.

300. [*21*] Suppose the domain of variable v is represented by a doubly linked list as in (101) and (102), and that the dancing links protocol is being followed (so that `PREV`(a) and `NEXT`(a) don't change when a is deleted). Show that if $0\le a<a'<d$ and $a'\in D_v$, then `NEXT`$(a)\le a'$ (even when $a\notin D_v$).

301. [*18*] Explain in detail the representation of the initial domain $\{0,1,\ldots,d-1\}$, when using the (a) bit vector (b) doubly linked list (c) sparse-set representations.

302. [*21*] Sometimes a program wants to know $\min D_v$, the smallest element of v's current domain. (By convention, $\min\emptyset=d$ in this context.) What's a good way to handle that in the (a) bit vector (b) doubly linked list (c) sparse-set representations?

303. [*M11*] True or false: Equation (111) says that $(b_{\lceil d/e\rceil}\ldots b_1b_0)_{2^e}=\sum\{2^a\mid a\in D\}$.

▶ **304.** [*25*] Work out the details of the AND operation for reversible sparse bitsets: Given a set D that's represented using b, `D`, and `S` as in Fig. 118, together with another set $D'\subseteq\{0,1,\ldots,d-1\}$ that's represented as an ordinary bitset using a q-element array b', design an algorithm that sets $D\leftarrow D\cap D'$, putting appropriate entries on the trail so that this operation is reversible. Minimize the number of changes made to b, `D`, and `S`.

▶ **305.** [*27*] (*Compact-Table tuples.*) A finite k-ary relation can be defined in general by listing the k-tuples that satisfy it, as we did in (2) at the beginning of this section. Such a list, called a "table constraint," is also the domain of the hidden variable for that relation (see answer 270). One of the best ways to represent large table constraints in practice is to use reversible sparse bitsets.

Suppose $R(v_1,\ldots,v_k)$ is a relation whose variables v_j each have a d-ary domain D_j, with a sparse-set representation $\mathtt{DOM}_j$, $\mathtt{IDOM}_j$, $\mathtt{SIZE}_j$, while R itself is represented by b, `D`, and `S` as in Fig. 118. Initially there's domain consistency with respect to R: Every binding (v_j,a_j) with $a_j\in D_j$ is supported in R, meaning that some tuple of R has $v_j=a_j$; conversely, every tuple of R is *valid*, meaning that each a_j belongs to D_j.

After other constraints have been propagated, some of the domains will have changed. Explain what needs to be done in order to restore domain consistency with respect to R. *Hints:* Let $\mathtt{OSIZE}_j$ be the value of $\mathtt{SIZE}_j$ before the recent propagations. Use the intersection algorithm of exercise 304, together with appropriate bitsets b'.

▶ **306.** [*M33*] (*Backmarking.*) Suppose we are solving a CSP by assigning values to variables $x_1, x_2, \ldots$, in that order. Step t of the search process begins at level $l=l_t$, at which time we've made certain provisional assignments $x_1\leftarrow a_1, \ldots, x_l\leftarrow a_l$ and we want to select a consistent value a_{l+1} for x_{l+1}. If we succeed, this step is a "forward step," and we'll have $l_{t+1}=l+1$; otherwise it's a "backward step," and $l_{t+1}=l-1$. (Initially $l_0=0$. A backward step from level 0 terminates the search.)

octomino
chunky-octs
Aztec diamond
kingwise connected
trail
stamping
undoing
doubly linked list
dancing links
bit vector
doubly linked list
sparse-set
minimum
bit vector
doubly linked list
sparse-set
AND
reversible sparse bitsets
sparse bitsets
bitset
trail
Compact-Table
k-ary relation
table constraint
hidden variable
sparse-set representation
binding
valid
Backmarking
forward step
backward step

nested parentheses
average
ACTIVE
minimum remaining values
MRV
hide
sharp preference

After the first backward step from level l, subsequent steps at that level tend to repeat much of the previous computations. Indeed, there's a value $s = s_t \le l$ for which the previous backward step at level l dealt with exactly the same assignments $x_j \leftarrow a_j$ for $1 \le j < s$. Thus we already "know" the results of all tests on s-ary relations between $a_1, \ldots, a_{s-1}$ and a_{t+1}, and we could have saved that information in an auxiliary array.

a) Forward and backward steps can be represented by a sequence of nested parentheses as in 7.2.1.6–(1). What values of l_t and s_t for $0 \le t < 30$ correspond to the sequence '`(())((())((()(()))())(()(())))`'? (Use $s = 0$ before backward steps.)

b) Devise a way to calculate $s_0, s_1, \ldots,$ from a given level sequence $l_0, l_1, \ldots.$ *Hint:* Maintain a sequence of intervals $[p_0 \,..\, q_0], [p_1 \,..\, q_1], \ldots, [p_r \,..\, q_r]$, where $0 = p_0 < p_1 < \cdots < p_r$, such that $s_t = p_k$ when k is maximum with $p_k \le l_t \le q_k$.

c) Show that the s values can indeed be rather complicated, by constructing a level sequence $l_0, l_1, \ldots$ for which the intervals in the preceding hint are

$$[0 \,..\, \infty],\ [2 \,..\, 8],\ [4 \,..\, 6],\ [5 \,..\, 5],\ [10 \,..\, 15],\ [11 \,..\, 12],\ [14 \,..\, 14].$$

d) Find levels $0 = l_0, l_1, \ldots, l_{30} = 0$ for which $s_0 + s_1 + \cdots + s_{29} \ge 107$.

e) What's the average of $s_0 + \cdots + s_{29}$ over *all* level sequences $0 = l_0, \ldots, l_{30} = 0$?

f) The amount of nonrepeated computation at step t can be measured by $l_t - s_t$. Generate random sequences of nested parentheses, 1000000 of each, and estimate the average value of $l_t - s_t$ for $0 \le t < 2000000$. *Hint:* See Algorithm 7.2.1.6W.

g) Let $D_j = \{1, \ldots, d_j\}$ be x_j's domain. Explain how to use s_t to avoid recomputation at step t, by maintaining a "mark" M_{ja} for each variable x_j and each $a \in D_j$.

307. [*05*] Explain the significance of `CLR`(x) in Table 3.

308. [*10*] What node in Table 7.2.2.1–2 corresponds to node x in Table 3, for $0 \le x \le 19$?

309. [*20*] True or false: `ITM`$(x) <$ `SECOND` if and only if `LOC`$(x) <$ `SECOND`.

▸ **310.** [*20*] True or false: `ACTIVE` $= 0$ whenever Algorithm C finds a solution in step C9.

311. [*25*] Design an algorithm to set up the initial memory contents of an XCC problem, as needed by step C1 of Algorithm C and illustrated in Table 3. The input to your algorithm should consist of a sequence of lines with the following format:

- The very first line lists the names of all items, with the primary items first.
- Each remaining line specifies the items of a particular option, one option per line.

312. [*18*] Explain how to branch in step C2 on an item i for which `SIZE`(i) is minimum. If several items have that minimum length, i itself should also be minimum. (This choice is often called the "minimum remaining values" (MRV) heuristic.)

313. [*20*] In Table 3, find i and c such that hide(i, c) will set `FLAG` $\leftarrow 1$ if `FLAG` $= 0$.

▸ **314.** [*19*] Play through Algorithm C by hand, using exercise 312 in step C2 and the input in Table 3, until first reaching step C8. What will the memory contain at that time?

▸ **315.** [*21*] Why would it be a mistake to omit '`FLAG` $\leftarrow -1$' in step C4?

316. [*21*] In some applications the MRV heuristic of exercise 312 leads the search astray, because certain primary items have short lists yet convey little information about desirable choices. Modify answer 312 so that an item i whose name does not begin with the character '#' will be chosen only if `SIZE`$(i) = 1$ or no other choices exist. (This tactic is called the "sharp preference" heuristic.)

317. [*22*] Why doesn't step C7 hide i' when $i' \ge$ `SECOND` and `POS`$(i') \ge$ `OACTIVE`?

▶ **318.** [*33*] (C. Solnon.) Upgrade Algorithm C to Algorithm C^+ by treating cases with `SIZE`$(i) = 1$ more efficiently. *Hints:* Maintain a list of all such active primary items. Step C5 is unnecessary when `SIZE`$(i) = 1$, because step C11 will always go to C10.

▶ **319.** [*20*] Step C3 of Algorithm C might find $i' = i$, in which case the last five assignments can be skipped. Explain why it's probably *not* a good idea to skip them.

▶ **320.** [*18*] Suppose the item i that's chosen by the MRV heuristic in step C2 has options $o_1, \ldots, o_d$, where $d =$ `SIZE`$(i) > 1$. Show that, after we've considered all solutions in which i is covered by o_1, the MRV heuristic will tell us to branch again on this very same item i, as we explore the solutions to the remaining problem.

321. [*10*] Does the search tree (121) contain a node at stage 1 and level 3?

▶ **322.** [*33*] Design Algorithm B, which should be like Algorithm C^+ except that it does binary branching instead of d-way branching. Your algorithm should use a user-supplied heuristic function $h(i)$ for dynamic variable ordering, as described in the text.

323. [*20*] When the MRV heuristic function $h(i) =$ `SIZE`(i) is used in Algorithm B, the running time doesn't actually match the speed of Algorithm C^+. For example, Problem C needs 45.0 Gμ, not 41.6 Gμ. Why?

324. [*20*] Modify Algorithm B so that it incorporates the WTD heuristic, (122).

325. [*20*] Formulate the queens-and-knights problem as an XCC problem.

326. [*22*] Sketch the overall behavior of Algorithm B when it solves the queens-and-knights problem with the WTD heuristic. How large do the weights become?

327. [*20*] Compare WTD to MRV on the queens-and-knights problem when there are (a) 8 queens, 3 knights; (b) 8 queens, 7 knights; (c) 12 queens, 5 knights.

328. [*33*] The queens-and-knights problem is an example where WTD is exponentially better than MRV. Construct XCC problems for which WTD is exponentially worse.

329. [*18*] Modify Algorithm B so that it incorporates the FRB heuristic, (124).

330. [*20*] Do exercise 327, but with FRB instead of WTD.

▶ **331.** [*25*] Modify Algorithm C^+ (exercise 318) so that it can be used with heuristics such as WTD and FRB to do d-way branching instead of binary branching.

332. [*24*] Do the WTD and/or FRB versions of exercise 331 improve on (125)?

333. [*16*] The matrix (126) is only one of several almost-support matrices that can be constructed for the options $\{00, 05, 10, 13, 16, 19\}$. What are the other possibilities?

▶ **334.** [*22*] When setting up a support matrix, it's desirable to have a fast way to test whether or not a particular option o is compatible with at least one option o' that contains a given item i, where $i \notin o$. Design an algorithm to do this. *Hint:* Allocate a new 32-bit field `MARK`(i) in the `SET` array, for every item i, and use "stamping."

335. [*17*] True or false re (128): The items of every option in O_s belong to I_s.

▶ **336.** [*13*] Consider the example XCC problem of (126), after Algorithm S has explored all solutions with option 13 and has then backtracked to stage 0 and removed 13. What are O_{-1}, O_0^{init}, and O_0 at that time? What are the ages of the inactive options?

337. [*20*] Why is it better for the set Q to be a queue (FIFO) than a stack (LIFO)?

338. [*25*] Is it possible to call opt_out(o) at a time when $o \in Q$?

Solnon
forced moves
MRV heuristic
d-way branching
search tree
stage
level
binary branching
heuristic function $h(i)$
dynamic variable ordering
variable ordering
MRV heuristic function
heuristic function
queens-and-knights problem
XCC problem
WTD
MRV
FRB
d-way branching
binary branching
support matrices
compatible
stamping
queue (FIFO) than a stack (LIFO)
opt_out(o)

339. [*35*] The low-level data structures used by Algorithm S extend those of Algorithm C by giving each node a fourth field, XTRA, in addition to the fields ITM, LOC, CLR that are illustrated in Table 3. We use the abbreviations $\texttt{TRIG}(o) = \texttt{CLR}(o)$, $\texttt{FIX}(o) = \texttt{XTRA}(o)$, and $\texttt{AGE}(o) = \texttt{XTRA}(o+1)$, when o is the spacer node preceding an option. The first item of an option, $\texttt{ITM}(o+1)$, is required to be primary.

The trigger and fixit stacks are implemented with classical singly linked list structures, using an array called POOL whose elements have two fields, INFO and LINK. The triggers of (127) could, for example, be represented with $\texttt{TRIG}(0) = 1$, $\texttt{TRIG}(5) = 3$, $\texttt{TRIG}(13) = 7$, $\texttt{TRIG}(19) = 11$, and the following POOL:

p:	1	2	3	4	5	6	7	8	9	10	11	12	13
$\texttt{INFO}(p)$:	19	4	13	4	13	17	5	11	19	11	0	17	—
$\texttt{LINK}(p)$:	2	0	4	5	6	0	8	9	10	0	12	0	—

Two pointers, QF and QR, define the queue Q, which is empty if and only if $\texttt{QF} = \texttt{QR}$. (The POOL above can accompanied by $\texttt{QF} = \texttt{QR} = 13$.) The insertion operation '$o \Rightarrow Q$' means "$\texttt{INFO(QR)} \leftarrow o$, $\texttt{LINK(QR)} \Leftarrow \texttt{AVAIL}$, $\texttt{QR} \leftarrow \texttt{LINK(QR)}$"; and the deletion operation '$Q \Rightarrow o$' means "$p \leftarrow \texttt{QF}$, $o \leftarrow \texttt{INFO}(p)$, $\texttt{QF} \leftarrow \texttt{LINK}(p)$, $p \Rightarrow \texttt{AVAIL}$," if $\texttt{QF} \neq \texttt{QR}$.

For example, starting with (127) represented as above, and with $\texttt{FIX}(0) = \cdots = \texttt{FIX}(19) = 0$, a call on opt_out(13) would have the effect of setting $\texttt{SIZE}(11) \leftarrow 1$, $\texttt{SIZE}(23) \leftarrow 2$, $\texttt{FIX}(5) \leftarrow 7$, $\texttt{FIX}(19) \leftarrow 9$, $\texttt{LINK}(8) \leftarrow 0$, $\texttt{INFO}(7) \leftarrow 13$, $\texttt{INFO}(9) \leftarrow 13$, $\texttt{TRIG}(13) \leftarrow 0$, $\texttt{INFO}(13) \leftarrow 5$, $\texttt{LINK}(13) \leftarrow 14$, $\texttt{INFO}(14) \leftarrow 19$, $\texttt{LINK}(14) \leftarrow 15$, $\texttt{QR} \leftarrow 15$.

Use these conventions to design Algorithm O, a "naïve" implementation of the 'opt_out' subroutine described in the text.

340. [*32*] Design Algorithm E, the 'empty_q' subroutine that's described in the text.

341. [*30*] Implement the portion of step S1 that establishes initial domain consistency.

342. [*20*] Explain how to delete all references to purged options from the trigger stacks of unpurged options, after the algorithm of exercise 341 has acted.

343. [*22*] The support-finding loop in the answer to exercise 334 runs through the active options that contain a given item i sequentially, from first to last. Are better results obtained by considering them (a) backwards (last to first)? (b) randomly?

▸ **344.** [*37*] When Algorithm O (exercise 339) deactivates option o, it looks at every entry (o', i') of o's trigger stack. If o' and i' are both active, it converts that entry to a fixit (o, i') on the trigger stack of o'; otherwise (o', i') remains on the trigger stack of o. We could save a lot of time if the trigger stack had the property that its entries for inactive o' all appeared at the bottom; then we wouldn't have to look at them all individually.

a) Explain why we can't hope to keep the trigger stacks sorted by age, with entries for the earliest-deactivated options o' nearest the bottom.

b) However, suggest a refined method, Algorithm O$^+$, that does tend to cluster the inactive entries near the bottom, and avoids looking at them all. *Hint:* Sort the entries (o', i') that remain in $\texttt{TRIG}(o)$, after opt_out(o) has acted, by $\texttt{AGE}(o')$.

345. [*20*] Demonstrate the importance of trigger hints empirically, by running Algorithm S on the "extreme" XC problem for $n = 12$ (7.2.2.1–(82)), with and without them.

346. [*22*] Step S2 of Algorithm S advances SSTAMP, a 32-bit number whose values go into the SS array that's used in the "hints" of exercise 344. Ordinarily we can just set $\texttt{SSTAMP} \leftarrow (\texttt{SSTAMP} + 1) \bmod 2^{32}$; but trouble will arise when the result is zero. Explain how to avoid trouble. (See exercise 334 for the solution to a similar problem.)

data structures
spacer node
first item of an option
primary
trigger
fixit
singly linked list
queue
empty
AVAIL list
opt_out
empty_q
domain consistency
trigger stack
"extreme" XC problem
stamping
overflow

▶ **347.** [*30*] Spell out the low-level details of what happens when step S6 of Algorithm S chooses an option $c_{s+1} = x_l$ to explore at the next stage of the search.

348. [*22*] Given m, n, i, j, and a set P, where $0 \le i < m$, $0 \le j < n$, and $P \subseteq \{1, 2, \ldots, mn\}$, construct an XCC problem whose solutions assign labels $\{1, 2, \ldots, mn\}$ to the cells of an $m \times n$ board, where the labels define steps 1, 2, ..., mn of a closed knight's tour (a Hamiltonian cycle of the $m \times n$ knight graph). Furthermore, if a queen is placed in cell (i, j), that queen must attack every cell whose label is in P.

▶ **349.** [*23*] (Peter Weigel, 2023.) Improve the construction of exercise 348 by having one option for every potential *pair* of knight moves, to and from a white cell, instead of having one option for every potential *single* move.

350. [*20*] Thanks to the construction of exercise 349, the author was able to celebrate his 85th birthday in 2023 with a felicitous closed solution to the 10×10 prime queen attacking problem: It featured the special pattern '$\begin{smallmatrix} 85 & 00 \\ 20 & 23 \end{smallmatrix}$' in the center, surmounted by the exact date of his birth, '01 10 19 38'! How many such solutions exist?

351. [*24*] The *strong prime queen attacking problem* is the special case of exercise 348 where P consists of all prime numbers $\le mn$ plus all numbers 2^e for $0 \le e \le \lg mn$.

a) Exhibit solutions of this problem, for as many $m \le n$ as you can.

b) Also count the total number of solutions, for as many $m \le n$ as you can.

▶ **352.** [*41*] (Filip Stappers, 2023.) Design Algorithm F, an MCC solver that accepts the same input as Algorithm 7.2.2.1M but uses dancing cells instead of dancing links. *Hint:* Modify Algorithm B (exercise 322).

353. [*22*] How can Algorithm F use dynamic heuristics such as WTD and FRB?

354. [*M21*] (*Covering with disks.*) Can an $m \times n$ rectangle be covered with k "discrete disks" of integer diameter d? (Namely the set of pixels of a $d \times d$ square whose centers are at distance $< d/2$ from the center of that square.) Formulate this as an MCC problem.

355. [*16*] Exercise 7.2.2.1–266 explains how to generate the options for an exact cover problem whose solutions are the ways to pack a given shape with a given set of polyominoes. What happens if we use those options in an MCC problem instead of in an XC problem, assigning the multiplicity $[u_{xy} \mathinner{..} v_{xy}]$ to each cell (x, y) of the shape and the multiplicity $[u_p \mathinner{..} v_p]$ to each piece p?

▶ **356.** [*22*] Exercise 216 explains how to encode "negative table constraints" as constraints of an XC problem, provided that each constraint is *binary*. Show that negative table constraints between $k > 2$ variables can be encoded as constraints of an MCC problem. For example, how could you encode '$x \ne a$ or $y \ne b$ or $z \ne c$'?

357. [*M21*] Use Algorithm F to find all solutions to the n queens problem such that no three queens lie in a straight line of any slope.

358. [*M20*] According to (125), Algorithm C$^+$ finds the 15 million solutions to the 16 queens problem in 43.9 Gμ. According to (134), Algorithm F finds the 71 thousand for which no three are collinear in 87.4 Gμ.

So why not simply remove unwanted solutions from the output of Algorithm C$^+$?

359. [*M21*] Find the maximum m such that m distinct straight lines each contain three or more queens, in some solution to the 16 queens problem.

360. [*21*] In how many ways can q queens and s knights be placed on an $m \times n$ board so that no two pieces attack each other? Formulate this as an MCC problem.

closed knight's tour
knight's tour
Hamiltonian cycle
knight graph
prime queen attacking problem+
queen
Weigel
author
birthday
prime queen attacking problem
strong prime queen attacking
prime queen attacking
Stappers
MCC solver
dancing cells
dancing links
Covering with disks
k-center problem
discrete disks
pixels
MCC problem
exact cover problem
pack a given shape
polyominoes
negative table constraints
n queens problem
no-three-in-line
16 queens problem
queens and knights, nonattacking
knights and queens, nonattacking
MCC problem

361. [*20*] For $1 \le q \le 7$, find the maximum number of knights that can be placed together with q queens on an 8×8 chessboard so that no piece attacks another. How does Algorithm F fare, in comparison to Algorithm 7.2.2.1M, when the options of exercise 360 are used to solve this problem?

362. [*21*] (Ian Tullis, 2022.) If possible, create a 4-colored 10×10 pattern in which, for $1 \le c \le 4$, (i) every row contains exactly c cells of color c; (ii) every column contains exactly c cells of color c; (iii) the rookwise-connected components of color c have exactly c cells; and (iv) every component of color 4 is an ell tetromino.

363. [*22*] (*Shifting.*) Let S be the following binary relation between k-tuples:

$$S(x_1x_2\ldots x_k, y_1y_2\ldots y_k) \iff x_1 = y_2 \text{ and } x_2 = y_3 \text{ and } \cdots \text{ and } x_{k-1} = y_k.$$

For example, if $k = 2$ and $x = x_1x_2$ and $y = y_1y_2$, then $x_1 = y_1$ if and only if there exists a 2-tuple $z = z_1z_2$ such that $S(x, z)$ and $S(y, z)$ both hold.

a) When $k = 3$ and $x = x_1x_2x_3$, express the condition $x_1 = x_3$ in terms of S.
b) When $x = x_1x_2x_3$ and $y = y_1y_2y_3$, express the condition $x_1 = y_3$ in terms of S.
c) When $x = x_1 \ldots x_k$, $y = y_1 \ldots y_k$, $1 \le i \le j \le k$, express the condition $x_i = y_j$.
d) When $k = 5$ and $x = x_1x_2x_3x_4x_5$, express '$x_1 = x_4 = x_5$ and $x_2 = x_3$'.

364. [*23*] Show that any CSP $\mathcal{P}$ can be transformed into an equivalent CSP $\mathcal{P}^*$, in which all but one of the constraints are applications of the shift relation of exercise 363; the sole exception is a single unary constraint. (Problem $\mathcal{P}^*$ is satisfiable if and only if $\mathcal{P}$ is satisfiable, although $\mathcal{P}^*$ might have many more solutions.) Illustrate your general construction when $\mathcal{P}$ is the CSP of (1) and (2). [*Hint:* Ignore efficiency.]

▸ **365.** [*M26*] Given a binary string $a_1 \ldots a_r$, let R_α be the relation $\{(x-1, x) \mid a_x = 1\} \cup \{(x, x-1) \mid a_x = 0\}$ on the domain $[0\,.\,.\,r]$. Also let $R_{it} = R_{(110)^i 1 (110)^{t-1-i} 11}$, for $0 \le i < t$. For example, $R_{13}(x, y) =$ '$xy \in \{01, 12, 32, 34, 45, 56, 76, 78, 89\}$'.

a) True or false: The relation R_α is min-closed and max-closed.
b) Prove that there's no homomorphism from R_{it} to R_{jt} when $i \ne j$ (see (4)).
c) Given $0 \le i < t$, find a homomorphism from $R_{(110)^t 11}$ to R_{it}.

366. [*20*] Suppose a CSP contains a variable v that is related to only one other variable, w. Explain how to simplify the CSP without losing any of its solutions.

367. [*17*] If $|D_v| = |D_w| = 2$, show that *every* relation between v and w is implicational. What are the *non*-implicational relations when $D_v = \{0, 1\}$ and $D_w = \{0, 1, 2\}$?

368. [*20*] True or false: The intersection of implicational constraints is implicational.

369. [*16*] Explain (135); how were these reduced domains computed?

370. [*M27*] Complete the proof of Theorem I by showing that (a) while traversing $\widehat{G}$, if a value has previously been assigned to any variable v_j of a sink component $\{(v_1, a_1), \ldots, (v_t, a_t)\}$, then *all* variables v_i of that component have already received values; (b) after we've assigned values to variables v and w, the constraint between them is satisfied; and (c) if $\mathcal{P}$ has a solution, the traversal process will find one.

▸ **371.** [*20*] One common way to represent a set of numbers is to use a singly linked list. Each node has two fields, `INFO` and `LINK`. The set $X = \{x_1, \ldots, x_t\}$ with $x_1 < \cdots < x_t$ is represented in $t + 1$ nodes $\{X, p_1, \ldots, p_t\}$, with

$$\texttt{LINK}(X) = p_1; \quad \texttt{INFO}(p_j) = x_j, \ \texttt{LINK}(p_j) = p_{j+1}, \text{for } 1 \le j \le t;$$

X is the list head, and $p_{t+1} = \perp$ is the address of a special node with $\texttt{INFO}(\perp) = \infty$.

If m sets $X_1, \ldots, X_m$ are represented in this way, explain how to compute their intersection $Y = X_1 \cap \cdots \cap X_m$, in $O(m + |X_1| + \cdots + |X_m|)$ steps.

chessboard
dancing cells versus dancing links
Tullis
polyominoes
ell tetromino
tetromino
Shifting
k-tuples
tuples
efficiency
min-closed
max-closed
homomorphism
implicational relations
domains
sink
represent a set
singly linked list
list head
intersection

372. [*16*] Change one 0 to a 1 so that both of these matrices represent max-closed relations:

$$\begin{pmatrix} 0&0&0&0&0&1&0&0&0\\ 0&0&1&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&1&0\\ 1&0&1&1&0&1&0&1&0 \end{pmatrix} \qquad \begin{pmatrix} 1&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&1&0&1\\ 0&1&0&1&0&0&0&0&1\\ 0&0&0&0&0&1&1&0&1\\ 0&0&0&0&0&0&0&0&0 \end{pmatrix}$$

373. [*18*] Are (a) '$x^2 \geq 4yz$' and (b) '$x^2 \leq 4yz$' max-closed on nonnegative integers?

▸ **374.** [*M20*] Prove that (142) is max-closed because (141) is max-closed.

375. [*M21*] Show that max-closed relations on a two-element domain are equivalent to Horn clauses.

▸ **376.** [*20*] Why does $v \leftarrow v_{\max}$ solve a max-closed CSP after (143) has done its work?

377. [*M21*] A *semilattice* is an algebraic system that has a binary operator that's idempotent, commutative, and associative. In other words, there's an operation $x \vee y$ that satisfies three axioms: $x \vee x = x$; $x \vee y = y \vee x$; $(x \vee y) \vee z = x \vee (y \vee z)$.

a) Prove that the relation '$x \preceq y$' defined by the condition $x \vee y = y$ is a *partial ordering*. (In other words, it is reflexive, antisymmetric, and transitive).

b) In that partial ordering, $x \vee y$ is the least upper bound of x and y.

c) Suppose a partial ordering '$\preceq$' relation has the property that every two elements $\{x, y\}$ have a least upper bound, $x \vee y$. Does the $\vee$ operator define a semilattice?

378. [*M22*] Let $\mathcal{P}$ be a CSP in which all variables have domain D and all constraints are closed under $\vee$, where $\vee$ is a semilattice operation on D. Prove that, after Algorithm D has achieved domain consistency, a solution is obtained by setting each variable v to $\bigvee\{a \mid a \in D_v\}$, unless some $D_v = \emptyset$.

379. [*M20*] Exercise 270 showed that every CSP $\mathcal{P}$ can be converted to a CSP $\mathcal{P}^*$ that has only binary constraints, using "hidden variables." Show that if $\mathcal{P}$ is max-closed, then $\mathcal{P}^*$ is $\vee$-closed, for some semilattice operation $\vee$.

380. [*20*] Explain how to find *all* solutions of a CSP to which Lemma R applies.

381. [*M24*] Prove that a binary relation R is CRC if and only if it is *median*-closed: $a_1b_1 \in R$, $a_2b_2 \in R$, and $a_3b_3 \in R$ implies that $\langle a_1a_2a_3\rangle\langle b_1b_2b_3\rangle \in R$.

382. [*M22*] Prove Theorem R. *Hint:* Use exercise 381.

▸ **383.** [*40*] When all constraints of a CSP are CRC, and we seek only a single solution, it may be wise to avoid the expense of path consistency checking until it proves to be really necessary, because a solution might be obtained quite quickly. Thus we could decide to be "lazy" about removing inconsistent elements of relations. If we do run into a dead end, we can then exploit the fact that the fully path consistent constraints will necessarily be row convex. Experiment with this idea.

384. [*HM21*] Say that a matrix is *reduced* if it has no all-zero rows and no all-zero columns. (Thus, if we represent a binary relation between variables x and y by a matrix with one row for every element in the domain of x and one column for every element in the domain of y, the matrices of a binary CSP are all reduced if and only if the CSP is domain consistent.)

Given a set $\mathcal{A}$ of reduced matrices, let $\mathcal{A}^+$ be the set of all matrices that reduce to a matrix of $\mathcal{A}$ when every all-zero row and every all-zero column has been deleted.

max-closed
Horn clauses
semilattice
idempotent
commutative
associative
axioms
partial ordering
reflexive
antisymmetric
transitive
least upper bound
domain consistency
binary constraints
hidden variables
CRC
median-closed
median
path consistency checking
lazy
reduced matrix
all-zero rows
matrix representation of binary relation
domain consistent

If A_{mn} is the number of $m \times n$ matrices in $\mathcal{A}$, and A^+_{mn} is the corresponding number for $\mathcal{A}^+$, explain how to compute the values A^+_{mn} from the values A_{mn} and vice versa.

▶ **385.** [*HM22*] Let P^+_{mn} be the number of $m \times n$ matrices for *implicational* relations, as defined in the text. Compute P_{mn} and P^+_{mn} for $1 \le m, n \le 6$. (See exercise 384.) Also explain how the calculation can be done efficiently for large m and n.

▶ **386.** [*HM30*] Continuing exercise 385, let Q^+_{mn} be the number of $m \times n$ matrices for *max-closed* relations, as defined in the text. Find generating functions for Q_{mn} and Q^+_{mn}, and exhibit the results for $1 \le m, n \le 6$.

▶ **387.** [*M26*] Continuing exercise 386, let R^+_{mn} be the number of $m \times n$ matrices for *connected row convex* relations, as defined in the text. Find recurrence relations by which R_{mn} can be computed efficiently, and exhibit the results for $1 \le m, n \le 6$.

▶ **388.** [*M20*] Continuing exercise 387, let S^+_{mn} be the number of $m \times n$ matrices for relations that are both *min-closed* and *max-closed*. Find recurrence relations by which S_{mn} can be computed efficiently, and exhibit the results for $1 \le m, n \le 6$.

389. [*24*] Confirm the values of P_{33}, Q_{33}, R_{33}, and S_{33} in the preceding exercises by explicitly listing all of the reduced 3×3 matrices for relations that are (a) implicational; (b) max-closed; (c) connected row convex; (d) min-and-max-closed.

390. [*22*] Compute the number Q^+_{lmn} of *ternary* max-closed relations between variables whose domain sizes are respectively l, m, and n, for $2 \le l \le m \le n \le 4$.

▶ **391.** [*M25*] Find a function $\Delta(x, y, z)$ such that a binary relation $R(x, y)$ is implicational if and only if Δ is a polymorphism of R.

392. [*HM26*] Let $\mu_d(x, y, z) = (x - y + z) \bmod d$. Show that an m-ary constraint is a conjunction of relations having the form $(a_1x_1 + \cdots + a_mx_m) \bmod d = b$, for integer constants $\{a_1, \ldots, a_m, b\}$, if and only if that constraint is closed under μ_d. (Thus, by Theorem D, a CSP for which μ_d is a polymorphism is solvable in polynomial time.)

393. [*M25*] Consider the k-ary polymorphisms of $\{01, 10, 11\}$ (OR) when $k \ge 1$.

a) The CNF formula (146), with six clauses, characterizes those polymorphisms when $k = 3$. How many clauses are in the analogous formula for general k?

b) How do the polymorphisms of $\{01, 10, 11\}$ relate to those of $\{00, 01, 10\}$ (NAND)?

c) Find a connection with families of mutually intersecting sets.

▶ **394.** [*M28*] Investigate the polymorphisms of R, when R is any one of the 16 Boolean binary relations. How many k-ary polymorphisms does R have, for $1 \le k \le 5$?

395. [*M21*] Find (by hand) the two solutions of (145) that don't solve (148).

396. [*M17*] Let $\mathcal{P}$ be a CSP with variables $\{v_1, \ldots, v_n\}$. The set of all n-tuples $a_1 \ldots a_n$ such that $\{v_1 = a_1, \ldots, v_n = a_n\}$ solves $\mathcal{P}$ is a n-ary relation, $R(\mathcal{P})$. True or false: If f is a polymorphism of every constraint of $\mathcal{P}$, then f is also a polymorphism of $R(\mathcal{P})$.

▶ **397.** [*22*] Discuss the indicator problem for the example CSP defined by (1) and (2), with which we began this section. (Re-encode the domains and relations so that $D_1 = D_2 = D_4 = D_5 = \{0, 1\}$ and $D_3 = \{0, 1, 2\}$; $R_1 = \{001, 020, 111\}$; $R_2 = \{00, 10, 11\}$; $R_3 = \{010, 011, 100\}$. Use the domain $D = \{0, 1, 2\}$; thus $D_1 = \{0, 1\}$ is a unary relation, and $D_2 = D_4 = D_5 = D_1$.) What are the ternary polymorphisms?

▶ **398.** [*21*] Formulate the indicator problem $\mathcal{I}_k(\Gamma)$ as an XCC problem.

▶ **399.** [*M25*] Show that the symmetric Boolean relation $R = S_{0,2,3} = \{000, 011, 101, 110, 111\}$ ('$x + y + z \ne 1$') has exactly $2^k + 1$ k-ary polymorphisms, for all $k \ge 1$.

implicational
max-closed
generating functions
connected row convex
CRC relations
recurrence relations
min-closed
recurrence relations
implicational
max-closed
min-and-max-closed
ternary
implicational
polymorphism
Maltsev term
OR
NAND
mutually intersecting sets
intersecting sets
Boolean binary relations
binary relations
solution set
indicator problem
unary relation
XCC problem

self-dual
unit vector
nonequality
Cooper
not all equal
gadget
pi, as source
relations
oriented d-cycle
shortcut
gadget
closed set
polarities

400. [*M22*] Show that the symmetric Boolean relation $S = S_1 = \overline{R} = \{001, 010, 100\}$ ('$x+y+z = 1$') has exactly k k-ary polymorphisms, for all $k \geq 1$. *Hint:* Show that each polymorphism is self-dual, and has $[e_j] = 1$ for exactly one unit vector $e_j = 0^{j-1}10^{k-j}$.

401. [*M23*] Find, by hand, all polymorphisms of the relation '$x \neq y$' on $\{0, 1, 2\}$.

▶ **402.** [*23*] What's the "essence" of each of the 125 constraints of $\mathcal{I}_3(\{`x+y+z\neq 1\text{'}\})$?

403. [*21*] Columns [011][100][101] of (152) obey a curious relation $Q = \{000, 011, 100, 101, 111\}$. Use an indicator problem to construct a gadget for Q analogous to (153).

404. [*22*] Construct R from $\widehat{R}$ by adapting exercise 402 to $\mathcal{I}_3(\{\widehat{R}\})$.

405. [*22*] Construct T from R by adapting exercise 402 to $\mathcal{I}_4(\{R\})$.

406. [*25*] (Martin C. Cooper, 2024.) Find a simpler gadget than (156): Express $T(x, y, z)$ in terms of just two R constraints and one auxiliary variable.

407. [*M22*] Let $S^{(m)} = S^{(m)}(x_1, \ldots, x_m)$ be the Boolean m-ary relation '$x_1 + \cdots + x_m = 1$'. Find a gadget that expresses $S^{(3)}$ in terms of $S^{(m)}$, when $m > 3$.

408. [*M22*] Let $S = S(w, x, y, z)$ be the Boolean quaternary relation '$w+x+y+z = 2$', and let $N = N(x, y, z)$ be the Boolean ternary relation 'x, y, and z are not all equal'. Find gadgets that express (a) N in terms of S; (b) S in terms of N.

▶ **409.** [*M25*] Let R be a relation with k m-tuples $\{a_{11} \ldots a_{1m}, \ldots, a_{k1} \ldots a_{km}\}$, and let Γ be a family of relations. Show that there's a gadget for R using only the relations in Γ if and only if we have $[a_{11} \ldots a_{k1}] \ldots [a_{1m} \ldots a_{km}] \in R$ in every solution of $\mathcal{I}_k(\Gamma)$.

410. [*23*] Consider five more-or-less "random" relations on the domain $\{0, 1, 2, 3, 4\}$:

$$R_1 = \{303, 232, 214\}; \quad R_2 = \{303, 343, 241\}; \quad R_3 = \{124, 122, 404\};$$
$$R_4 = \{140, 231, 421\}; \quad R_5 = \{114, 302, 031\}.$$

(They were inspired by the fact that $\pi = (3.03232214303343241124122404140 23\ldots)_5$.)

There are $\binom{25}{4} = 12650$ binary relations on this domain that are satisfied by exactly four pairs. How many of them can be expressed as gadgets, in terms of R_1 through R_5? (According to exercise 409, $\mathcal{I}_4(\{R_1, \ldots, R_5\})$ determines the answer.)

▶ **411.** [*M26*] Describe and count the k-ary polymorphisms of the following relations on $\{0, 1, 2\}$: $R_0 = \{001, 010, 200\}$; $R_1 = R_0 \cup \{100\}$; $R_2 = R_1 \cup \{020\}$; $R_3 = R_2 \cup \{002\}$.

▶ **412.** [*M30*] Relation (159) is just the smallest example Θ_3 of an infinite family of binary "shortcut" relations Θ_d for all $d \geq 3$, defined as follows: We start with d arcs $0 \longrightarrow 1 \longrightarrow 2 \longrightarrow \cdots \longrightarrow d{-}1 \longrightarrow 0$, which give us the oriented d-cycle $\vec{C_d}$; then we add a "shortcut" arc $0 \longrightarrow 2$, so that we also have an oriented $(d{-}1)$-cycle.

a) Prove that Θ_d has only one unary polymorphism.
b) Construct a gadget for the relation $\vec{C_d}$ from Θ_d.
c) Use Θ_d to make a gadget for some relation R that contains exactly $d^2 - 1$ pairs.
d) If Φ_d is *any* binary relation on $\{0, 1, \ldots, d{-}1\}$ that satisfies the conditions proved for Θ_d in (a), (b), and (c), prove that Φ_d is potent.

413. [*M26*] Can a binary relation on $\{0, 1, \ldots, d-1\}$ with at most d pairs be potent?

414. [*M20*] Re (171) and (172), show that $\overline{X}$ is the *smallest* closed set containing X.

415. [*18*] How would the polarities and closed sets of the text's 3×5 example change if we added a new row to R that is (a) all 0s? (b) all 1s?

▶ **416.** [*M22*] Suppose $A' \subseteq A$, $B' \subseteq B$, and $R' \subseteq R$. Compare the sets that are closed with respect to (A', B', R') to the sets that are closed with respect to (A, B, R).

▸ **417.** [*HM30*] Identify the closed subsets of A and B in the following cases of polarity:

a) $A = B =$ all positive integers, $x\,R\,y =$ 'x divides y'.

b) $A = B =$ all positive integers, $x\,R\,y =$ 'x^2 divides y^3'.

c) $A = B =$ all 2×2 matrices of integers, $x\,R\,y =$ '$xy = yx$'.

d) $A = B =$ all rational numbers, $x\,R\,y =$ '$x \le y$'.

e) $A = B =$ all rational numbers, $x\,R\,y =$ '$x < y$'.

f) $A =$ all real numbers, $B =$ all polynomials over the integers, $x\,R\,y =$ '$y(x) = 0$'.

418. [*HM25*] (Évariste Galois, 1832.) Let $p(x) = x^n + a_1x^{n-1} + \cdots + a_n$ have integer coefficients, and let its (complex) roots be $z_1, \ldots, z_n$. Let A be the smallest field that contains all rational numbers and all the roots $\{z_1, \ldots, z_n\}$. Let B be the group of automorphisms of that field. And let $x\,R\,y =$ 'y fixes the point x'. Prove that the closed subsets of A are the subfields of A, and the closed subsets of B are the subgroups of B.

419. [*M21*] If $\mathcal{C}$ is a clone of operations on a finite set, its "spectrum" is the sequence $f_0, f_1, \ldots$, where f_k is the number of k-ary operations in $\mathcal{C}$. The "essential spectrum" $e_0, e_1, \ldots$ is similar, but e_k is the number of k-ary operations that are *full*. Here $e_0 = f_0$ is the number of constant functions among the 1-ary operations.

a) Show that $f_n = \sum_k \binom{n}{k} e_k$ and $e_n = \sum_k (-1)^{n-k}\binom{n}{k} f_k$.

b) What's the spectrum of the *smallest* clone of operations on a set of size $d \ge 1$?

c) How many Boolean operations $f(x_1, x_2, x_3, x_4)$ are full?

420. [*M33*] (*Willard pairing.*) If $f(x_1, \ldots, x_k)$ is a k-ary function and $1 \le i, j \le k$, $i \ne j$, let $f_{ij}(x_1, \ldots, x_k)$ be the subfunction obtained by setting $x_i \leftarrow x_j$. For example, if $k = 4$, $f_{24}(x_1, x_2, x_3, x_4) = f(x_1, x_4, x_3, x_4)$. This exercise will prove the following "pairing lemma": *Every full k-ary operator f on the domain $\{0, 1, \ldots, d-1\}$, where $d < k$, has a subfunction f_{ij} whose essential arity is at least $k - 2$.*

a) Show that the assumption $d < k$ is necessary when $k > 2$.

b) True or false: If f_{ij} depends on x_j, then f depends on x_i or x_j (or both); if f_{ij} depends on x_l where $i \ne l \ne j$, then f depends on x_l.

c) Let $r = \max_{1 \le i < j \le k} \operatorname{ea}(f_{ij})$. If the lemma is false, show that at least one subfunction f_{ij} with $\operatorname{ea}(f_{ij}) = r$ is independent of x_j.

d) Now assume that $f(x_1, \ldots, x_{k-2}, x, x) = h(x_1, \ldots, x_r)$, where both f and h are full. Prove that $f(a_1, \ldots, a_k) = h(a_1, \ldots, a_r)$ when $\{a_1, \ldots, a_k\}$ aren't distinct.

421. [*M25*] Sketch an algorithm that generates all of the k-ary operations on $D = \{0, 1, \ldots, d-1\}$ in the clone generated by a given finite set of operators O, when $k > 0$ is also given. *Hint:* Use a procedure that visits every l-tuple $i_1 \ldots i_l$ of nonnegative integers such that $\max\{i_1, \ldots, i_l\} = m$, given $l > 0$ and $m \ge 0$.

422. [*M21*] What is the clone of the binary operation '$x+3y$' over the positive integers?

423. [*M23*] Continuing (176) and the previous exercise, what clone is generated over the positive integers by the *two* binary operations '$x + 2y$' and '$x + 3y$'?

424. [*M25*] In (177), prove that m_5 is in the clone of m_4, but not vice versa.

▸ **425.** [*M25*] On the domain $D = \{0, 1, \ldots, d-1\}$, let $\mathcal{C}$ be the clone of the ternary operation $\mu_d(x, y, z) = (x - y + z) \bmod d$ in exercise 392.

a) What are the k-ary operations in $\mathcal{C}$? *Hint:* $(3w + x - 4y + z) \bmod d$ is one of them when $k = 4$, but $(3w + x - 5y + z) \bmod d$ isn't.

b) What is the spectrum of $\mathcal{C}$?

c) Find a value of d such that $\mathcal{C}$ contains no k-ary WNU operation for $k < 1000$.

2×2 matrices
matrices
rational numbers
Galois
field
automorphisms
spectrum
essential spectrum
full
Boolean operations
Willard pairing
clone
Maltsev term
spectrum

▶ **426.** [*M30*] On the domain $D = \{0, 1, \ldots, d-1\}$, let $\mathcal{C}$ be the clone of the ternary operation $t(x, y, z) = (x = y?\ z\colon x)$. (This operation is called the "discriminator".)

a) Consider the function $f(w, x, y, z) = t(t(z, y, w), t(x, t(z, w, x), z), t(w, x, y))$. If $d \ge 4$, for which values of (w, x, y, z) does it equal (i) w? (ii) x? (iii) y? (iv) z?

b) Describe all of the k-ary operations in $\mathcal{C}$.

c) What is the spectrum of $\mathcal{C}$?

427. [*M25*] How many k-ary functions are in the clone of the median function $\langle xyz\rangle$ on an ordered domain?

428. [*HM40*] How many clones on the domain $\{0, 1, 2\}$ contain the function $\langle xyz\rangle$?

429. [*M22*] Let O be a clone on the domain $\{0, 1, \ldots, d-1\}$, and let $g(x_1, \ldots, x_k)$ be an element of $\overline{O} = \mathrm{Pol}(\mathrm{Inv}(O))$. Prove that $g \in O$. *Hint:* Consider the set of d^k-tuples $R = \{f(0, \ldots, 0) \ldots f(d-1, \ldots, d-1) \mid f \text{ is a } k\text{-ary operation of } O\}$.

▶ **430.** [*M22*] On domain $\{0, 1, 2\}$, let $f_k(x_1, \ldots, x_k) = (x_1 + \cdots + x_k = 1?\ 0\colon 2)$; also let $R_m(x_1, \ldots, x_m) = [x_1 + \cdots + x_m = 1] \vee [2 \in \{x_1, \ldots, x_m\}]$. For which $k \ge 1$ and $m \ge 1$ is the function f_k a polymorphism of the relation R_m?

431. [*M15*] Why are *empty* (unsatisfiable) relations included in every relational clone?

432. [*M18*] True or false: Every relational clone contains (a) the complete m-ary relation D^m for all $m \ge 0$; (b) the empty m-ary relation $\emptyset$ for all $m \ge 0$; (c) the equality relation '$x = y$' for all $x, y \in D$.

433. [*M15*] If the binary relations R and R' both belong to a relational clone $\mathcal{C}$, does their composition $R \circ R' = \exists y\,(xRyR'z)$ also belong to $\mathcal{C}$?

434. [*M16*] If R and R' are m-ary elements of a relational clone $\mathcal{C}$, is $R \cap R'$ also in $\mathcal{C}$?

435. [*M22*] An exercise in the algebra of relations: Find a sequence σ such that $\Sigma_{12}\Sigma_{34}\Sigma_{56}\Pi_\sigma(\Theta \otimes \Theta \otimes \Theta) = \{001122\}$, where Θ is the relation defined in (159).

▶ **436.** [*M30*] Describe the relational clone $\mathrm{Inv}(\{t\})$ of the discriminator (exercise 426).

437. [*M20*] Exactly how many k-ary operations on $\{0, 1, \ldots, d-1\}$ are (a) NU? (b) WNU? (c) LWNU?

▶ **438.** [*27*] According to the previous exercise, $3^6 = 729$ of the ternary operations on $\{0, 1, 2\}$ are near unanimity operations. Six ternary digits, '$f_{012}f_{021}f_{102}f_{120}f_{201}f_{210}$', identify each of them by giving the values of $f(x_1, x_2, x_3)$ when x_1, x_2, x_3 are distinct.

Some of these operations can be derived from others. For example, if $g(x_1, x_2, x_3)$ is '000001', it turns out that $g(g(x_1, x_2, x_3), x_1, x_3)$ is '000000' and in fact that

$$g(x_1, g(x_3, x_2, g(x_1, x_3, x_2)), g(x_2, x_1, g(x_3, x_2, x_1))) \text{ is '100110'.}$$

Construct a digraph with 729 vertices, where there's an arc $u \longrightarrow v$ if and only if the operation v can be derived from the operation u. What are the strong components of this digraph? Is there any operation from which all the others can be derived?

▶ **439.** [*21*] There are $2^9 = 512$ binary relations on the domain $\{0, 1, 2\}$. How many of them are invariant under at least one of the $3^6 = 729$ NU operations considered in exercise 438? Which of those operations support the most relations? The fewest?

▶ **440.** [*20*] Modify exercise 398 so that it finds only the LWNU polymorphisms.

441. [*M23*] Describe and count the k-ary polymorphisms of Γ on domain D when (a) $\Gamma = \{\{a\} \mid a \in D\}$; (b) $\Gamma = \{D \setminus \{a\} \mid a \in D\}$.

clone
discriminator
spectrum
median function
clones
clone
empty
relational clone
complete m-ary relation
equality relation
composition
intersection of relations
algebra of relations
projection of a relation
equality selection of a relation
Cartesian product of relations
discriminator
NU
WNU
LWNU
strong components

442. [*M23*] (*Squashing functions.*) An *endomorphism* f of a set of CSP constraints is a function that preserves solutions: Whenever the values $(a_1, \ldots, a_n)$ for variables $(v_1, \ldots, v_n)$ satisfy all constraints, so do the values $(f(a_1), \ldots, f(a_n))$.

Let Γ be a set of relations on a finite domain D, and let O be the set of its 1-ary polymorphisms. (These are the endomorphisms of Γ, also the solutions to the indicator problem $\mathcal{I}_1(\Gamma)$.) If $f \in O$ and $g \in O$, we write fg for their functional composition: $fg(x) = f(g(x))$. Let's also denote the range of f by $fD = \{f(x) \mid x \in D\}$.

An endomorphism f is called a "squashing function" if its range size $|fD|$ is minimum, over all $f \in O$.

a) Prove that there's always a squashing function $e \in O$ such that $ee = e$.
b) Is the range fD of a squashing function uniquely determined by O?
c) In the remainder of this exercise let e be a fixed squashing function with $ee = e$. If $f \in O$ and $x, y \in eD$, prove that $f(x) = f(y)$ implies $x = y$.
d) Consequently every $f \in O$ has a quasi-inverse $f^{\sim} \in O$ such that $f^{\sim}fe = e$.
e) The set Γ is called a *core* if every endomorphism of Γ is an automorphism (a permutation of D). Show that every CSP $\mathcal{P}$ can be transformed in polynomial time to a CSP $\widehat{\mathcal{P}}$ whose relations $\widehat{\Gamma}$ are a core, where $\mathcal{P}$ is satisfiable if and only if $\widehat{\mathcal{P}}$ is.

443. [*HM30*] (*Laxity.*) In Theorem D, prove that Γ has an LWNU polymorphism w if and only if the associated core $\widehat{\Gamma}$ of exercise 442(e) has a WNU polymorphism $\widehat{w}$.

▶ **444.** [*M35*] Let $\Gamma = \{U, V, W\}$ be the following three binary relations on $D = \{0, 1, 2\}$:

$$U = \{01, 10\}; \qquad V = \{00, 10, 11, 12, 20, 22\}; \qquad W = \{00, 02, 10, 11, 12, 22\}.$$

Prove that $f(x, y, z) = (xy + yz + zx) \bmod 2$ is an LWNU polymorphism for Γ, but $\mathrm{Pol}(\Gamma)$ contains no k-ary WNU for any k. (Thus, "laxity matters" in Theorem D.)

445. [*M16*] Show that a lax Taylor term cannot be a projection function.

446. [*M30*] A *generalized majority-minority operation*, or 'GMM op' for short, is a k-ary operation on a domain D with the following property: For every $\{a, b\} \subseteq D$ we have either the "majority" law (182) for $x, y \in \{a, b\}$, also called near unanimity, or the "minority" law

$$f(y, x, \ldots, x) \;=\; y \;=\; f(x, \ldots, x, y)$$

for $x, y \in \{a, b\}$, also known as Maltsev's identity. We assume that $k \geq 3$. Notice that the majority law has k equalities, but the minority law has only two. When $k = 3$ the operator μ_d of exercise 392 satisfies the minority law.

Given a GMM op $f(x_1, \ldots, x_k)$, construct a Taylor term $t(x_1, \ldots, x_{2k-2})$.

447. [*M22*] Deep theorems of universal algebra prove that a clone of operations contains an LWNU if and only if it contains a "lax Siggers term," namely a quaternary operation $s(w, x, y, z)$ that satisfies the identity $s(r, a, r, e) = s(a, r, e, a)$.

a) Prove that such a clone also contains a lax Maróti pair (see (186)).
b) What lax Maróti pairs (p, q) arise when p or q is a projection?

448. [*M17*] Let p be a ternary operation on $\{0, 1, \ldots, d{-}1\}$. What's the probability that a *random* ternary operation q on that domain will form a lax Maróti pair with p?

449. [*21*] Continuing exercise 444, what are the lax Maróti pairs of those relations Γ?

▶ **450.** [*24*] Investigate all of the lax Maróti pairs of the relation G in (187).

451. [*M30*] Construct a lax Taylor term from any given lax Maróti pair (p, q).

452. [*M31*] What are the k-ary polymorphisms of the relation Z in (189)?

Squashing functions
endomorphism
indicator problem
composition
quasi-inverse
core
automorphism
Laxity
LWNU polymorphism
WNU polymorphism
LWNU polymorphism
WNU
Taylor term
generalized majority-minority operation
near unanimity
Maltsev
Taylor term
universal algebra
LWNU
lax Siggers term
Siggers term
lax Maróti pair
projection
lax Maróti pair
Maróti pairs
Taylor term

453. [*M24*] Construct a gadget for $2S = \{002, 020, 200\}$ from the relation Z in (189). *Hint:* First construct a gadget for the *unary* relation 'x is even'.

454. [*M22*] True or false: (a) If relation R has a k-ary NU polymorphism, then R also has a $(k{+}1)$-ary NU polymorphism. (b) If relation R has a k-ary NU polymorphism and $k > 3$, then R also has a $(k{-}1)$-ary NU polymorphism.

▶ **455.** [*M22*] Suppose $R' \subseteq R$, where the m-ary relation R has a k-ary NU polymorphism f, and the $(r{-}1)$-wise projections of R and $\langle R'\rangle_f$ are the same, for some r with $k \le r \le m$. Prove that the r-wise projections also agree.

456. [*20*] How many binary relations on $\{0, 1, 2, 3\}$ are preserved by the f in (191)?

457. [*M22*] Experiment with "random" NU operations on $\{0, 1, 2, 3\}$ instead of (191). How many binary relations do they tend to preserve?

458. [*16*] By hand, find all stubs of $\{002, 012, 013, 202, 212, 302\}$ with respect to (191).

459. [*M24*] What computation contributed the tuple τ with $\Pi_{25}\,\tau = 02$ to (196)?

460. [*20*] What compact stub lists elements of (196) in *decreasing* lexicographic order?

▶ **461.** [*M20*] According to answer 441, the polymorphism f in Algorithm N will preserve arbitrary unary operations if and only if f is *conservative* (that is, $f(x_1, \ldots, x_k) \in \{x_1, \ldots, x_k\}$). Modify step N1 for the conservative case, by initializing S to $D_1 \times \cdots \times D_n$ instead of D^n, where D_i is the initial domain of the ith variable.

462. [*HM21*] Given d, k, and n, step N1 constructs a stub of size $\Omega((nd)^{k-1})$ for the complete relation $[0\,.\,.\,d)^n$. Use randomization to show that much smaller stubs exist.

463. [*M21*] After the dictionary used in the text's implementation of step N5 has been seeded, explain how to apply (144) until the list of l-tuples is complete.

▶ **464.** [*M23*] There are $2^{27} = 134217728$ ternary relations on $\{0, 1, 2\}$. How many of them are invariant under at least one of the $3^{12} = 531441$ Maltsev operations? Which of those operations support the most relations? The fewest?

465. [*M30*] A $d \times d$ latin square defines the multiplication table of a "quasigroup," which is an algebraic system with a binary operator '$\cdot$' that satisfies the cancellation laws: $x \cdot y = x \cdot y'$ implies $y = y'$, and $x \cdot y = x' \cdot y$ implies $x = x'$. If $x \cdot y = z$ we write $x = z/y$ and $y = x\backslash z$; that is, $x = (x \cdot y)/y$ and $y = x\backslash(x \cdot y)$. The identities $x = (x/y) \cdot y$, $y = x \cdot (x\backslash y)$, and $x = y/(x\backslash y) = (y/x)\backslash y$ are also valid.

Using the operators '$\cdot$', '$/$', and '$\backslash$', construct a ternary operation $f(x, y, z)$ that satisfies the Maltsev identities (198).

466. [*M17*] What equivalence classes correspond to the forkings in Fig. 120?

467. [*M21*] Suppose $R' \subseteq R$, where the m-ary relation R has a Maltsev polymorphism f, and $\langle \Pi_{1\ldots(i-1)}\,R'\rangle_f = \Pi_{1\ldots(i-1)}\,R$, for some i with $1 < i \le m$. Assume furthermore that R' and R have the same level-i forkings. Prove that $\langle \Pi_{12\ldots i}\,R'\rangle_f = \Pi_{12\ldots i}\,R$.

468. [*20*] In (201), for which tuples τ of P' is $P' \setminus \tau$ still a valid Maltsev stub?

469. [*M22*] When S is a compact Maltsev stub, we sometimes want to update it by adding new tuples, while keeping it compact. Two subroutines are convenient for this purpose. First, "contribute τ to S," where $\tau = t_1 \ldots t_m$, leaves S unchanged if the monoforkings $(1, t_1, t_1)$, …, (m, t_m, t_m) are already present in S. The other, "contribute τ and τ' to S," is similar, but it leaves S unchanged only if every forking of $\{\tau, \tau'\}$ is already present. Devise an implementation for those subroutines.

NU polymorphism
k-wise projections
stubs
compact stub
lexicographic order
unary operations
conservative
complete relation $[0\,.\,.\,d)^n$
randomization
Maltsev operations
latin square
quasigroup
cancellation laws
Maltsev identities
equivalence classes
forkings
Maltsev polymorphism
Maltsev stub
stub
compact Maltsev stub
contribute
monoforkings

470. [*M20*] Let R' be a stub for R with respect to a Maltsev polymorphism f, and let R_c be the tuples of R that begin with c. Prove that (i, a, a') is a forking of R_c, with $i > 1$, if and only if (i, a, a') is a forking of R and $ca \in \Pi_{1i} R$.

471. [*M20*] Explain how to get the forking $(8, 2, 2)$ of P' into a stub for P_3 (see (201)).

472. [*M28*] Given R', R, and f as in exercise 470, let R_{-c} be the tuples of R that end with c. What's a good way to construct a compact stub, R'_{-c}, for R_{-c}? Illustrate your method by applying it to $R = P$, $R' = P'$, and $c = 2$ in Fig. 120 and (201).

473. [*M30*] Design Algorithm M, an "online" CSP solver based on a given Maltsev polymorphism $f(x, y, z)$ that operates on domain $D = \{0, 1, \ldots, d-1\}$. It inputs a sequence of zero or more f-invariant constraints on n variables that all have domain D. It should maintain a compact Maltsev stub S for the n-ary relation T, the set of all solutions to the problem-so-far; and it should terminate unsuccessfully if those constraints are unsatisfiable. Assume that each input constraint is given as a trie.

474. [*M48*] Is there a procedure analogous to Algorithms M and N that exploits *two* polymorphisms $p(x, y, z)$ and $q(x, y, z)$, when p and q form a Maróti pair?

475. [*M20*] If all constraints of a CSP are the relation Y^+ in (205), why is it easy to visit every solution with only polynomial-time delay between visits?

476. [*M46*] Is there an analog of Theorem D for the class of all Γ whose CSPs have solution sets that can be visited with polynomial-time delay between visits?

477. [*15*] True or false: If Montanari's procedure (206) ever sets $R_{ij} \leftarrow O$ (the all-0 matrix) for at least one pair (i, j), it will eventually set $R_{i'j'} \leftarrow O$ for *all* pairs (i', j').

478. [*23*] Summarize what (206) will do when presented with each of the following inputs, assuming that every unspecified relation R_{ij} is the identity matrix when $i = j$, or the all-1s matrix when $i \neq j$. (Domain sizes can be deduced from the given matrices.)

a) $n = 5$, $R_{12} = R_{23} = R_{34} = R_{45} = R_{51} = \binom{0\,1}{1\,0}$.

b) $n = 5$, $R_{12} = R_{23} = R_{34} = R_{45} = \left(\begin{smallmatrix}0&1&0\\0&0&1\\1&0&0\end{smallmatrix}\right)$ and $R_{51} = \left(\begin{smallmatrix}0&1&1\\0&0&1\\1&0&0\end{smallmatrix}\right)$.

c) $n = 5$, $R_{12} = R_{23} = R_{34} = R_{45} = R_{51} = \left(\begin{smallmatrix}0&1&1\\1&0&1\\1&1&0\end{smallmatrix}\right)$.

d) $n = 3$, $R_{12} = \binom{1\,1}{1\,0}$, $R_{13} = \binom{1\,1\,0}{0\,1\,1}$, and $R_{23} = \binom{1\,0\,1}{1\,0\,0}$.

479. [*M25*] (U. Montanari, 1974.) If (206) makes no change to any relation, prove that the following property holds for every $(s, t) \in R_{ij}$ and every sequence $k_0 k_1 \ldots k_r$ of indices with $k_0 = i$, $1 \le k_l \le n$ for $0 < l < r$, and $k_r = j$: There's a sequence of values $x_0 x_1 \ldots x_r$ such that $x_0 = s$, $(x_l, x_{l+1}) \in R_{k_l k_{l+1}}$ for $0 \le l < r$, and $x_r = t$.

▸ **480.** [*31*] A *constraint satisfaction automaton* (CSA) is a nondeterministic automaton based on a given CSP. Like all automata, it has a set Q of states, which contains a set $I \subseteq Q$ of input states and a set $\Omega \subseteq Q$ of output states, together with a transition rule that takes us from state to state. In this case the transitions have the general form

$$q \;\mapsto\; v_1 \setminus a_1, \ldots, v_t \setminus a_t,\ (v \leftarrow a?\ q' \colon q''), \quad \text{for some } t \ge 0,$$

where the v's are variables, the a's are domain elements, and the q's are states. The meaning is, "Begin deterministically: For $1 \le j \le t$, if v_j is unassigned, remove a_j from its domain if a_j was present. Then branch nondeterministically: Either (i) assign a as the value of variable v, and go to state q', or (ii) remove a from the domain of v and go to state q''." Variable v must not previously have been assigned a value. Case (i) is permitted only when a is in v's current domain. It means that the domain of v is reduced to the single value $\{a\}$; furthermore, the domain of every other unassigned

online
trie
Maróti pair
polynomial-time delay
dichotomy theorem
Montanari
O, the all-0 matrix
Montanari
constraint satisfaction automaton
CSA
nondeterministic automaton
automata
states
input states
output states
transition rule
variables
domain elements

variable w is also reduced, if necessary, so that every constraint for which all variables but w are assigned is fully satisfied by every value in w's remaining domain.

A CSA computation begins in an initial state, with all variables unassigned, and with all domains equal to the initial domains but restricted by the unary constraints. It ends successfully in an output state when all variables have been assigned; or it can end unsuccessfully, either in a state q for which some domain is empty, or for which all variables are assigned but $q \notin \Omega$, or for which no transition rule was specified. The *solutions* of a CSA are the tuples of assigned values that a successful computation can produce. (In particular, those solutions will also solve the given CSP.)

Either v or a in the '$v \leftarrow a$' part of a transition rule, or both, can be replaced by an *asterisk* ($*$), meaning that the automaton itself is supposed to choose the variable and/or the value to be assigned, deterministically, using an arbitrary heuristic. Of course such a "wildcard" transition is inapplicable when no valid assignment is possible.

For example, the CSA with $Q = I = \Omega = \{q\}$ and the wildcard transition rule '$q \mapsto (* \leftarrow *?\ q\colon q)$' simply has the same solutions as the given CSP. The CSA with $Q = \{q_0, q_1, q_2\}$, $I = \{q_0\}$, $\Omega = \{q_2\}$, and transitions

$$q_0 \mapsto (v \leftarrow a?\ q_1\colon q_2); \quad q_1 \mapsto w \setminus b, (* \leftarrow *?\ q_2\colon q_2); \quad q_2 \mapsto (* \leftarrow *?\ q_2\colon q_2)$$

has all solutions except those for which $v = a$ and $w = b$.

The domain element in a transition rule can also be a *named wildcard* of the form 'a^*', where a is a local identifier. It means that the value a chosen by the automaton can be used in the specification of the states q' and q''. For example, the transition rule

$$q \mapsto (v \leftarrow a^*?\ q_a\colon q_{-a})$$

will cause the automaton to choose an arbitrary value a in v's domain. Then if, say, $a = 3$, it will branch nondeterministically, either assigning $v \leftarrow 3$ and going into state q_3 or making no assignment and going into state q_{-3}.

Notice that a CSA essentially adds a global constraint to the given CSP. "Find all solutions that correspond to a sequence of states in the CSA from I to Ω." It can be simulated by any procedure that makes further domain reductions, for example to maintain consistency, as long as those reductions don't eliminate any solutions.

The following examples exhibit some of the versatility provided by this CSA formalism. Let the variables of a given CSP be $\{v_1, \ldots, v_n\}$, each with domain $[0\,..\,d) = \{0, 1, \ldots, d-1\}$, and subject to any number of further constraints.

a) Define a CSA whose solutions $v_1 \ldots v_n$ are those with $(v_1 + \cdots + v_n) \bmod 5 \in \{1, 3\}$.
b) Define a CSA for the solutions where each value occurs at most twice.
c) Define a CSA for the solutions where each value occurs either twice or not at all.
d) Similarly, design a CSA for all solutions $v_1 \ldots v_n$ that are *restricted growth strings*. (See Section 7.2.1.5; in particular, $v_1 = 0$ and v_2 is 0 or 1.)
e) Let $d = 2$, and restrict the solutions to binary strings $v_1 \ldots v_n$ that correspond to *nested parentheses* when $0 \leftrightarrow$ (and $1 \leftrightarrow$). (In particular, $v_1 = 0$ and $v_n = 1$.)

481. [*21*] Suppose the reflection $v_n \ldots v_2 v_1$ solves a certain CSP whenever $v_1 v_2 \ldots v_n$ does. All domains are $[0\,..\,d)$. Design a CSA that yields only one of those solutions.

482. [*23*] Suppose the cyclically shifted tuple $v_2 \ldots v_n v_1$ solves a certain CSP whenever $v_1 v_2 \ldots v_n$ does. Design a CSA that yields just one solution in each equivalence class under cyclic shifts. All domains are $[0\,..\,d)$. *Hint:* Consider *prime strings* (Section 7.2.1.1).

solutions
asterisk
wildcard
global constraint
consistency
restricted growth strings
nested parentheses
reflection
symmetry breaking (removal)
breaking symmetries
cyclic shifts
Lyndon words, see prime strings
prime strings

▸ **483.** [*28*] Solutions to the n queens problem belong to the same equivalence class if they differ only by a reflection and/or rotation of the board. The purpose of this exercise is to define *canonical solutions*, of which there's exactly one in each class.

Denote the cells by (i,j) for $0 \le i,j < n$. Let R_i be the column containing a queen in row i, and let C_j be the row containing a queen in column j; thus $R_i = j$ if and only if $C_j = i$. Let $\bar{x} = n-1-x$; notice that rotation by 90° changes R_i to $C_{\bar\imath}$ and C_j to $\bar{R}_j$.

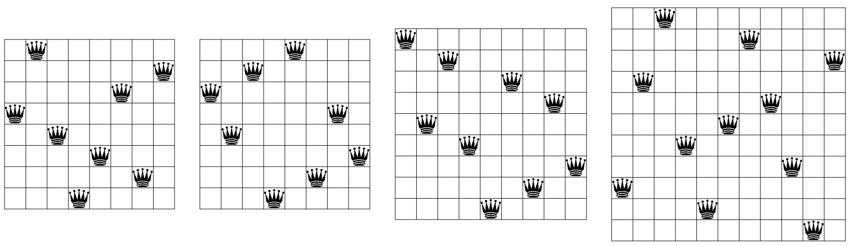
a) Let $(a_i, b_i, c_i, d_i) = (R_i, C_{\bar\imath}, \bar{R}_{\bar\imath}, \bar{C}_i)$. Can we have $\{a_i, b_i, c_i, d_i\} \cap \{\bar{a}_i, \bar{b}_i, \bar{c}_i, \bar{d}_i\} \neq \emptyset$?

b) How does reflection of the board change the numbers (a_i, b_i, c_i, d_i) of a solution?

c) Let $n' = \lfloor n/2 \rfloor$. Write out the eight values of the $4\lceil n/2 \rceil$-tuple

$$(a_{n'}, b_{n'}, c_{n'}, d_{n'}; a_{n'+1}, b_{n'+1}, c_{n'+1}, d_{n'+1}; \ldots; a_{n-1}, b_{n-1}, c_{n-1}, d_{n-1})$$

that occur when the following solutions are rotated and/or reflected:



d) Explain why the lexicographically least of eight such tuples is a canonical solution.

e) True or false: If $n = 2n'$, the canonical tuple begins with $a_{n'} \le n' - 2$.

f) Design a CSA for canonical solutions to the n queens problem.

484. [*27*] What's the lexicographically *largest* canonical solution that uses 32 queens?

485. [*24*] A *superqueen* (also called an "amazon") combines the moves of a queen and a knight. Use the methods of exercise 483 to determine the number of inequivalent solutions to the n superqueens problem for small n.

486. [*40*] Implement a version of Algorithm S that includes proof logging, in order to allow an independent verifier to certify that every solution has been obtained. (In particular, if the problem has no solutions, the algorithm should output some sort of certificate of unsatisfiability.)

n queens problem
queens problem
equivalence class
reflection
rotation
canonical solutions
lexicographically least
superqueen
amazon
knight
n superqueens problem
proof logging
certificate of unsatisfiability
Christie, Agatha
McGinty, Mrs
POIROT

What I should like to do is
to eliminate one or other of the trails I indicated just now.
And to eliminate the Mrs McGinty trail — trail No 1 — will
obviously be quicker and easier than to attack trail No 2.

— HERCULE POIROT, in *Blood Will Tell* (1951)

CHAUCER
Bennaceur
Järvisalo
Niemelä
tautology
binary relation
Mackworth
computer vision
historical notes
Fikes

ANSWERS TO EXERCISES

For oure excercise, With sharpe scourges of adversitee Ful ofte to be bete.
— GEOFFREY CHAUCER, *The Tales of Caunterbury* (c. 1395)

SECTION 7.2.2.3

1. Only BCAON, BCUOD, BLUED, and SCION satisfy R_1 and R_3; the first two fail R_2.

2. (a) The literals of each clause define the domain of the corresponding variable. If one clause contains x and the other contains $\bar{x}$, forbid the pair $x\bar{x}$. [See H. Bennaceur, *ECAI* **12** (1996), 155–159. Satisfiability/unsatisfiability is preserved, but the number of solutions may change; when $m = 1$ the 3SAT problem has 7 solutions, the CSP has 3.]

(b) Seven variables $c_1 \in \{1, 2, \bar{3}\}$, ..., $c_7 \in \{\bar{3}, \bar{4}, \bar{1}\}$; $\binom{7}{2} = 21$ constraints. Three constraints are satisfied in 6 ways (for example, $c_1c_5 \in \{1\bar{2}, 13, 2\bar{1}, 23, \bar{3}\bar{1}, \bar{3}\bar{2}\}$); the other 18 in 8 ways ($c_1c_7 \in D_1 \times D_7 \setminus 1\bar{1}$). The SAT problem has 2 solutions, the CSP has 48.

(c, d) Adding Boolean variables $\{x_1, x_2, x_3, x_4\}$, we need only 5-out-of-6 constraints such as $c_1x_1 \in \{11, 20, 21, \bar{3}0, \bar{3}1\}$. [See M. Järvisalo and I. Niemelä, *Workshop on Modelling and Reformulating Constraint Satisfaction Problems* **3** (2004), 111–124.]

3. Let $x_{1\mathtt{B}} = [x_1 = \mathtt{B}]$, etc. Then the clauses $(x_{1\mathtt{B}} \vee x_{1\mathtt{S}})$, $(\overline{x_{1\mathtt{B}}} \vee \overline{x_{1\mathtt{S}}})$, $(x_{2\mathtt{C}} \vee x_{2\mathtt{L}})$, $(\overline{x_{2\mathtt{C}}} \vee \overline{x_{2\mathtt{L}}})$, $(x_{3\mathtt{A}} \vee x_{3\mathtt{I}} \vee x_{3\mathtt{U}})$, $(\overline{x_{3\mathtt{A}}} \vee \overline{x_{3\mathtt{I}}})$, $(\overline{x_{3\mathtt{A}}} \vee \overline{x_{3\mathtt{U}}})$, $(\overline{x_{3\mathtt{I}}} \vee \overline{x_{3\mathtt{U}}})$, $(x_{4\mathtt{E}} \vee x_{4\mathtt{O}})$, $(\overline{x_{4\mathtt{E}}} \vee \overline{x_{4\mathtt{O}}})$, $(x_{5\mathtt{D}} \vee x_{5\mathtt{N}})$, $(\overline{x_{5\mathtt{D}}} \vee \overline{x_{5\mathtt{N}}})$ establish the domains. And the clauses $(R_{11} \vee R_{12} \vee R_{13})$, $(\overline{R}_{11} \vee x_{1\mathtt{B}})$, $(\overline{R}_{11} \vee x_{3\mathtt{A}})$, $(\overline{R}_{11} \vee x_{5\mathtt{N}})$, $(\overline{R}_{12} \vee x_{1\mathtt{B}})$, $(\overline{R}_{12} \vee x_{3\mathtt{U}})$, $(\overline{R}_{12} \vee x_{5\mathtt{D}})$, $(\overline{R}_{13} \vee x_{1\mathtt{S}})$, $(\overline{R}_{13} \vee x_{3\mathtt{I}})$, $(\overline{R}_{13} \vee x_{5\mathtt{N}})$, ..., $(\overline{R}_{33} \vee x_{2\mathtt{L}})$, $(\overline{R}_{33} \vee x_{4\mathtt{E}})$, $(\overline{R}_{33} \vee x_{5\mathtt{D}})$ establish the relations.

(Many other encodings are possible; this one is systematic and avoids trickery.)

4. Primary R_1, R_2, R_3; secondary x_1, ..., x_5. Options 'R_1 x_1:B x_3:A x_5:N', 'R_1 x_1:B x_3:U x_5:D', 'R_1 x_1:S x_3:I x_5:N', ..., 'R_3 x_2:L x_4:E x_5:D'. (See exercise 7.2.2.1–100.)

5. There are just two subsets of $\{\epsilon\}$, namely $\emptyset$ and $\{\epsilon\}$. The first of those relations is always false, so it's a constraint that wipes out all solutions. The second is a tautology, always true; it doesn't really constrain anything. (In general, there are $2^{d_1 \ldots d_k}$ k-ary relations on $(D_1, \ldots, D_k)$, when each D_i has d_i elements; hence there are 2^{d^k} k-ary relations over any d-element set. One of them is always false; another is always true.)

6. Given any binary relation on $A \times B$, consisting of ordered pairs (a, b), math texts say furthermore that the "domain" is the set of left coordinates and the "range" is the set of right coordinates. Yet constraint satisfiers have happily spoken of the domains of variables ever since Mackworth's paper of 1977 introduced the terminology.

Mackworth was influenced by earlier work in computer vision, where the value of a variable was often a rectangle (say) where some object might be found in a digital image; that would be an extramathematical sense of the word "domain," like a "dominion." Moreover, his main focus was on constraints, not variables; the domains of the constraints are the values of the variables. [Fikes had actually used the term "range," *not* domain, in his original paper of 1970.]

7. False. For example, (012343434) is a homomorphism from C_9 to C_5. (The most that can be concluded, from the existence of a homomorphism from C_{odd} to G, is that G isn't bipartite, because it contains an odd cycle.)

8. (Solution by P. Jeavons.) Construct a new graph G' by replacing every edge $u \longrightarrow v$ of G by a path $u \longrightarrow uv \longrightarrow vu \longrightarrow v$, where uv and vu are new vertices. Then there's a homomorphism from G' to C_5 if and only if there's a homomorphism from G to K_5. Hence the problem is NP-complete. (In general the "H-coloring problem," to decide whether or not a homomorphism from G to H exists, is trivial when H is bipartite; otherwise it's NP-complete [P. Hell and J. Nešetřil, *J. Comb. Theory* **B48** (1990), 92–110].)

9. (a) Let $\overline{E} = \{\{u,v\} \mid u \neq v \text{ and } \{u,v\} \notin E\}$ be the edges of the complement graph $\overline{G}$. (See Eqs. 7–(15) and 7–(35).) An independent set in G is a clique in $\overline{G}$. "Is there a homomorphism from K_k to $(V, \overline{E})$?"

(b) The vertices *not* in a cover are independent. Use (a) with $k \leftarrow |V| - k$.

(c) They're isomorphic if and only if each is embeddable in the other. It's a *single* GCP if $|V| = |V'|$: "Is there a homomorphism from $(V, E, \overline{E})$ to $(V', E', \overline{E'})$?"

(d) Let G' be the graph on $\{1, \ldots, |V|\}$ for which $i \longrightarrow j$ if and only if $|i-j| \leq k$. "Is G embeddable in G'?"

(e) Let A' be the relation $\{(uv, u'v') \mid v = u'\}$ on ordered pairs of vertices in V, and let $(\{0, \ldots, m-1\}, O)$ be the oriented cycle $\vec{C}_m$, where $m = |A|$ and $O = \{ij \mid j = (i+1) \bmod m\}$. "Is there a homomorphism from $(\{0, \ldots, m-1\}, O, \neq)$ to $(A, A', \neq)$?"

10. "$u \neq v$ implies $h(u) \parallel h(v)$" is the same as saying that $|V|$ mutually unlike k-tuples satisfy relation R. And that's precisely the kDM problem (k-dimensional matching).

11. Given similar relational structures $S = (U, R_1, \ldots, R_t)$ and $S' = (U', R'_1, \ldots, R'_t)$, the corresponding CSP has variables U, each with domain U'. Suppose $U = \{1, \ldots, n\}$. The values $x_{i_1} \ldots x_{i_k}$ of every k-tuple $i_1 \ldots i_k \in R_j$, where $k = k_j$, are constrained to satisfy the relation R'_j, for $1 \leq j \leq t$.

12. (a) Let T be the matrix $\bigl(\begin{smallmatrix} wz^- & w^-z \\ wz & w^-z^- \end{smallmatrix}\bigr)$, where z^- denotes $1/z$. By induction we have $G_N(z) = {}^{(w\;w^-)}\,T^{N-1}\bigl(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\bigr)$. For example, $G_1(z) = w^- + w$ and $G_2(z) = w^{-2}z^- + 2z + w^2 z^-$.

Now let $u = (\frac{w+w^-}{2})/z$, $v = ((\frac{w-w^-}{2})/z)^2 + z^2$, $\lambda = u + \sqrt{v}$, $\mu = u - \sqrt{v}$. Then we have $T = S\bigl(\begin{smallmatrix} \lambda & 0 \\ 0 & \mu \end{smallmatrix}\bigr)S^-$, where $S = \bigl(\begin{smallmatrix} \lambda - w^-z^- & \mu - w^-z^- \\ wz & wz \end{smallmatrix}\bigr)$. Hence $G_N(z) = a\lambda^{N-1} + b\mu^{N-1}$, with coefficients $a = \lambda z + (z - z^3)/\sqrt{v}$, $b = \mu z - (z - z^3)/\sqrt{v}$. (Notice that when $B = 0$, everything simplifies enormously because $w = 1$. For example, $\lambda = z^- + z$.)

(b) Differentiate and plug in. (The exact formulas are hairy, until we get to (c).)

(c) When N is large we can ignore μ. Thus $G'(z)/G(z)$ in (b) is $\frac{d}{dz} \ln G(z) \sim \frac{d}{dz} N \ln \lambda$, where $\lambda = e^{\beta} \cosh \beta B + \sqrt{e^{2\beta} \sinh^2 \beta B + e^{-2\beta}}$.

(d) Now we have $G_k(z) = {}^{(w\;w^-)}\,T^{k-1} X T^{N-k}\bigl(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\bigr)$, where $X = \bigl(\begin{smallmatrix} -1 & 0 \\ 0 & 1 \end{smallmatrix}\bigr)$. To put this in closed form, let $Y = S^- X S$, so that $T^{k-1} X T^{N-k} = S\bigl(\begin{smallmatrix} \lambda & 0 \\ 0 & \mu \end{smallmatrix}\bigr)^{k-1} Y \bigl(\begin{smallmatrix} \lambda & 0 \\ 0 & \mu \end{smallmatrix}\bigr)^{N-k} S^-$. Hence $G_k(z) = \hat{a}\lambda^{N-1} + \hat{b}\mu^{N-1} + c\lambda^{N-k}\mu^{k-1} + c\lambda^{k-1}\mu^{N-k}$, where $\hat{a} = (\frac{w-w^-}{2z})a/\sqrt{v}$, $\hat{b} = (\frac{w^- - w}{2z})b/\sqrt{v}$, $c = \frac{w-w^-}{2}(1-z^2)/v$. So the average comes to $(\hat{a} + c\lambda/(N\sqrt{v}))\lambda^{N-1} + (\hat{b} - c\mu/(N\sqrt{v}))\mu^{N-1}$, divided by $G(z)$; asymptotically, it's $\sinh \beta B / \sqrt{\sinh^2 \beta B + e^{-4\beta}}$.

[This answer is based on §2.5.1 of the book by Mézard and Montanari.]

13. It turns out that 17 constraints like (12) are sufficient to force $x_i \neq x_j$ whenever $i \neq j$. (The problem without (14) is in fact equivalent to "radio coloring" as in exercise 7.2.2.2–36; the graph in (11) can't be 7-colored radiowise.) But the second model, with only 7 constraints like (15), has 20,358 solutions without the all-different constraint! We can, for instance, set `A` $\leftarrow$ `C` $\leftarrow$ `E` $\leftarrow$ `G` $\leftarrow 1$ and `B` $\leftarrow$ `D` $\leftarrow$ `F` $\leftarrow$ `H` $\leftarrow 8$.

[The inventor of this puzzle is unknown. After Martin Gardner publicized it in *Scientific American* **206**, 2 (February 1962), 150, Fred Gruenberger told him that he'd learned of the problem in 1961 from a friend at Walt Disney Studios, "where it had already consumed a fair amount of Mr. Disney's staff time." Gruenberger had used

Jeavons
NP-complete
H-coloring problem
Hell
Nešetřil
complement graph
clique
embeddable
oriented cycle
$\vec{C}_m$
kDM problem
k-dimensional matching
3DM
Mézard
Montanari
radio coloring
Gardner
Gruenberger
historical notes
Disney

it that year in a TV documentary, "How a Digital Computer Works," featuring three high-school students who solved it from scratch in about five minutes, working at a blackboard, while a computer would supposedly have to run through $8! = 40320$ permutations in order to find the answer! Ten years later, D. K. Cohoon called (11) the "no-touch puzzle" in *Math. Mag.* **45** (1972), 261–265, without mentioning his source.]

Notice that the CSP model using (15) is essentially based on the *complement* of the graph in (11), which has only 11 edges and is easy to draw. According to that model, the problem is to make $(\mathtt{A}, \mathtt{B}, \ldots, \mathtt{H})$ label a *Hamiltonian path* in the complement graph — an observation made independently by T. H. O'Beirne and H. Koplowitz in letters to Gardner, and later by Cohoon. There are four such paths, easy to find.

Cohoon
no-touch puzzle
Hamiltonian path
O'Beirne
Koplowitz
Gardner
symmetry
pairwise ordering trick
XC: exact cover
sharp preference heuristic
Kμ: kilomems
symmetry
Dull
slot

14. We can save a factor of 2 by assuming that `A` occurs in the left half of the graph: Remove `A` from the domains of $\{x_2, x_5, x_6, x_8\}$ in the first model; remove $\{2, 5, 6, 8\}$ from the domain of `A` in the second.

To save another factor of 2, we can add the constraint $x_2 < x_8$ (say) in the first model. That can't be done in the second, without probing deeper into the solution.

15. Let there be $17 \cdot 7$ secondary items juv, one for every combination of a letter j with $\mathtt{A} \le j < \mathtt{H}$ and an edge $u — v$, where $u < v$. There are 64 options (v, k), where $1 \le v \le 8$ and $\mathtt{A} \le k \le \mathtt{H}$; option (v, k) contains the primary items v and k, meaning that vertex v is labeled with letter k. To prevent adjacent letters in edge $u — v$, add secondary item juv to options (u, j), (v, j), $(u, j{+}1)$, and $(v, j{+}1)$. For example, option $(2, \mathtt{E})$ is '`2 E D12 D24 D25 D26 E12 E24 E25 E26`'. (This construction nicely incorporates both of the text's CSP models; notice that the all-different constraint "comes for free.")

That XC problem has 4 solutions, found in 300 kilomems with 485 nodes in the search tree. To break the symmetry as in exercise 14, first remove options $(2, \mathtt{A})$, $(5, \mathtt{A})$, $(6, \mathtt{A})$, $(8, \mathtt{A})$; then also remove options $(2, \mathtt{H})$ and $(8, \mathtt{B})$, and use the pairwise ordering trick of exercise 7.2.2.1–20 with $m = 6$, $\alpha_i = (2, \mathtt{B} + i)$, $\beta_i = (8, \mathtt{C} + i)$ to ensure that the label of 2 is less than the label of 8. (This introduces secondary items $y_1, \ldots, y_5$; it also puts y_2 and y_3 into option $(2, \mathtt{E})$.) The resulting XCC problem has 1 solution, costs 108 kilomems, and examines 146 nodes. [If we cleverly change 5 to #5 and use the sharp preference heuristic of exercise 7.2.2.1–10, thereby forcing the first branch to be on vertex 5, the search tree decreases to just 43 nodes and the running time to just 35 Kμ.]

16. Let variables (`AB`, `BC`, `CD`, `DE`, `EF`, `FG`, `GH`) each have the 11-element domain of all edges not in the graph. Constrain each of (`AB`, `BC`), ..., (`FG`, `GH`) to be one of the 48 ordered pairs of edges that have one vertex in common. Also constrain each of the nonoverlapping pairs of variables, namely (`AB`, `CD`), (`AB`, `DE`), ..., (`EF`, `GH`), to be one of the other 62 ordered pairs of edges. (The all-different constraint would be redundant.)

17. FABABACDCE (and its mirror image ECDCABABAF).

18. The mirror image of a solution with $f \ge 5$ has $f < 5$. (Alternatively, we could have assumed that $d < 5$, or $e < 5$, or even that $a_1 < 5$; but F is probably harder to place. When t is even, the symmetry can be broken by choosing *any* model of odd multiplicity, and requiring more than half of its occurrences to be $< t/2$.)

19. (Solution by B. C. Dull.) No. If that new constraint is violated, so is (18) when $l = l' + l''$, because we have $f_{0k} + f_{1k} + \cdots + f_{(l'q_k - 1)k} \le l'p_k$ by (17).

But that "solution" is *wrong*! The new constraints *are* useful, for example, when $l'' = 0$ and we have a partial solution for which f_{ik} is known only when $i > t/2$.

20. Introduce a primary item, representing slot i, for $0 \le i < t$. Also a primary item for the name of each model type, with its given multiplicity.

Thus there will be one option for each slot, and one for each type. (In Fig. 100, for example, item A has multiplicity 3.)

To implement the constraints (17), introduce primary items u_{jk} for $0 \le j \le t - q_k$ and $0 \le k < m$, having multiplicity $[0\,..\,p_k]$. (If $p_k = 1$, this item could be secondary.) Include u_{jk} in the option for every model that uses feature k in slot i, for $j \le i < j + q_k$. (Thus, one option for Fig. 100 is '2 B u_{10} u_{20} u_{03} u_{13} u_{23}'.)

21. Notice that $f_{0k} + \cdots + f_{(t-lq_k-1)k} \ge r_k - lp_k$ if and only if $\bar f_{0k} + \cdots + \bar f_{(t-lq_k-1)k} \le s_{lk} = t - lq_k - r_k + lp_k$. Therefore introduce primary items v_{lk} for $0 < l < \lceil r_k/p_k \rceil$ and $0 \le k < m$, having multiplicity $[0\,..\,s_{lk}]$. Include v_{lk} in the option for every model that does *not* use feature k, for every slot i in the range $0 \le i < t - lq_k$. (If $s_{lk} = 0$, any options that would include v_{lk} should be omitted, like the options for 0 B and 0 D in Fig. 100. The option in answer 20 becomes '2 B u_{10} u_{20} v_{41} v_{71} v_{72} u_{03} u_{13} u_{23}'. Other redundant constraints such as those of exercise 19 can be implemented in a similar way.)

22. Yes: The only solutions are FEBAGAHDCAGECDACDCEGACDHAGABEF and its mirror image (change 'AC' to 'CA' in the middle). The running time is (a) 28 gigamems, with 22 meganodes in the search tree; (b) 4 megamems, with 1670 nodes.

23. No; Algorithm 7.2.2.1M verifies this in 202 Gμ, with 158 meganodes.

There's actually an easy way to prove the impossibility by hand, because Model F can only appear at the beginning, or at the end, or next to Model 0; furthermore F0F is impossible. Hence the shortest possible way to produce four Model Fs is to put one at each end and to have two occurrences of F0 or 0F inside the sequence.

One way to solve the 62-car problem is to place '00' between two solutions of the 30-car problem. That 62-car problem actually has 19050 solutions, of which 18 are unchanged under left-right reflection and the others form 9516 mirror pairs. Only 69 Gμ of computation are needed to find the symmetric ones. Every solution begins with FEBA and ends with ABEF. Six of the palindromic solutions, such as FEBAF0HDCAGECDC-AGEBAGAHDCAGECDAADCEGACDH . . . GACDH0FABEF, have two F's near each end.

24. (a) We've seen equivalent problems before (for example, in Sections 5.4.2, 7.2.1.1, and 7.2.1.7); but let's start from scratch. Consider the digraph whose vertices are the q-bit patterns α with $\nu\alpha \le p$, having arcs $\alpha \longrightarrow \beta$ when the last $q-1$ bits of α match the first $q-1$ bits of β. (It's a subgraph of the digraph in exercise 2.3.4.2–23.) The answer is the number of walks of length 10 that start from vertex 0^q in this digraph: 144 when $(p,q) = (1,2)$; 60 when $(p,q) = (1,3)$; 504 when $(p,q) = (2,3)$.

(b) $p\lfloor n/q \rfloor + \min(p, n \bmod q)$.

(c) In general, the generating function $G(z)$ for walks of length n from vertex σ in a given digraph is $\sum_\alpha G^\alpha(z)$, where $G^\alpha(z) = [\alpha = \sigma] + z\sum_{\beta\to\alpha} G^\beta(z)$ for each vertex α. For example, when $(p,q) = (1,2)$ and $\sigma = 00$ we have $G(z) = G^{00}(z) + G^{01}(z) + G^{10}(z)$; $G^{00}(z) = 1 + z(G^{00}(z) + G^{10}(z))$; $G^{01}(z) = z(G^{00}(z) + G^{10}(z))$; $G^{10}(z) = zG^{01}(z)$; hence $G_{12}(z) = G(z) = (1+z)/(1-z-z^2)$. (They're Fibonacci numbers: $C_{12n} = F_{n+2}$.)

Similarly $G_{13}(z) = (1+z+z^2)/(1-z-z^3)$ (Narayana's cows sequence); $G_{23}(z) = (1+z+z^2)/(1-z-z^2-z^3)$ (Tribonacci numbers). In general, $G_{1q}(z) = (1+z+\cdots+z^{q-1})/(1-z-z^q)$ and $G_{(q-1)q}(z) = (1+z+\cdots+z^{q-1})/(1-z-z^2-\cdots-z^q)$. But the other cases don't fit any evident pattern: $G_{24}(z) = (1+z+z^2+z^3-z^4-z^5)/(1-z-z^2-z^4+z^6)$; $G_{25}(z) = (1+z+2z^2+2z^3+2z^4-z^5-z^6-2z^7-z^8-z^9)/(1-z-z^3-2z^5+z^8+z^{10})$; $G_{35}(z) = (1+z+z^2+2z^3+2z^4-z^5-z^6-z^8-z^9)/(1-z-z^2-z^4-2z^5+z^7+z^{10})$.

25. (a) Given a plane partition whose elements P_{ij} satisfy $0 \le P_{ij} \le m$, $P_{ij} \ge P_{i(j+1)}$, $P_{ij} \ge P_{(i+1)j}$, and $P_{ij} = 0$ for $i > p$ or $j > q - p$, construct an extreme (p/q)-string

palindromic
walks
digraph
Fibonacci numbers
Narayana's cows sequence
Tribonacci numbers

as follows: For $k = 1, 2, \ldots, m$, form the tableau shape whose boxes are the elements with $P_{ij} \geq k$, and write down its *rim representation*, as in 7.2.1.4–(13) and (14). (This will be a binary string of length q that contains exactly p 1s.)

For example, suppose $p = 2$, $q = 5$, $m = 6$, and consider the plane partition ${}^{441}_{210}$. The rim representations for $k = 1, 2, 3, 4, 5, 6$ are respectively 10100, 01010, 01001, 01001, 00011, 00011; and the concatenation of those strings is extreme. (This beautiful construction, devised by Ira Gessel in March 2020, is clearly reversible.)

(b) Let $r = n \bmod q$. Then c_{pqn} is $e_{(p-r)(q-r)}(\lfloor n/q \rfloor)$, if $r < p$; 1, if $r = p$; $e_{pr}(\lceil n/q \rceil)$, if $r > p$.

26. Each point (x, y, z) satisfies three equations in three unknowns, so the respective vertices are $\bigl((-140,0,0)$, $(75,75,-100)$, $(0,252,-280)$, $(40,-100,-200)$, $(90,-50,0)$, $(140,50,0)$, $(-240,0,200)$, $(140,0,0)$, $(240,0,200)$, $(-140,-50,0)$, $(-90,50,0)$, $(-40,100,-200)$, $(0,-252,-280)$, $(-75,-75,-100)\bigr)$. Then the seven hexagons 023 — 310 — 501 — 054 — 460 — 206, 134 — 421 — 612 — 165 — 501 — 310, ..., 612 — 206 — 460 — 643 — 356 — 165 do the job, because we can construct a model (with stiff paper or computer graphics). [*Structural Topology* #13 (1986), 69–80.]

27. The simplest example whose histoscape is *not* a 3VP is the identity matrix $\binom{1\,0}{0\,1}$, because more than three edges (in fact, five of them) touch vertex $(1,1,1)$. Moreover, the edge from $(1,1,1)$ to $(1,1,0)$ is adjacent to *four* faces! [*Beware:* The standard row-and-column convention for coordinates ij of a matrix are sometimes confusingly at odds with the standard Cartesian coordinates (x, y, z) of three-dimensional geometry.]

In general, consider the histoscape for $\binom{a\,b}{c\,d}$ when $a = \max\{a,b,c,d\}$ and $b \geq c$. It fails to be 3VP when $d > b$, because the cubies $(0,0,b)$ and $(1,1,b)$ have a boundary edge in common. A milder violation occurs when $a > b$ and $c > d$, because four faces meet at vertex $(1,1,b)$. Four faces meet at that vertex also when $a > b = d > c$.

But the other cases are fine: *Case 1*, $a = b = c \geq d$. *Case 2*, $a = b > \max\{c,d\}$. *Case 3*, $a > b = c = d$. *Case 4*, $a > b > d \geq c$. When we take symmetry into account, these cases contribute respectively $\bigl(\binom{10}{1}+4\binom{10}{2}, 4\binom{10}{2}+8\binom{10}{3}, 4\binom{10}{2}, 8\binom{10}{3}+8\binom{10}{4}\bigr)$ valid 3VPs, a total of 4150.

(And the B^4 histoscapes of 2×2 matrices with $a_{ij} < B$ yield $B^4/3 + O(B^3)$ 3VPs.)

28. An $m \times n$ histoscape is a 3VP if and only if $r(a_{(i-1)(j-1)}, a_{(i-1)j}, a_{i(j-1)}, a_{ij})$ holds for $1 \leq i < m$ and $1 \leq j < n$, where r is the relation in the previous answer, because the vertices (x, y, z) for which $x = i$ and $y = j$ depend only on those four matrix entries.

The best way to enumerate the solutions to a CSP whose relations are enforced in such a structured manner is to use the techniques of "dynamic programming," which is the topic of Section 7.7. This problem offers us a nice preview of those coming attractions, because the following remarkable algorithm finds the total number of $m \times n$ matrices whose 2×2 submatrices all satisfy an *arbitrary* quaternary relation r. We assume that each variable has the domain $0 \leq a_{ij} < t$; and we use an $(n+1)$-dimensional array of t^{n+1} potentially large integers $c(x_0, \ldots, x_n)$, all initially 1.

Q1. [Iterate on rows.] Do step Q2 for $i = 1, \ldots, m-1$; then go to Q3.

Q2. [Iterate on columns.] Do subroutine (i, j) below for $j = 1, \ldots, n - [i = m-1]$.

Q3. [Sum.] The answer is $\sum\{c(x_0, \ldots, x_n) \mid 0 \leq x_0, \ldots, x_n < t\}$. ▌

Subroutine (i, j) is the following: Set $q \leftarrow (j - i) \bmod (n+1)$. For all t^n choices of $(x_0, \ldots, x_n)$ such that $x_q = 0$, compute t sums for $0 \leq d < t$, namely

$$s_d \leftarrow \sum_{0 \leq k < t} \bigl[r_{ij}(k, x_{(q+1) \bmod (n+1)}, x_{(q-1) \bmod (n+1)}, d)\bigr] c(x_0, \ldots, x_{q-1}, k, x_{q+1}, \ldots, x_n);$$

tableau shape
rim representation
Gessel
coordinates
matrix coordinates
Cartesian coordinates
structured
dynamic programming
quaternary relation

then set $c(x_0,\ldots,x_{q-1},d,x_{q+1},\ldots,x_n) \leftarrow s_d$ for $0 \le d < t$. (Notice that this computation is rather similar to the discrete Fourier transform in Eq. 4.6.4–(40).)

The relation r_{ij} in the formula for s_d is r when $j < n$; but r_{ij} is the universal relation (always true) when $j = n$. (One could in fact let r_{ij} be a different quaternary relation for each (i,j), where r_{in} constrains the joint values of $(a_{(i-1)(n-1)}, a_{i0}, a_{i(n-1)}, a_{(i+1)0})$. Imagine the $2 \times (m-1)n$ matrix $\binom{a_{00}a_{01}\ldots a_{0(n-1)}a_{10}\ldots a_{1(n-1)}a_{20}\ldots}{a_{10}a_{11}\ldots a_{1(n-1)}a_{20}\ldots a_{2(n-1)}a_{30}\ldots}$!)

The method works because, when subroutine (i,j) begins, $c(x_0,\ldots,x_n)$ is the number of ways to set the initial matrix entries $a_{i'j'}$, for (i',j') lexicographically less than (i,j), so that all constraints on those variables are satisfied and

$$(a_{(i-1)(j-1)},\ldots,a_{(i-1)(n-1)},a_{i0},\ldots,a_{i(j-1)}) = (x_q,x_{q+1},\ldots,x_n,x_0,\ldots,x_{q-1}).$$

About 1.8 teramems of computation suffice to show that the desired number of 8×8 matrices is 1,927,084,607,409,168,698,157,388,476,170,741,096,757,035,906,066. (Those "mems" were however longer than usual, because 24 gigabytes of memory were needed.)

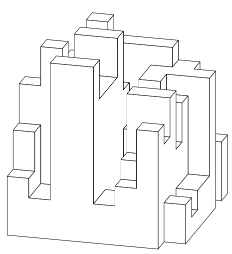
29. We essentially want to run that algorithm in reverse. To reverse step Q3, let the counts $c(x_0,\ldots,x_n)$ be renamed c_j for $0 \le j < t^{n+1}$, in any convenient way. Then for $j = 0, 1, \ldots,$ set $k \leftarrow k - c_j$ if $k \ge c_j$; but stop when $k < c_j$. That gives us suitable values of $(x_0,x_1,\ldots,x_n)$, which will be $(a_{(m-2)(n-1)}, a_{(m-1)0},\ldots,a_{(m-1)(n-1)})$. And we'll want the kth solution for which those $n+1$ values are prespecified.

Similarly, we can run subroutine (i,j) in reverse, if we're given the t^{n+1} counts that it ends with, because each of those counts was obtained as the sum of at most n counts c_j whose sum exceeds k. That will give us enough information to determine $a_{(i-1)(j-1)}$, as well as a new value of k. The remaining problem is then to find the kth solution when the final $(m-i+1)n-j-1$ elements are given.

We must rerun the algorithm for each $(i,j) = (m-1,n-1), (m-1,n-2), \ldots, (1,1)$, because the previous counts have been discarded. However, we can save time by cleverly omitting the computation of counts that won't contribute to solutions having the prespecified final elements. (See the author's program HISTOSCAPE-UNRANK.)

The "random" 8×8 solution shown here was found by setting $k \leftarrow N/\phi$, where N is the total number of solutions. (It can be fabricated from sugar cubes.)

$$\begin{pmatrix} 5&4&2&3&2&2&6&3\\ 6&7&9&8&8&8&7&0\\ 5&1&1&6&4&7&7&7\\ 5&3&9&9&1&7&6&2\\ 5&4&5&7&1&1&4&2\\ 7&9&6&7&1&7&7&1\\ 5&2&5&7&2&4&5&2\\ 3&2&9&9&2&3&6&0 \end{pmatrix}$$



Incidentally, this histoscape has 184 vertices and 94 faces. Only 89 of the vertices are visible in this particular view, and only 48 of the faces are at least partly visible. There are 35 T junctions, 24 V junctions, 42 W junctions, and 23 Y junctions. When half edges are forced at the boundary, the line labeling problem has six solutions, because of two independent ambiguities in the "central canyon"; all but four labels are forced.

30. It's convenient to use the even/odd coordinate system of exercise 7.2.2.1–145, with cubie (i,j,k) represented by $(2i+1, 2j+1, 2k+1)$. In the following description we shall use the notation $\bar{k}$ to stand for $k \bmod 2$. Assume that $a_{ij} < t$ for all i and j, and set up a $(2m+1)\times(2n+1)\times(2t+1)$ array b, initially zero.

discrete Fourier transform
Fourier transform
universal relation
lexicographically less
mems
author
downloadable programs
golden ratio
junctions
half edges
boundary
line labeling
even/odd coordinate system

First, mark all the cubies, by setting $b_{(2i+1)(2j+1)(2k+1)} \leftarrow 1$ for $0 \le k < a_{ij}$.

Second, mark all the "visible" faces of cubies, by doing the following for all (i,j,k) with $\bar{\imath}\bar{\jmath}\bar{k} = 111$ and $b_{ijk} = 1$: If $b_{(i\pm2)jk} = 0$, set $b_{(i\pm1)jk} \leftarrow 1$; if $b_{i(j\pm2)k} = 0$, set $b_{i(j\pm1)k} \leftarrow 1$; if $b_{ij(k\pm2)} = 0$, set $b_{ij(k\pm1)} \leftarrow 1$. (We assume that $b_{ijk} = 0$ whenever $i < 0$ or $j < 0$ or $k < 0$ or $i > 2m$ or $j > 2n$ or $k > 2t$.)

Third, to mark all the "visible" edges, do the following for all (i,j,k) with $\bar{\imath}\bar{\jmath}\bar{k} = 011$ and $b_{ijk} = 1$: If $b_{i(j\pm2)k} = 0$, set $b_{i(j\pm1)k} \leftarrow 1$; if $b_{ij(k\pm2)} = 0$, set $b_{ij(k\pm1)} \leftarrow 1$. Also do this, for all (i,j,k) with $\bar{\imath}\bar{\jmath}\bar{k} = 101$ and $b_{ijk} = 1$: If $b_{(i\pm2)jk} = 0$, set $b_{(i\pm1)jk} \leftarrow 1$.

Fourth, mark all the vertices, by doing the following for all (i,j,k) with $\bar{\imath}\bar{\jmath}\bar{k} = 001$ and $b_{ijk} = 1$: If $b_{ij(k\pm2)} = 0$, set $b_{ij(k\pm1)} \leftarrow 1$.

Finally, now that we know the vertices, we're ready to output the face polygons (some of which might be "holes" enclosed in a larger polygon). Every vertex will be part of three polygons, one with constant i, another with constant j, another with constant k. All three cases are similar; the polygon with constant i can be found as follows, starting at ijk where $\bar{\imath}\bar{\jmath}\bar{k} = 000$: "While $b_{ijk} = 1$, do a j-step and a k-step." A j-step means, "Output vertex $(i/2)(j/2)(k/2)$; set $b_{ijk} \leftarrow 2$; set $\delta \leftarrow 2$ if $b_{i(j+1)k} > 0$, otherwise $\delta \leftarrow -2$; repeat $j \leftarrow j + \delta$ until $b_{ijk} > 0$." A k-step is similar. (The polygon will have an even number of vertices, because we alternate j-steps with k-steps.) After all faces with constant i have been output, all vertices will have $b_{ijk} = 2$.

For example, consider the histoscape for $\left(\begin{smallmatrix}1&1&1\\1&0&1\\1&1&1\end{smallmatrix}\right)$. It has 16 vertices: 000, 001, 030, 031, 110, 111, 120, 121, 210, 211, 220, 221, 300, 301, 330, 331. Its i-face polygons are 000 — 030 — 031 — 001 — 000, 110 — 120 — 121 — 111 — 110, 210 — 220 — 221 — 211 — 210, 300 — 330 — 331 — 301 — 300; its j-face polygons are 000 — 001 — 301 — 300 — 000, 030 — 031 — 331 — 330 — 030, 110 — 111 — 211 — 210 — 110, 120 — 121 — 221 — 220 — 120; its k-face polygons are 000 — 300 — 330 — 030 — 000, 001 — 301 — 331 — 031 — 001, 110 — 210 — 220 — 120 — 110, 111 — 211 — 221 — 121 — 111. It looks like a square torus.

31. (a) Swap 14 with 15.

(b) Swapping adjacent elements of a vortex changes it to a non-vortex. (Moreover, the 2×2 matrix $\left(\begin{smallmatrix}a&b\\c&d\end{smallmatrix}\right)$ is a vortex if and only if $[a<b]+[b<d]+[d<c]+[c<a]$ is odd.)

(c) First row $(1,\ldots,n)$, second row $(2n,\ldots,n+1)$, and so on.

(d) True, by answer 27 (case 4).

(e) It suffices to verify this for 2×2 matrices, when it's clearly true.

32. Let $r_{ij}(w,x,y,z)$ be any 4-ary relation that depends only on the relative order of four distinct elements $\{w,x,y,z\}$. (There are 2^{24} such relations.) We can enumerate all $m\times n$ matrices whose elements are a permutation of $\{0,1,\ldots,mn-1\}$ and whose 2×2 submatrices satisfy $r_{ij}(a_{(i-1)(j-1)}, a_{(i-1)j}, a_{i(j-1)}, a_{ij})$, with a dynamic programming algorithm structured as the method of answer 28. But this time we need counts $c(x_{n-t},\ldots,x_t)$ for each of the $t^{\underline{n+1}}$ choices of *distinct* elements with $0 \le x_{n-t},\ldots,x_t < t$, where $t = in+j$ when starting subroutine (i,j) and $t = in+j+1$ when finishing. (For example, when $m = n = 5$, the number of counts is only $13^{\underline{6}} = 1235520$ when $(i,j) = (2,2)$, but it rises to $25^{\underline{6}} = 127512000$ during the last round when $(i,j) = (4,4)$.)

Two ideas make it possible to represent these numerous counts efficiently in memory. Count $c(x_{n-t},\ldots,x_t)$ is the number of partial solutions $x_0\ldots x_t$ whose final $n+1$ elements are $x_{n-t}\ldots x_t$. Those counts can be represented by $y_{n-t}\ldots y_t$, where y_j is x_j minus the number of elements "inverted" by x_j (namely the smaller elements to its right, as in Section 5.1.1). For example, if $n = 3$ and $t = 8$, the final four elements of a permutation $x_0\ldots x_8$ might be $x_5x_6x_7x_8 = 3142$; we represent them

square torus
torus
vortex
dynamic programming
all different
inversions
pi, as random example

by $y_5y_6y_7y_8 = 1132$. Or, going the other way, if $y_5y_6y_7y_8 = 3141$, then $x_5x_6x_7x_8$ must have been 6251. This representation has the nice property that $0 \le y_j \le j$ for $n-t \le j \le t$, so there clearly are $t^{\underline{n+1}}$ possibilities.

Every permutation $x_0 \ldots x_t$ of $\{0,\ldots,t\}$ yields $t+2$ permutations $x'_0 \ldots x'_{t+1}$ of $\{0,\ldots,t+1\}$, if we choose x'_{t+1} arbitrarily and then set $x'_j \leftarrow x_j + [x_j \ge x'_{t+1}]$. For example, if $t = 8$ and $x_5x_6x_7x_8 = 3142$, the ten permutations obtained from $x_0 \ldots x_8$ will have $x'_5x'_6x'_7x'_8x'_9 =$ 42530, 42531, 41532, 41523, 31524, 31425, 31426, 31427, 31428, or 31429. And the representations $y'_5y'_6y'_7y'_8y'_9$ of those last five elements will simply be respectively 31420, 31421, ..., 31429! In general, we'll have $y'_j = y_j$ for $0 \le j \le t$, and $y'_{t+1} = x'_{t+1}$ will be arbitrary; this inversion-oriented representation works beautifully.

Furthermore, there's a beautiful way to arrange the counts in memory, so that subroutine (i,j) doesn't clobber any of the existing counts when it updates t to $t+1$. These details are all worked out in the author's program WHIRLPOOL-COUNT (online).

The answer to the stated problem is 2,179,875,344,187,129,600 (found in 10 Gμ).

33. (a) If $n > 0$, $2Q_n = 2nU_n$ is the number of permutations $a_0 \ldots a_{2n-1}$ for which $a_{2k-1} < a_{2k} \iff a_{2k} < a_{2k+1}$. Hence Q_n counts those which also have $a_0 < a_1$. The permutations enumerated by U_{n+1} have the form $a_1 \ldots a_{2k}(2n+1)a_{2k+1}\ldots a_{2n}$, for some k, where $a_1 \ldots a_{2k}$ and $a_{2k+1} \ldots a_{2n}$ are independently counted by Q_k and Q_{n-k}.

(b) Hence $U'(z) = Q(z)^2$, where $Q(z) = 1 + U_1z^2/2! + 2U_2z^4/4! + \cdots = 1 + zU(z)/2$. The solution to this differential equation, with $U(0) = 0$, turns out to be slightly scary: $U(z) = \sqrt{2}\tanh(z/\sqrt{2})/(1 - (z/\sqrt{2})\tanh(z/\sqrt{2}))$.

[Let $p_n(k)$ be the number of up-up-or-down-down permutations of the $2n+1$ numbers $\{-n,\ldots,0,\ldots,n\}$ that begin with k. For example, the values $(p_n(-n),\ldots,p_n(n))$ for $1 \le n \le 3$ are $(1,0,1)$; $(4,2,2,2,4)$; $(42,28,22,20,22,28,42)$. Ira Gessel has discovered a surprisingly simple formula for the bivariate exponential generating function

$$\sum_{m,n} p_{(m+n)/2}\bigl(\tfrac{m-n}{2}\bigr)\frac{w^m}{m!}\frac{z^n}{n!} = \frac{\cosh\bigl((w-z)/\sqrt{2}\bigr)}{\cosh\bigl((w+z)/\sqrt{2}\bigr) - \bigl((w+z)/\sqrt{2}\bigr)\sinh\bigl((w+z)/\sqrt{2}\bigr)},$$

by using the fact that these curious numbers satisfy the unusual recurrence relation $p_{n+1}(k) = \sum_{j=-n}^{n} |j-k|\,p_n(j)$. [arXiv:2411.16113 [math.CO] (2024), 8 pages.]

(c) Let $V(z) = 1/(1 - z\tanh z) = 1 + V_1z^2/1! + V_2z^4/3! + \cdots$, where $V_n = 2^{n-1}U_n$, and let μ be the positive number that satisfies $\mu\tanh\mu = 1$. We have $z\tanh z = \sum_{k=0}^{\infty} c_k(z-\mu)^k$ when z is near μ, where $c_0 = \mu\tanh\mu = 1$, $c_1 = \mu + \tanh\mu - \mu\tanh^2\mu = \mu$, and $c_2 = 1 - \mu\tanh\mu - \tanh^2\mu + \mu\tanh^3\mu = 0$. The only other root of $z\tanh z = 1$ for $|z| \le 2\mu$ is $z = -\mu$. Hence the function $V(z) - 2/(\mu^2(\mu^2 - z^2))$ is analytic in $|z| \le 2\mu$; and we have $U_n/(2n-1)! = 2^{1-n}V_n/(2n-1)! = 2^{2-n}/\mu^{2n+2} + O(1/(2\mu)^{2n})$.

The constant μ is a well-studied number called the dual Laplace limit,

$$\mu = 1.19967\,86402\,57733\,83391\,63698\,48641\,14194\,42615-;$$

the even more famous Laplace limit constant $\sqrt{\mu^2-1}$ is

$$\lambda = 0.66274\,34193\,49181\,58097\,47420\,97109\,25290\,70562+.$$

[*Historical notes:* See P. S. Laplace, *Connaissance des Tems de 1828* (1825), 311–321, who thought the value was 0.66195. Cauchy published the correct value of λ to five decimals in an important memoir of 1831, which laid the foundations of complex variable theory; see his *Œuvres complètes* (2) **12** (1916), 101, where he also computed μ and μ^2.]

To get further accuracy, Philippe Jacquet observes that there are constants μ_k with $\mu_k\tan\mu_k = -1$ and $(k-.5)\pi < \mu_k < k\pi$, for all $k \ge 1$; for example, $\mu_1 \approx 2.79839$.

author
downloadable programs
differential equation
Gessel
bivariate exponential generating function
generating function
recurrence relation
dual Laplace limit
fundamental constants
Laplace limit constant
Historical notes
Laplace
Cauchy
complex variable theory
Jacquet

Thus $z = \pm i\mu_k$ is another root of $z \tanh z = 1$ and another pole of the meromorphic function $V(z)$. (Apparently these, together with $z = \pm\mu$, are the *only* poles.)

(d) See the author's note "Whirlpool permutations" (May 2020), available online.

34. To formulate an $m \times n$ whirlpool puzzle as a CSP, there's one variable x_{ij} for each empty cell, having as domain the numbers not yet present; those variables must be all different. Also introduce redundant variables r_{ij} for $0 \le i < m$ and $1 \le j < n$, with binary domains $\{<, >\}$, constrained to describe the result of comparing $x_{i(j-1)} : x_{ij}$. Similarly, c_{ij} describes $x_{(i-1)j} : x_{ij}$, for $1 \le i < m$ and $0 \le j < n$. Finally we constrain $(r_{ij}, c_{ij}, r_{(i-1)j}, c_{i(j-1)})$ to yield a vortex, for $1 \le i < m$ and $1 \le j < n$.

(This setup is easily expressed as an XCC problem. For example, puzzle (iv) has 72 primary items, 44 secondary items, and 1808 options; it is solved in 800 kilomems.)

Puzzles (i) and (iv) have unique solutions. But puzzle (ii) has none; indeed, two entries are required to be 4. Puzzle (iii) has two solutions (one can swap $7 \leftrightarrow 8$).

(i)

1	3	5	7	9
17	16	15	14	13
23	24	25	11	12
22	21	20	19	18
2	4	6	8	10

; (ii)

3	14	15	9	2
4?	6		5	4?

; (iii)

3	14	15	12	13
7	9	2	6	16
5	20	24	23	17
4	1	25	22	18
8	10	11	21	19

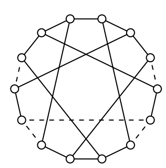
; (iv)

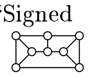
3	13	14	23	15
9	12	2	24	6
10	11	5	8	7
1	20	21	16	25
4	19	18	17	22

.

35. Start with a tetrahedron, and introduce a "crease" in one of its faces, either concave ( ) or convex ( ). That gives us an object with six vertices, nine edges, two triangular faces, and three quadrilateral faces. Now crease a quadrilateral face, between the two triangular faces; that gives us six quadrilateral faces and the desired skeleton:

36. (We've seen this graph before in 7–(57). It's called the Heawood graph, after its discovery by P. J. Heawood [*Quarterly Journal of Pure and Applied Mathematics* **24** (1890), 332–338 and Fig. 16 following 386], and it has 336 automorphisms. At present this is its only known signed skeleton that is realizable as a 3VP, up to automorphism.)



37. Partial results on small graphs are discussed in the author's online note "Signed skeletons" (April 2020). For example, 13 signed realizations of the 8-vertex graph  are known(!), and there may be others. Does the 3-cube have more than four?

38. (a) The determinant is zero if and only if $\{v_0, v_1, v_2, v_3\}$ are coplanar; but they aren't. If it's negative, swap $v_2 \leftrightarrow v_3$. (Hence the cyclic order $(v_1v_2v_3)$ is unique.)

[See F. Joachimsthal, *Crelle* **40** (1850), 21–47, who observed that the volume of the tetrahedron formed by $\{v_0, v_1, v_2, v_3\}$ is $|D(v_0, v_1, v_2, v_3)|/6$. See also J. de la Grange, *Nouveaux Mém. Acad. Sciences et Belles-Lettres* **4** (Berlin: 1773), 85–120, §5.]

(b) $D(v_0, v_1, v_2, v) = 0$.

(c) $D(v_0, v_1, v_2, v) > 0$.

(d) For example use $(o_1o_2o_3)_8$, where $o_1 = [v$ is opposite v_1 with respect to $p_{23}]$, …, $o_3 = [v$ is opposite v_3 with respect to $p_{12}]$. (There's no standard convention for numbering octants; roman numerals are traditionally used in some arbitrary way.)

(e) With those v_i, that method gives octant *0* whenever x, y, z are all positive.

(f) It's now in octant *2*, because $\pi > \phi + \gamma$.

meromorphic function
author
internet
all different
XCC problem
crease
concave
convex
Heawood
automorphisms
author
internet
determinant
coplanar
Joachimsthal
volume
tetrahedron
de la Grange
Lagrange
historical notes
roman numerals

plane in three dimensions
ell

39. (a) A careful case analysis shows that edge $v_0 — v_1$ is concave if and only if X_ϵ intersects octant *3*. Similar conclusions hold for $v_0 — v_2$ with respect to *5*, and for $v_0 — v_3$ with respect to *6*.

(b) For example, if θ is the angle at edge $v_0 — v_1$, we have $(v_2 - v_0)\cdot(v_3 - v_0) = \|v_2 - v_0\|\|v_3 - v_0\|\cos\theta$. Choose $0 < \theta < 180^\circ$ if concave, otherwise $180^\circ < \theta < 360^\circ$.

40. First, if (x, y, z) is a vertex of X, there must be no edge containing a point (x, y, z') with $z \neq z'$. (In particular, there must be no vertex (x, y, z') with $z \neq z'$.)

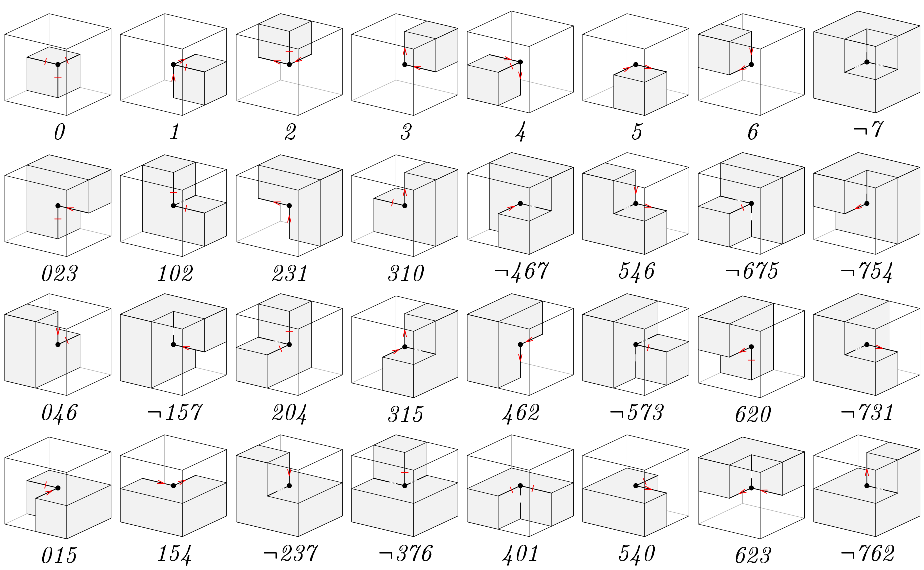
Second, X mustn't contain noncollinear edges whose projections are collinear. For example, if the line segment $\{(t, 0, 0) \mid 0 \le t \le 1\}$ is an edge of X, there shouldn't also be an edge of the form $(u, 0, u)$. Quantitatively, each edge has the form $\{(x_0 + \alpha t, y_0 + \beta t, z_0 + \gamma t) \mid t_0 \le t \le t_1\}$ for some $(\alpha, \beta, \gamma) \neq (0, 0, 0)$; by the first assumption, we have in fact $(\alpha, \beta) \neq (0, 0)$. Distinct edges must not have $\alpha\beta' = \alpha'\beta$.

[Consequently X has no faces perpendicular to the (x, y) plane. Indeed, every plane in three dimensions is characterized by an equation of the form $ax + by + cz = d$, where a, b, and c are not all zero. Since adjacent edges of a face aren't collinear, the equation for its plane must have $c \neq 0$. Hence we may assume that $c = 1$.]

41. (a) There obviously are 8 cases with one cubie. Three cubies that make the "ell" tricube can be placed in 24 ways. Five cubies whose complement is an "ell" can also be placed in 24 ways. Seven cubies can be placed in 8 ways. An even number of cubies can't make a 3VP with the center as vertex. Total, $8 + 24 + 24 + 8 = 64$. (Incidentally, a solution with (1, 3, 5, 7) cubies has respectively (0, 1, 2, 3) concave edges at the center.)

(b) Only the cubie in the corner closest to the camera obscures the center.

(c) This chart shows the octants that contain cubies, when octant *7* is closest:



(Notice that the rotation $x \mapsto y \mapsto z \mapsto x$ always gives an equivalent junction pattern.) By exercises 38 and 39, the possible labels of a V, W, or Y junction in an HC picture depend only on which octants adjacent to the corresponding vertex are occupied.

(d) By definition, the two "bars" of a T must be half edges that point left.

42. $(3t + 2v + 3w + 3y)/2$ variables and $t + v + w + y$ constraints.

van Beek
unrealizable
Gardner
$K_{3,3}$
interior angles
complexity

43. (a) $(\mathrm{a},\mathrm{b}) \in \{41, 51, 33, 62\}$, where '11' abbreviates $(1,1)$, etc.

(b) $(\mathrm{n},\mathrm{p}) \in \{12, 13, 22, 23, 32, 33, 42, 43\}$; $(\mathrm{o},\mathrm{p}) \in \{13, 23, 36\}$.

(c) $t+v+w+y$ variables and $(3t+2v+3w+3y)/2$ constraints (role reversal!).

(d) The text's model has the nice feature that it allows us to deduce some labels immediately (see (24) and (25)). Although we can deduce $\mathrm{p}=3$ from the two constraints in part (b), the corresponding inference from (22) is just as easy. The total size of the new state space, $4^t 6^v 3^w 5^y$, does however tend to be quite a bit smaller than $4^{(3t+2v+3w+3y)/2}$; the ratio is $(1/2)^t(3/2)^v(3/8)^w(5/8)^y$, which is $\approx .00014$ in example (20). Computational experience is generally advisable when choosing between models, because different models typically suggest different branching heuristics. [See P. van Beek, *AAAI Conf.* **10** (1992), 447–452, Example 3; see also exercise 56.]

44. With 19 primary items $\{\mathrm{a},\mathrm{b},\ldots,\mathrm{s}\}$ and 26 secondary items $\{\mathrm{ab},\mathrm{ac},\ldots,\mathrm{rs}\}$ (see (21)), the options are 'a ab:< ac:+', 'a ab:< ac:>', ..., 's rs:+ ls:- qs:+', as in exercise 7.2.2.1–100. (In general, continuing exercise 42, there will be $t+6v+3w+5y$ options.)

45. Change the lower Y labels to '---'. (The "hole" is now filled in.)

46. Whenever j is $T(l,m,r)$ or $V(l,r)$ or $W(l,m,r)$ or $Y(a,b,c)$ in H, j is respectively $T(r,m,l)$ or $V(r,l)$ or $W(r,m,l)$ or $Y(c,b,a)$ in H^R. (This rule defines H^R also in cases where H is unrealizable as an HC picture.)

Notice that H and H^R have the same variables and the same domains, but different relations. The values $x_1 \ldots x_n$ solve H if and only if $x_1^R \ldots x_n^R$ solves H^R, where $\mathtt{+}^R = \mathtt{+}$, $\mathtt{-}^R = \mathtt{-}$, $\mathtt{<}^R = \mathtt{>}$, $\mathtt{>}^R = \mathtt{<}$. (For example, in the reflection of (20) we have $\mathrm{a} = V(\mathrm{c},\mathrm{b})$; the corresponding constraint is $(\mathrm{ac},\mathrm{ab}) \in \{\mathtt{<+}, \mathtt{<>}, \mathtt{+>}, \mathtt{>-}, \mathtt{><}, \mathtt{-<}\}$, which is the same as $(\mathrm{ab},\mathrm{ac}) \in \{\mathtt{+<}, \mathtt{><}, \mathtt{>+}, \mathtt{->}, \mathtt{<>}, \mathtt{<-}\}$, which is the same as $(\mathrm{ab},\mathrm{ac}) \in \{\mathtt{>+}, \mathtt{><}, \mathtt{+<}, \mathtt{<-}, \mathtt{<>}, \mathtt{->}\}$.)

[People often say that mirror reflection interchanges left and right, but not top and bottom. Martin Gardner explains why in his book *The Ambidextrous Universe*.]

47. (a) For example, $\mathrm{a} = V(\mathrm{b},\mathrm{c})$, $\mathrm{b} = V(\mathrm{c},\mathrm{a})$, $\mathrm{c} = V(\mathrm{b},\mathrm{a})$.

(b) H is realizable if and only if each of its connected components is realizable. If H is connected and its junctions $\{j_0, j_1, \ldots, j_{p-1}\}$ all have type V, we can assume that $j_k = V(j_{k+\sigma_k}, j_{k-\sigma_k})$ for $0 \le k < p$, with subscripts treated mod p, where each σ_k is ± 1. When $p=3$, we must have $\sigma_0 = \sigma_1 = \sigma_2$. When $p=4$, we must not have $\sigma_0 \ne \sigma_1 \ne \sigma_2 \ne \sigma_3$. When $p>4$, we can assume (by switching to H^R if necessary) that $\sigma_0 = \sigma_k = \sigma_l = +1$ for some $0<k<l<p$. Then H is realized by putting j_0, j_k, j_l at the vertices of a triangle, and placing the intervening junctions at roughly equidistant positions near the intervening edges of that triangle — perturbing them slightly so that each junction is convex or concave as desired, when seen from outside the triangle.

(c) The graph of $\mathrm{a}=\mathrm{b}=\mathrm{c}=W(\mathrm{d},\mathrm{e},\mathrm{f})$, $\mathrm{d}=\mathrm{e}=\mathrm{f}=W(\mathrm{a},\mathrm{b},\mathrm{c})$ is $K_{3,3}$.

(d) The interior angles of a polygon with m vertices sum to $(m-2)180°$; hence at most $m-3$ of them are greater than $180°$.

(e) True. (Just jiggle the junction a little bit.)

48. If indeed this question is recursively decidable, what is its complexity?

49. (a) Each "level" has a sequence of junctions $j_1 = W(j_0, j_1', j_2)$, $j_2 = Y(j_1, j_2', j_3)$, $j_3 = W(j_2, j_3', j_4)$, ..., $j_9 = W(j_8, j_9', j_{10})$, whose connecting lines $j_0 j_1$, $j_1 j_2$, ..., $j_9 j_{10}$ must all be given the same label: either + or - or <. The standard boundary forces the labels <<<<<<<<<< at the bottom, but ---------- on the other levels. These, in turn, immediately force the labels in their vicinity, so the standard labeling is unique.

(b) Similarly, junctions of the form $j_0 = V(j_0', j_1)$, $j_1 = W(j_0, j_1', j_2)$, $j_2 = Y(j_1, j_2', j_3)$, ..., $j_6 = Y(j_5, j_6', j_7)$, $j_7 = W(j_6, j_7', j_8)$, $j_8 = V(j_7, j_8')$, which appear upwards at the right and downwards at the left, force the labels from j_0 to j_8 to be either `++++++++` or `--------` or `<<<<<<<<`. But `++++++++` is excluded, because it doesn't combine with `++++++++++` or `----------` or `<<<<<<<<<<` at the bottom.

Let the V junctions at the top be $t_1 = V(t_2, t_0)$, $t_2 = V(t_1, t_3)$, $t_3 = V(t_4, t_2)$, $\ldots$, $t_6 = V(t_5, t_7)$, $t_7 = V(t_8, t_6)$. We know that t_0t_1 must be `-` or `>`; the same holds for t_7t_8. Hence the legal labelings from t_0 to t_8 are `->+<->>-`, `->+<->>>`, `->+<<+>-`, `->+<<+>>`, `->+<<<->`, `->>-<+>-`, `->>-<+>>`, `->>-<<->`, `->>>+<->`, `->>>>>>-`, `->>>>>>>`, `>-<+>>>-`, `>-<+>>>>`, `>-<<->>-`, `>-<<->>>`, `>-<<<+>-`, `>-<<<+>>`, `>-<<<<->`, `>>+<->>-`, `>>+<->>>`, `>>+<<+>-`, `>>+<<+>>`, `>>+<<<->`, `>>>-<+>-`, `>>>-<+>>`, `>>>-<<->`, `>>>>+<->`, `>>>>>>>-`, `>>>>>>>>`: 4 from `-` to `-`, 7 from `-` to `>`, 7 from `>` to `-`, and 11 from `>` to `>`. The latter can be used with either `----------` or `<<<<<<<<<<` at the bottom. So the total number of boundary labelings is $4 + 7 + 7 + 11 + 11 = 40$.

(c) There are exactly 40, because each of those 40 boundary-only solutions imposes exactly the same constraints on the lines touching the boundary. [P. H. Winston, who presented this picture as Fig. 3-17 in the second edition of his book *Artificial Intelligence* (Addison–Wesley, 1984), noted that "The background border contributes considerable constraint to line-drawing analysis efforts." He may not have been aware, however, that *any* border constrains the interpretation of the interior in the same way!]

50. (a) These possibilities are essentially forced by the definition of boundary cycle.

(b) In each of the following 4×4 matrices, the rows and columns are indexed by (`>`, `<`, `-`, `+`), where rows represent the label of $j_{k-1}j_k$ and columns represent the label of j_kj_{k+1}. The entry is 0 when the row/column labels are illegal; otherwise it is 1 when there's no junction j'_k; otherwise it is (g, l, m, p) when $j_kj'_k$ must be labeled (`>`, `<`, `-`, `+`).

$$\boldsymbol{L} = \begin{bmatrix} l&0&0&0\\ l&0&0&0\\ l&0&0&0\\ l&0&0&0 \end{bmatrix},\ \boldsymbol{R} = \begin{bmatrix} g&g&g&g\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0 \end{bmatrix},\ \boldsymbol{V} = \begin{bmatrix} 1&0&1&0\\ 0&1&0&1\\ 1&0&0&0\\ 0&1&0&0 \end{bmatrix},\ \boldsymbol{\Lambda} = \begin{bmatrix} 1&0&0&1\\ 0&1&1&0\\ 0&1&0&0\\ 1&0&0&0 \end{bmatrix},\ \boldsymbol{W} = \begin{bmatrix} p&0&0&0\\ 0&0&0&0\\ 0&0&p&0\\ 0&0&0&m \end{bmatrix},\ \boldsymbol{Y} = \begin{bmatrix} m&0&0&0\\ 0&0&l&0\\ 0&g&m&0\\ 0&0&0&p \end{bmatrix}.$$

(c) Multiply the matrices of the boundary cycle together, treating $\{g, l, m, p\}$ as noncommuting variables. The diagonal entries of the resulting matrix then specify the permissible labelings of the internal lines. For example, the boundary cycle of (20) gives

$$\boldsymbol{VWYWVWVWYW} = \begin{bmatrix} 3pmpppmp & 0 & 2pmpppmp & 0\\ 0&0&0&0\\ 2pmpppmp & 0 & pmpppmp & 0\\ 0&0&0&0 \end{bmatrix};$$

hence the boundary (in isolation) can be labeled four ways, with $j_{-1}j_0$ labeled `>` in three cases and `-` in the other. (The sum of diagonal elements is called the "trace.") In all four cases the interior labels are respectively `+-+++-+`; hence (20) has a free boundary.

This free-boundary-testing algorithm needn't implement arithmetic on string polynomials in full generality. For each matrix entry, it needs to remember only whether that entry is (i) zero, (ii) a multiple of a certain string σ, or (iii) mixed. At the end, the boundary is free if and only if the sum of the four diagonal elements isn't mixed.

51. The trace of the boundary cycle matrix product $\boldsymbol{W\Lambda}^n\boldsymbol{WV}^n$ of exercise 50 is $F_{n+1}^2pp + 0 + F_{n-1}^2pp + F_{n-1}^2mm$. Therefore, to complete the labeling, we need to consider a sequence of n V junctions, preceded and followed by the same sign. That's equivalent to binary strings $1x_1 \ldots x_{n-1}1$ with no two consecutive 1s — of which there are F_{n-1} (see exercise 7.2.1.1–91). Altogether, then, there are $F_{n-1}(F_{n+1}^2 + 2F_{n-1}^2)$ labelings, of which F_{n-1} are standard. (*Nitpicking note:* S_1 has a free boundary, by definition, although it cannot be fully labeled.)

Winston
noncommuting variables
trace
free boundary
string polynomials
consecutive 1s
free boundary

bridge
boundary cycle
trace
organ pipe order, inverse
Kane
operator norm
eigenvalues

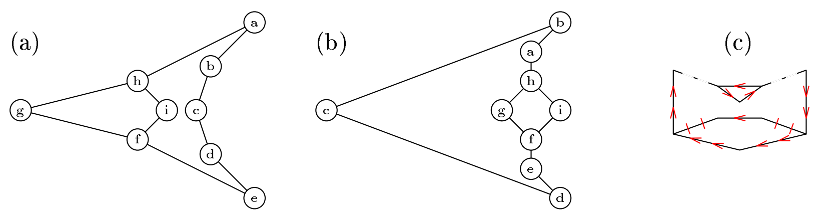


Fig. A–13. Unusual examples of HC pictures.

52. There are (1, 9, 1, 1) labelings with the bridge in the middle labeled (`+`, `-`, `>`, `<`), respectively. (This example shows that an HC picture need not have a boundary cycle consisting of distinct lines.)

53. No: The HC pictures in Figs. A–13(a) and A–13(b) have the same HC network but different cycles. (Consequently the algorithm of exercise 50(c) must be told the boundary cycle as well as the network.)

54. See Fig. A–13(c). (Answer 50(c) gives trace$(\boldsymbol{VYYVWVW}) = 4mmpp + glpp$; so the boundary cycle can be labeled in five ways. Only one of those ways, `><++`, gives usable labels to the inner lines, because a V junction doesn't allow `--`.)

55. (a) Let P be the 4×4 matrix product $j_0 j_1 \ldots j_{q-1}$, and let $M = M_0 M_1 \ldots M_{q-1}$. By induction we can verify that $P_{ij} = P_{(i\oplus 1)(j\oplus 1)}$ for $0 \le i,j < 4$; $P_{00}+P_{01} = F_{q+1}$; $P_{02}+P_{03} = P_{20}+P_{21} = F_q$; $P_{22}+P_{23} = F_{q-1}$; and $M_{ij} = P_{(2i)(2j)} - P_{(2i)(2j+1)}$ for $0 \le i,j < 2$. Hence $\operatorname{trace} P = P_{00}+P_{11}+P_{22}+P_{33} = P_{00}+P_{00}+P_{22}+P_{22}$ and $\operatorname{trace} M = M_{00}+M_{11} = P_{00}-P_{01}+P_{22}-P_{23} = P_{00}-(F_{q+1}-P_{00})+P_{22}-(F_{q-1}-P_{22})$.

(b) The matrix products can be expressed in closed form using the identities

$$A^a = \begin{pmatrix} F_{a+1} & F_a \\ F_a & F_{a-1} \end{pmatrix}, \quad B^b = \begin{pmatrix} F_{b+1} & -F_b \\ -F_b & F_{b-1} \end{pmatrix}, \quad A^a B^b = \begin{pmatrix} \Delta_{a,b} & -\Delta_{a,b-1} \\ \Delta_{a-1,b} & -\Delta_{a-1,b-1} \end{pmatrix},$$

where $\Delta_{a,b} = F_{a+1}F_{b+1} - F_a F_b = \frac{1}{5}\bigl(L_{a+b+1} + 2(-1)^b L_{a-b}\bigr)$. Hence $\Delta_{a,b} - \Delta_{a-1,b-1} = \frac{1}{5}\bigl(L_{a+b} + 4(-1)^b L_{a-b}\bigr)$, and the values of $t_p = \operatorname{trace}(A^p B^{q-p})$ occur in a peculiar order:

$$t_1 < t_3 < \cdots < t_{\lfloor q/2\rfloor} = t_{\lceil q/2\rceil} < \cdots < t_2 < t_0, \quad \text{with } t_p = t_{q-p}.$$

The extremes are $t_p + L_q = 2F_q$ when $p \in \{1, q-1\}$; $t_p + L_q = 2L_q$ when $p \in \{0, q\}$.

(c) (Solution by D. M. Kane.) Note that $B = XAX$, where $X = \bigl(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\bigr)$. Thus M is a product of q As, but with m Xs inserted somehow, where m is the number of switches between $\boldsymbol{V}$ and $\boldsymbol{\Lambda}$ in the cycle. Our goal is to prove that $\operatorname{trace} M \ge 2F_q - L_q = -F_{q-3}$.

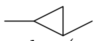
We can assume that $M = (AXA)A^{p_1}(AXA)A^{p_2}\ldots(AXA)A^{p_m}$, where $p_k \ge -1$ for $1 \le k \le m$. If all p_k are nonnegative, $\operatorname{trace} M \ge 0$, because $AXA = \bigl(\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}\bigr)$.

If $p_1 = p_m = -1$, we have $M = ABA^{p_2+2}B^{p_3+2}\ldots B^{p_{m-1}+2}$. And $ABA = BAB$ implies that $\operatorname{trace} M = \operatorname{trace}(BABA^{p_2+1}B^{p_3+2}\ldots B^{p_{m-1}+2}) = \operatorname{trace}(ABA^{p_2+1}B^{p_3+2}\ldots B^{p_{m-1}+3}) = \cdots = \operatorname{trace}(AB^{p_3+1}\ldots B^{p_{m-1}+p_2+4})$, thus reducing m by 2. Therefore we can assume that at least one p_k is -1, but no two -1s are consecutive.

Now let $\|M\|$ be the *operator norm* of M, namely $\sup|Mx|$ over all vectors x of length 1. Then we have $\|A\| = \|AXA\| = \phi$ and $\|AXAXA\| = 1$. Consequently $\|M\| \le \phi^{n-5}$ when $m \ge 4$. (We save a factor of ϕ^2 when $p_k = -1$, ϕ when $p_k \ge 0$.)

Finally, let M have eigenvalues λ and $\hat\lambda$, where $|\lambda| \ge |\hat\lambda|$. Then $\operatorname{trace} M = \lambda + \hat\lambda$, and $\lambda\hat\lambda = \det M = (-1)^q$. So $|\operatorname{trace} M| \le |\lambda| + 1/|\lambda| \le \phi^{n-5} + \phi^{5-n} \le F_{n-3}$, for $n > 6$.

56. Let $D_0 = I$ and $D_{n+1} = D_n A D_n^{RX}$, where ^{R} means left-right reflection and ^{X} means 'change A to B and B to A'. Thus $D_1 = A$, $D_2 = AAB$, $D_3 = AABAABB$, etc. We have $D_n^R = D_n^T$, because $A^T = A$ and $B^T = B$. Hence, using the matrices of answer 55, $D_{n+1} = D_n AXD_n^T X$; and the surprising formula $D_{n+3} = \left(\begin{smallmatrix} 1 & n-1 \\ -1-n & -n^2 \end{smallmatrix}\right)$ arises by induction for $n \geq 0$. Consequently T_n has $\mathrm{trace}(D_{n-1}AD_{n-1}A) + L_{2^n} = L_{2^n} - 1$ labelings, when $n \geq 4$(!). The same formula holds for $n = 3$; but T_2 has 14.

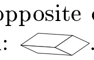
57. . [This is a subpicture of Figure 9(d) in D. A. Huffman's 1971 paper. Examples (24) and (25) come from his Figure 8.]

58. (a) The junctions are $t_k = T(t_{k-1}, t_{k+1}, u_k)$, $u_k = V(w_{k+1}, t_k)$, $v_k = V(w_k, w_{k-1})$, $w_k = W(v_k, u_{k+1}, v_k)$, with subscripts mod n, for $0 \leq k < n$.

(b) The Lucas number L_n. (But only one of these labelings is standard; these networks have a free boundary. Exercise 51 has similar considerations.)

(c) (Solution by K. Sugihara.) The network defines a graph that's uniquely embeddable as an HC picture H in the plane. Suppose H is the projection of some 3VP, X, and let F_k be the face of X that corresponds to the region of H bounded by the polygon $(w_k u_{k-1} t_{k-1} t_k u_k w_{k+1} v_{k+1})$. Let $P = (x, y)$ be a point in H's center region, and let L be the line through P perpendicular to the plane of the picture. Then L intersects F_k at some point (x, y, z_k). Since the edge $u_{k-1}w_k$ is convex, by part (a), we have $z_k > z_{k-1}$. But $z_{n-1} > z_{n-2} > \cdots > z_0 > z_{n-1}$ is impossible.

[See also the discussion by S. W. Draper in *Perception* **7** (1978), 283–296, as well as the comments by Bruno Ernst in Chapter 2 of his book *Adventures with Impossible Figures* (1986). Ernst shows the Penrose square and hexagon, together with a *different* pentagon(!). The Penrose pentagon of the present exercise is #85 in the comprehensive website *Impossible World* by Vlad Alexeev, `https://im-possible.info`, a gallery launched in 2001 that features more than 1000 mind-bending images.]

59. Take a cube and flatten it so that opposite corners are near each other. (Here's a view from the side, only 90% squashed: .) This gives a crumpled object very like a hexagonal tile; you can place such "chips" on a table with any desired overlaps.

Historical notes: A copy of Reutersvärd's original 'Opus 1' is held by Moderna Museet in Stockholm [NMH 42/1981]. It does not show the boxes in general position — the blank region in the middle is a symmetrical "star of David" — so HC picture (32) is slightly different. He told Bruno Ernst in 1986 that he discovered the pattern while doodling during a boring lecture about Latin! [See Figure 1 in Chapter 6 of Ernst's book *Optical Illusions* (1992). Figure 7 in Ernst's Chapter 1 is (26), 'perspective japonaise no. 231 aga', part of a series of more than 2500 artworks now prized by collectors.]

60. The central region has three V junctions, whose left lines can independently be labeled `-` or `<`. Hence there are 8 standard labelings — all realizable as in exercise 59.

There's a free boundary, since each of the corners can be labeled in three ways, and each of the other six in two ways; these $2^6 3^3 = 1728$ boundary labelings all force the same labels inside. So there are $8 \cdot 1728 = 13824$ labelings altogether.

61. Image (i) has a unique standard labeling. But (ii) has $33{,}554{,}432 = 2^{25}$, because each of 25 interior "box tops" has a V junction that can be labeled in two ways.

Image (iii) shows what happens when the 36 cells of the 6×6 hexagonal rhombus are partitioned into three independent sets of 12. One set of twelve boxes is placed in front, another in back. The front ones are labeled uniquely. The back ones are labeled uniquely at the edges, but in five ways when they appear only as a Y in the interior. The middle ones each have two labelings of a W near the edges (except at the very

transpose of matrix
Huffman
Lucas number
free boundary
Sugihara
Draper
Ernst
Alexeev
Historical notes
Reutersvärd
general position
star of David
Ernst
free boundary

bottom), but nine in the interior (when they show up as an unconstrained Y with three Ws). Altogether 11,809,800,000 $= 5^5 2^6 9^5$ standard labelings.

In image (iv) there's clockwise overlapping in the outer loop, enclosing a loop with counterclockwise overlapping; but it's realizable with "squashed boxes." As with the other three images, a large number of T junctions makes the labelings factor into small independent subnetworks, and we find 5,242,880 $= 2^{20}\cdot 5$ standard labelings altogether.

[An interesting mapping was used to draw these images: If x, y, and z are each ± 1, corner (x,y,z) of the box in row i and column j of the array is assigned to point $(6i+j-2y-2z, -i+5j+2x+2z, -5i-6j-2z+2y)$ in barycentric coordinates. (At most seven corners of each box are visible — all except corner $(1,1,-1)$.) With this scheme, all points where the edges of two boxes intersect are distinct, and those points are also distinct from all corner points; thus the images appear in general position.]

62. (a, b) When $m=n=6$ there are 85 Boolean variables, 50 ternary constraints; in general there are $m(n-1)+(m-1)n+(m-1)(n-1)$ Boolean variables and $2(m-1)(n-1)$ ternary constraints. Each constraint has the form $[A<B]+[B<C]+[C<A]\in\{1,2\}$.

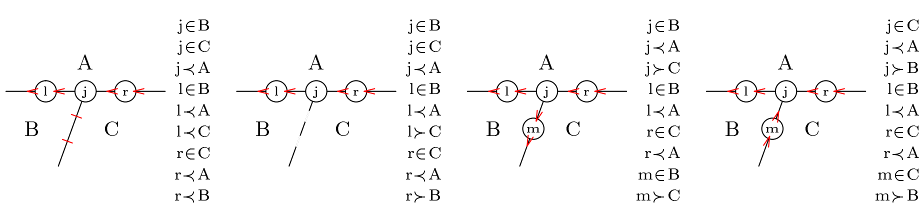
Dynamic programming works well, as in exercise 28, and this problem is considerably easier than that one: Let box (i,j) in row i, column j for $0\le i<m$ and $0\le j<n$ be adjacent to boxes $(i,j+1)$, $(i+1,j)$, and $(i+1,j+1)$; and consider the number $c_n(x_1,\ldots,x_{m-1})$ of $m\times n$ solutions with $x_j=[(i-1,n-1)<(i,n-1)]$. After setting $c_1(x_1,\ldots,x_{m-1})\leftarrow 1$, we can readily compute the 2^{m-1} counts $c_{n+1}(x_1,\ldots,x_{m-1})$ from the 2^{m-1} counts c_n. For example, when $m=3$ we have $c_{n+1}(0,0)=13c_n(0,0)+11c_n(0,1)+9c_n(1,0)+6c_n(1,1)$; $c_{n+1}(0,1)=11c_n(0,0)+12c_n(0,1)+10c_n(1,0)+9c_n(1,1)$; $c_{n+1}(1,0)=c_{n+1}(0,1)$; $c_{n+1}(1,1)=c_{n+1}(0,0)$. (These 2^{2m-3} coefficients are themselves each precomputed in $O(m)$ steps by solving a small-and-simple CSP.)

The total number of solutions is $t_{m,n}=\sum\{c_n(x_1,\ldots,x_{m-1})\mid 0\le x_k\le 1\}$. For example, $(t_{3,1}, t_{3,2}, t_{3,3}, \ldots, t_{3,n}, \ldots)=(4, 162, 6570, \ldots, \lceil cr^n\rceil, \ldots)$, where $r=(41+\sqrt{1609})/2\approx 40.556$ and $c=(1609+31\sqrt{1609})/28962\approx .0985$. We also have $t_{6,6}=22406540276117433798\approx 2^{64.28}$; $t_{10,10}=2333623171515704644702\ldots 99558\approx 2^{193.89}$.

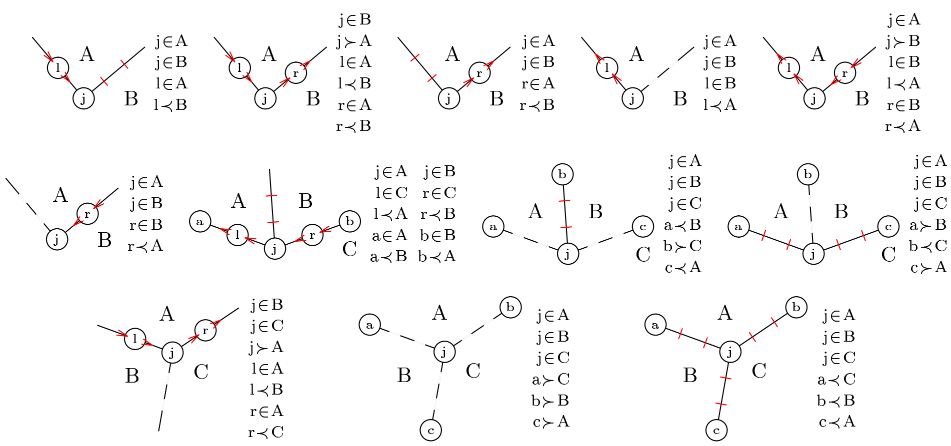
63. Regarding H as an embedded planar graph, let F be the set of its faces. Each $f\in F$ will correspond to part (or all) of some face $\hat f$ of X — except that H's exterior face f_0 will corresponds to a "background plane" $\hat f_0$, which is sufficiently distant that it doesn't conceal any of X.

For each line of H that is labeled < or >, introduce a new "shadow junction" at the midpoint of that line; and let J be the set of all junctions (shadow or not). Each junction j of type V, W, or Y will correspond to a vertex $\hat\jmath$ of X. Every remaining junction j will correspond to an *artificial* vertex $\hat\jmath$, namely the point of X or of the background plane that lies just behind the point (x_j,y_j) of H.

Now use the following chart to establish relations between junctions and faces:



squashed boxes
barycentric coordinates
general position
Dynamic programming
planar graph
background plane

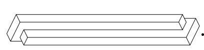


Here '$(j \in A,\ j \prec A,\ j \succ A)$' means "$\hat{\jmath}$ lies (in, behind, in front of) the plane of $\hat{A}$."

To represent those relations linearly, we introduce a real variable z_j for each $j \in J$, meaning that $\hat{\jmath} = (x_j, y_j, z_j)$, where x_j and y_j are given constants. We also introduce three real variables (a_f, b_f, c_f) for each $f \in F$, meaning that plane $\hat{f}$ consists of all (x, y, z) for which $ax+by+z+c = 0$. (See answer 40.) By convention, point (x, y, z) lies behind point (x, y, z') if and only if $z > z'$. Hence $j \in A \iff a_Ax_j + b_Ay_j + z_j + c_A = 0$; $j \prec A \iff a_Ax_j + b_Ay_j + z_j + c_A \geq 1$; $j \succ A \iff a_Ax_j + b_Ay_j + z_j + c_A \leq -1$.

(Actually '> 0' and '< 0' were expected here instead of '≥ 1' and '≤ -1'; but strict inequality is difficult to deal with, in general, while the theory of linear programming handles nonstrict inequality with ease. Fortunately the two notions are equivalent in this case: If there's a solution to the strict inequalities, the nonstrict ones will be satisfied after we multiply all variables $\{a_f, b_f, c_f, z_j\}$ by a suitably large positive constant.)

[This construction is based on Chapter 3 of K. Sugihara's book *Machine Interpretation of Line Drawings* (1986), where a considerably more general problem is treated. It is unknown whether or not this linear system is *sufficient* for a 3VP X to exist.]

64. No — it's strongly realizable as a 3VP. (Start by realizing .)

65. See *J. Computer and System Sciences* **37** (1988), 14–38. The construction is based on D. Lichtenstein's theorem [*SICOMP* **11** (1982), 329–343] that 3SAT is NP-complete even when the clauses are planar and severely restricted.

(The authors show, however, that labelability can be decided in *linear* time if the HC picture arises from an "orthohedral" 3VP, in which every plane face is perpendicular to the x-, y-, or z-axis. For example, a histoscape is orthohedral. In such a case all angles can be assumed to be multiples of 60°. Furthermore, the two entries of Table 1 for which a V junction has a + label can arise only for 60° angles; the other four possibilities for V can arise only for 120° angles.)

66. If indeed this question is recursively solvable, what is its complexity? [Partial results were given by Kirousis and Papadimitriou in the paper just cited. K. Sugihara presented polynomial time necessary and sufficient conditions for strong realizability, in his book cited in answer 63, based on a related but different mathematical model of the problem. Consequently the realizations constructed there aren't 3VP in general.]

67. Replace $(13,14)$ by $(2,13)$ or $(4,14)$ or $(13,2)$ or $(14,4)$ or $(14,13)$. (And to get two more solutions, change either $(12,6,13)$ to $(6,13,7)$ or $(8,6,14)$ to $(6,7,4)$.)

strict inequality
linear programming
Sugihara
strongly realizable
Lichtenstein
3SAT
planar
orthohedral
histoscape
complexity
Kirousis
Papadimitriou
Sugihara

isolated vertices
Domain consistency
bitwise complement
blurred state

68. Almost true (but false when $m = 1$). Given any graceful labeling l, we obtain $2k$ equivalent labelings $l(v\alpha)$ and $m - l(v\alpha)$ when α runs through G's automorphisms. If those labelings aren't distinct, there are automorphisms α and β for which $l(v\alpha) = m - l(v\beta)$ for all v. But then $\beta^{-}\alpha$ would be an automorphism satisfying $l(v\beta^{-}\alpha) = m - l(v)$; that is, complementation would be an automorphism.

That can't happen when $m > 1$: By adding isolated vertices if necessary, we can assume that the vertices are $\{0, \ldots, m\}$ and that $l(v) = v$ for $0 \le v \le m$. The edge labeled m must be $0 — m$, and we can assume that the edge labeled $m-1$ is $1 — (m-1)$. Then m is not adjacent to 1, so complementation isn't an automorphism.

69. (a) For example, eliminate all options with $l(\texttt{NY}) > l(\texttt{MA})$ or $l(\texttt{GA}) > l(\texttt{SC})$. (Then 5814 options remain, and the running time goes down to 33 gigamems.)

(b) Add a new primary item '`*`' and the new option '`* GA`:0 `SC`:18 `NJ`:5'. (The search tree now has 192 nodes. The algorithm of exercise 100 solves it with 62 nodes. Domain consistency is much more expensive but prunes the tree to only 23 nodes.)

70. $\texttt{LO}'[l] = m - l - \texttt{LO}[l]$, $\texttt{NAME}'[l] = \texttt{NAME}[m-l]$, $\texttt{FIRST}'[l] = m - \texttt{FIRST}[m-l]$, for $0 \le l \le m$; $\texttt{NEXTL}'[l] = m - \texttt{NEXTH}[l]$, $\texttt{NEXTH}'[l] = m - \texttt{NEXTL}[l]$, for $1 \le l \le m$; but change $m+1$ to -1. (Other settings of $\texttt{FIRST}'$, $\texttt{NEXTL}'$, $\texttt{NEXTH}'$ are also possible.)

71. The first four real vertices can't be $\{0, m-2, m-1, m\}$ or $\{0, 1, m-1, m\}$, because only one edge can be labeled 1. Hence they are $\{0, 2, m-1, m\}$; and $\texttt{LO}[m-3] = 2$. That forces $\texttt{LO}[m-4] = 0$, leaving no choices for $\texttt{LO}[m-5]$.

72. The key idea is to have a good way to represent the partial path fragments formed by the already-chosen edges. If l is an unchosen vertex label, let $\texttt{MATE}[l] = l$; if l is chosen and the endpoint of a partial subpath, let $\texttt{MATE}[l]$ be the other endpoint; otherwise let $\texttt{MATE}[l]$ be the bitwise complement of the value it had when it was most recently an endpoint, during the backtracking. For example, the MATE table $(\texttt{MATE}[0], \ldots, \texttt{MATE}[5])$ at node '405' of (38) is $({\sim}5, 1, 2, 3, 5, 4)$; at node '4052,13' it's $({\sim}5, 3, 4, 1, 2, {\sim}4)$.

P1. [Initialize.] Set $\texttt{MATE}[l] \leftarrow l$ for $0 \le l < n$, then set $l \leftarrow 1$.

P2. [Enter level l.] If $l = n$, visit a solution and go to P5. Otherwise set $v \leftarrow 0$.

P3. [Try $\texttt{LO}[n-l] = v$.] Set $w \leftarrow v + n - l$, $v' \leftarrow \texttt{MATE}[v]$, $w' \leftarrow \texttt{MATE}[w]$. Go to P4 if $v' < 0$ or $w' < 0$ or $v' = w$. Otherwise set $\texttt{LO}[n-l] \leftarrow v$, $\texttt{MATE}[v] \leftarrow {\sim}v'$, $\texttt{MATE}[w] \leftarrow {\sim}w'$, $\texttt{MATE}[v'] \leftarrow w'$, $\texttt{MATE}[w'] \leftarrow v'$, $l \leftarrow l+1$, and return to P2.

P4. [Try again.] Set $v \leftarrow v+1$. If $v < l$ and $l > 2$, go to P3.

P5. [Backtrack.] Set $l \leftarrow l-1$, and terminate if $l = 0$. Otherwise set $v \leftarrow \texttt{LO}[n-l]$, $w \leftarrow v+n-l$, $v' \leftarrow \texttt{MATE}[v]$, $w' \leftarrow \texttt{MATE}[w]$. If $v' \ge 0$ set $\texttt{MATE}[v] \leftarrow v$; otherwise set $\texttt{MATE}[v] \leftarrow {\sim}v'$ and $\texttt{MATE}[{\sim}v'] \leftarrow v$. If $w' \ge 0$ set $\texttt{MATE}[w] \leftarrow w$; otherwise set $\texttt{MATE}[w] \leftarrow {\sim}w'$ and $\texttt{MATE}[{\sim}w'] \leftarrow w$. Return to P4. ▌

73. A "blurred state" is obtained from MATE when all the negative entries are replaced by '$-$'. For example, 1738092 and 1809372 both have $(-, 2, 1, -, 4, 5, 6, -, -, -)$ as their blurred state. With a suitable hashing scheme we can maintain a dictionary of all the distinct blurred states that arise during the search.

We also maintain a list of branch specs (v_p, β_p, o_p) for $p = 1, 2, \ldots$; here v_p is a value of LO; β_p is the blurred state if v_p is chosen; and o_p is the branch when v_p isn't. If α represents a blurred state, $\texttt{FIRST}(\alpha)$ represents its first branch and $\texttt{LOC}(\alpha)$ represents the corresponding output node. Both FIRST and LOC are 0 unless changed.

In step P1, set $p \leftarrow 0$ and α_1 to the initial blurred state.

In step P2, "visit" a solution by setting $\texttt{LOC}(\alpha_l) \leftarrow 1$.

At the end of step P3, do the following just before returning to P2: Set α_l to the current blurred state, and set $p \leftarrow p+1$, $v_p \leftarrow v$, $\beta_p \leftarrow \alpha_l$, $o_p \leftarrow \texttt{FIRST}(\alpha_{l-1})$, and $\texttt{FIRST}(\alpha_{l-1}) \leftarrow p$. If α_l has occurred before, jump to the second sentence of step P5.

Finally, after backtracking is complete, we can transform the branch specs into something like a ZDD with the following procedure: "Set $z \leftarrow 2$, $s \leftarrow 1$, $\beta_1 \leftarrow \alpha_1$, $o_1 \leftarrow 0$, $\texttt{LOC}(\alpha_1) \leftarrow z$. While $s \neq 0$ do the following: "Set $p \leftarrow \texttt{LOC}(\beta_s)$, $q \leftarrow \texttt{FIRST}(\beta_s)$, $s \leftarrow o_s$. While $q \neq 0$ do the following: "Set $q' \leftarrow o_q$, $\alpha \leftarrow \beta_q$. If $\texttt{LOC}(\alpha) = 0$ and $\texttt{FIRST}(\alpha) \neq 0$, set $o_q \leftarrow s$, $s \leftarrow q$, $\texttt{LOC}(\alpha) \leftarrow z$, $z \leftarrow z+1$. If $q' \neq 0$, output $I_p = (\bar{v}_q?\ z\colon \texttt{LOC}(\alpha))$ and set $p \leftarrow z$, $z \leftarrow z+1$; otherwise output $I_p = (\bar{v}_q?\ 0\colon \texttt{LOC}(\alpha))$. Set $q \leftarrow q'$." " "

The output isn't necessarily a true ZDD: Its "variables" have to be understood correctly, it isn't necessarily reduced, and its instructions can sometimes have the form $I_p = (\bar{v}?\ 0\colon 0)$. But many algorithms that manipulate ZDDs will handle it correctly. For example, the algorithm of exercise 7.1.4–208 will count the total number of solutions.

Equivalent nodes occur only on the same level, so it might seem that a breadth-first search is needed. But this method coexists nicely with (depth-first) backtracking.

This exercise is based on the ideas of M. Adamaszek [*J. Combin. Math. Combin. Computing* **87** (2013), 191–197], who was the first to enumerate graceful permutations for $20 < n \leq 40$. It gives a tremendous speedup over exercise 72; for example, when $n = 30$ the running time decreases from 25 teramems to 34 megamems!

[*Historical notes:* Graceful permutations were implicitly introduced by J. Abrham and A. Kotzig, *Cong. Numerantium* **72** (1990), 163–174, who proved that they have exponential growth. T. Kløve, *IEEE Trans.* **IT-41** (1995), 279–283, considered them independently and used them to design certain error-correcting codes. See J. McGill and M. A. Ollis, *Discrete Math.* **342** (2019), 793–799, for further developments.]

74. (a) Let l_1 and l_2 be the longest distinct segment lengths. If we perturb each point by less than $|l_1 - l_2|/(2n)$, we change the path length by less than $|l_1 - l_2|$. So we may assume that the points $p_1 \ldots p_n = (x_1, 0) \ldots (x_n, 0)$ of a longest path have *distinct* x's.

The path can't be longest if $\max(x_{i-1}, x_i) < \min(x_j, x_{j+1})$ or if $\min(x_{i-1}, x_i) > \max(x_j, x_{j+1})$ for some $1 < i < j < n$: $p_1 \ldots p_{i-1} p_j \ldots p_i p_{j+1} \ldots p_2 p_n$ would be longer.

Let S be the $\lfloor n/2 \rfloor$ points with *smallest* x's, and let T be the other $\lceil n/2 \rceil$ points. No two points of S can be consecutive in the path; otherwise there would also be two consecutive points of T. Hence we can assume that $S = \{x_2, x_4, \ldots, x_{2\lfloor n/2 \rfloor}\}$.

The maximum path length is therefore $(x_1 - x_2) + (x_3 - x_2) + (x_3 - x_4) + \cdots = 2\sum T - 2\sum S - x_1 + x_n[n \text{ even}]$, where x_1 is the smallest x in T and x_n is the largest x in S.

Similarly, the longest *cycle* $(p_1 \ldots p_n)$ has length $2\sum T - 2\sum S - 2x_1[n \text{ odd}]$.

(b) A graceful permutation with $p_n = p_1 + m$ yields a cycle $(p_1 \ldots p_n)$ of length $1 + \cdots + (2m-1) + (p_n - p_1) = 2m^2$, which is maximum because $\sum T - \sum S = m^2$.

(c) The path length is $2\sum T - p_n - 2\sum S + p_1 = 1 + \cdots + (2m-1) = 2m^2 - m$.

75. There are $m = 2n+1$ edges. Call S a (d,m)-set if $S \cup \{\, |k-d| \mid k \in S \,\} \cup \{d\} = \{1, \ldots, m\}$. A canonical graceful labeling of $K_{1,1,n}$ has vertices 0 and d in the first two parts, where $1 \leq d \leq m$, and the vertices S of the third part are a (d,m)-set. Furthermore, we require that $1 \notin S$ if $d = m$, to rule out the complementary labeling.

There clearly is no (d,m)-set with $d > m$. But there are $2^{(m-1)/2}$ (m,m)-sets, because S must contain 1 or $m-1$, 2 or $m-2$, …, $\lfloor m/2 \rfloor$ or $\lceil m/2 \rceil$.

There's no (d,m)-set when $\lceil m/2 \rceil < d < m$. For S would have to contain m, $m-1$, …, $d+1$, and then there'd be no way to get edge $d-1$.

There's a unique $(\lceil m/2 \rceil, m)$-set, namely $S = \{m, m-1, \ldots, \lceil m/2 \rceil + 1\}$.

breadth-first vs depth-first
depth-first vs breadth-first
Adamaszek
Historical notes
Abrham
Kotzig
Kløve
McGill
Ollis
longest *cycle*

recursion
breaks reflection symmetry
Historical notes
Petrie
Smith
symmetry
Smith
Puget
KP
KC
unique
KC graphs

Finally, if $d < \lceil m/2 \rceil$, a (d,m)-set S must be $\{m, m-1, \ldots, m-d+1\} \cup S'$, where S' is a $(d, m-2d+2)$-set. (An interesting recursion!)

So the total number of solutions is $\sum_{d\backslash m} 2^{(d-1)/2} + \sum_{d\backslash n+1} 1 - 2^{n-1} - 1 - 2[n=1]$.

76. If $1 \mathrel{-\!\!-} m$, change each x_{ij} to $m - x_{ij}$. Then if $\min\{x_{11}, \ldots, x_{n1}\} > \min\{x_{1r}, \ldots, x_{nr}\}$, change each x_{ij} to $x_{i(r+1-j)}$. Finally, sort the rows so that $x_{11} < \cdots < x_{n1}$.

77. It appears in level 4, because that placement of vertex 2 creates not only edge 7 (the goal of level 3) but also edge 6. One can think of it as belonging to both levels.

78. (a) At level 5 we've created the $\binom{5}{2}$ edges $\{1, 2, 3, 4, m-6, m-4, m-3, m-2, m-1, m\}$; so the algorithm's next step is to create edge $m-5$. The possibilities are (i) $x_{21} = m-5$; (ii) $x_{51} = 1$; (iii) $x_{41} = m-3$; (iv) $x_{31} = 4$; (v) $x_{11} = 5$.

(b) Moves (i)–(v) of part (a) all work, and they nicely break left-right symmetry. There's also one more possibility, namely (vi) $x_{61} = 3$ and $x_{63} = m-2$; again this breaks reflection symmetry. [All these cases will take us through level 6 to level 7.]

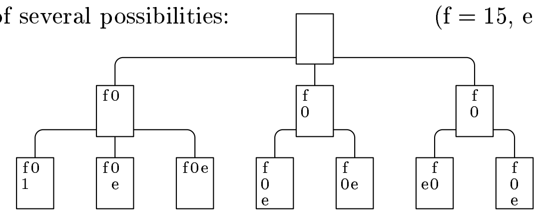
Historical notes: K. E. Petrie and B. M. Smith studied $K_n \square P_2$ for $n \le 5$, in order to test several strategies that exploit symmetry in instances of CSP [*Lecture Notes in Computer Science* **2833** (2003), 930–934]. Their methods were significantly improved by B. M. Smith and J.-F. Puget [*Constraints* **15** (2010), 64–92], who considered KP and KC graphs in general and discovered the unique labeling of $K_6 \square P_3$. However, the method illustrated in Fig. 107 is significantly faster than all of those approaches.

79. Instead of filling the matrix (x_{ij}) with explicit numbers, calculate *symbolically* with values of the form '$m-c$' or 'c' for small values of c. (See exercise 78, and imagine replacing (9, 8, 7, 6, 5) in levels 0 through 5 of Fig. 107 by (m, $m-1$, $m-2$, $m-3$, $m-4$). Notice that nodes on level $l+1$ involve only the values $\{0, 1, \ldots, l\}$ and $\{m, m-1, \ldots, m-l\}$, when $l < \lceil m/2 \rceil$.)

Hence the top $\lceil m/2 \rceil$ levels of this symbolic tree will be the same for all n, except for nodes that have too many rows. It turns out that this tree has only 8910 nodes, and its maximum level is 23. So we can't get an edge labeled $m-23$ when $m > 46$.

[The analogous trees for $K_n \square P_3$ and $K_n \square C_3$ have maximum level 52.]

80. Here is one of several possibilities: (f = 15, e = 14)



The nodes on level 2 have respectively (5, 6, 3, 3, 5, 6, 3) children; and they lead to respectively (60, 49, 29, 23, 47, 63, 13) solutions on level 16. (Left-right symmetry must still be broken below the rightmost node: Use column 1, not column 3, on level 3.)

81. For example, the numbers are 1, 177, 12754, 164273 for $n = 1, 2, 3, 4$; and an instance of $K_6 \square P_4$ is exhibited in Fig. 108. But by extending the method used for $r = 3$, it appears likely that $K_n \square P_4$ will be ungraceful for all sufficiently large n.

82. Applying exercise 102 to this 99-edge graph quickly yields many solutions(!), such as

$$\begin{pmatrix} 31 & 41 & 59 & 26 & 53 & 58 & 97 & 93 & 23 & 84 & 62 & 64 & 33 & 83 & 27 & 95 & 02 \\ 25 & 71 & 19 & 77 & 17 & 86 & 08 & 81 & 65 & 91 & 37 & 99 & 01 & 98 & 04 & 87 & 22 \\ 68 & 24 & 10 & 29 & 74 & 45 & 07 & 18 & 89 & 05 & 96 & 00 & 88 & 03 & 80 & 13 & 94 \end{pmatrix}.$$

[Smaller examples are also of interest. Consider, for example,

$$\begin{pmatrix} 3 & 14 & 15 & 9 & 26 \\ 22 & 21 & 0 & 27 & 13 \\ 7 & 2 & 25 & 1 & 4 \end{pmatrix}, \quad \begin{pmatrix} 3 & 14 & 15 & 9 & 26 & 5 \\ 32 & 0 & 33 & 2 & 11 & 7 \\ 19 & 22 & 10 & 29 & 1 & 31 \end{pmatrix}, \quad \begin{pmatrix} 3 & 14 & 15 & 9 & 26 & 5 & 35 \\ 12 & 0 & 39 & 1 & 30 & 20 & 4 \\ 39 & 36 & 2 & 34 & 7 & 25 & 32 \end{pmatrix},$$

where there are respectively 1, 3, and 16 solutions having those top rows prescribed.]

83. Let $x_{1(2k+1)} = m - x_{2(2k+1)} = 4k$ and $x_{3(2k+1)} = m - 8k - 1$ for $0 \le k < \lceil r/2 \rceil$; $m - x_{1(2k)} = x_{3(2k)} = 4k - 2$ and $x_{2(2k)} = 8k - 3$ for $1 \le k \le \lfloor r/2 \rfloor$; here $m = 6r - 3$ is always odd. (These values are distinct; for example, the even numbers among them are $\{0, 2, \ldots, 2r-2\}$ together with about 1/4 of the larger even numbers $\le 6r-2$.)

The differences between rows 2 and 3 give the odd edges $\{1, 3, \ldots, 2r-1\}$. The other odd edges can be found in the differences between rows 1 and 2 or 3, and between adjacent columns of row 1. Finally, the even edges $\{m - (12k + \{1, 3, 5, 7, 9, 11\})\}$ are all present too. [G. Suresh Singh, *National Academy Science Letters* **15** (1992), 193–194.]

84. Gracefulness is known, via exercise 102, for $1 \le r \le 14$ at least (thanks to computations by the author and Filip Stappers).

85. $K_n \,\square\, C_r$ has $2rn!$ symmetries: We can reflect the corresponding matrix left↔right, and/or shift its columns cyclically, and/or permute its rows arbitrarily.

86. There are $\binom{n+1}{2}r$ edges; and $\binom{n+1}{2} \bmod 4 = (1, 2, 3, 0)$ when $n \bmod 8 = (1, 3, 5, 7)$. Thus $K_n \,\square\, C_r$ is ungraceful when $n \bmod 8 = 1$ and $r \bmod 4 \in \{1, 2\}$; when $n \bmod 8 = 3$ and $r \bmod 4 \in \{1, 3\}$; when $n \bmod 8 = 5$ and $r \bmod 4 \in \{2, 3\}$. (See Fig. 108 for the case $n = r = 5$. There's no restriction when $n \bmod 8 = 7$.)

87. The odd-degree vertices are those in the $r-2$ "middle" cliques, if n is even; otherwise they're the ones in the two "extreme" cliques. This observation can sometimes be used to prune the search tree by ruling out partial solutions whose odd-degree vertices have all been labeled. For example, when proving that $K_6 \,\square\, P_3$ has a unique labeling, it decreases the tree size from 225 meganodes to less than 213 meganodes (about 95%).

88. The method of Fig. 107 shows, in fact, that $K_4 \,\square\, K_4$ has eleven different graceful labelings, one of which is shown here. (It needs only 3 Gμ to discover this, with a search tree of 12 million nodes. It needs 0.76 and 190 Tμ to prove that $K_5 \,\square\, K_4$ and $K_5 \,\square\, K_5$ are *not* graceful.)

$$\begin{pmatrix} 48 & 0 & 39 & 35 \\ 1 & 17 & 47 & 23 \\ 38 & 45 & 7 & 2 \\ 19 & 20 & 5 & 46 \end{pmatrix}$$

89. No; $K_2 \oplus K_2 \oplus K_2 \oplus K_2$ can't be graceful because it has 8 vertices. (But every graph with four edges and ≤ 5 vertices *is* graceful; see the list following Theorem S.)

90. (a) This is the mixed-radix representation $\pi = \left[\begin{smallmatrix} 3; & a_1, & a_2, & a_3, & \ldots \\ 1; & 2, & 3, & 4, & \ldots \end{smallmatrix}\right]$; see 4.1–(9). The recurrence $x_1 = \pi - 3$, $a_n = \lfloor (n+1)x_n \rfloor$, $x_{n+1} = (n+1)x_n - a_n$ yields $(a_1, \ldots, a_{20}) =$ (0, 0, 3, 1, 5, 6, 5, 0, 1, 4, 7, 8, 0, 6, 7, 10, 7, 10, 4, 10) [OEIS A075874].

(b) (0, 0, 0, 1, 0, 1, 1, 2, 2, 1, 2, 1, 1, 2, 1, 3, 2, 4, 3, 5) isolated; (1, 1, 1, 2, 2, 2, 3, 3, 3, 2, 3, 3, 3, 4, 3, 5, 3, 5, 4, 6) components. [These 20 graphs are all planar.]

(c) $\chi(G^\pi_m) = 2$ for $m \in \{1, 2, 3, 9, 10, 12, 15, 17\}$; $\chi(G^\pi_m) = 3$ for the other $m \le 20$.

[From this data we might be tempted to conjecture that a "random graceful labeling," with $m \to \infty$ edges, is a.s. planar, and 3-colorable. But F. Stappers has studied G^π_m for $m \le 10000$, and found them *nonplanar* for $m =$ 33, 38, 41, 44, 46–49, 51–52, 54–56, 58–61, and all cases ≥ 63. On the other hand, they're all 3-colorable.]

91. While generating the 16! instances, as in the proof of Theorem S, we can maintain connectivity information, because the steps of union-find are easily undone (see Algorithm 2.3.3E). We get $\frac{\text{connected}}{\text{total}} = (\frac{864}{864}, \frac{1141312}{1141312}, \frac{159551124}{159601936}, \frac{6537511962}{6562523200}, \frac{106698003000}{108536168696},$

Suresh Singh
author
Stappers
mixed-radix representation
OEIS
planar
Stappers
union-find

$\frac{795992914532}{838037875584}$, $\frac{2869123162654}{3252044834968}$, $\frac{4974721374674}{6508147089024}$, $\frac{3859250594040}{6590461997960}$, $\frac{1104325114202}{3099651627904}$, $\frac{67540932632}{519187026552}$) for $7 \le n \le 17$. (Divide all numerators and denominators by 2 to avoid complement symmetry. Values for graphs with fewer edges are tabulated in OEIS A329790.)

92. This goes faster, because the union-find algorithm can be modified to detect the creation of an odd cycle as soon as it occurs (see Section 7.4.1.1). The new counts are $\frac{8}{8}$, $\frac{22242}{22242}$, $\frac{6317382}{6318302}$, $\frac{427805408}{428781978}$, $\frac{10110694366}{10233657368}$, $\frac{99592576642}{103635506314}$, $\frac{432843270752}{479912612982}$, $\frac{796114433250}{1009922060716}$, $\frac{516439259812}{876211145722}$, $\frac{67540932632}{234013536424}$, for $8 \le n \le 17$; 2714363642056 altogether ($\approx 0.1297 \cdot 16!$).

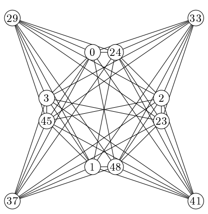
Incidentally, there are 11932174 graphs with 16 edges and at most 17 vertices, of which 915503 (about 7.67%) are bipartite.

[When the labelings are also restricted to be α-graceful, in the sense of exercise 111, the results become $\frac{6}{6}$, $\frac{6840}{6840}$, $\frac{1855942}{1856280}$, $\frac{124467512}{124746754}$, $\frac{2945525928}{2980811422}$, $\frac{29277794448}{30452911120}$, $\frac{128904318498}{142798046522}$, $\frac{240333763962}{304499321272}$, $\frac{157722174046}{267381496426}$, $\frac{20772768256}{72154842584}$; 820394039226 altogether ($\approx .0392 \cdot 16!$).]

93. Such a graph must have $n = 2m/r$ vertices; so $2m/r$ must be an integer $> r$. We can proceed as in Theorem S and exercise 91, but prune the search by disallowing partial solutions with more than n nonisolated vertices, or with any vertex of degree $> r$.

Examples for small r are easy, and unique: K_3 when $r = 2$, K_4 when $r = 3$, and the octahedron $K_{2,2,2}$ when $r = 4$. There are six labelings when $(m, r) = (20, 5)$: Two of them give $\overline{C_8}$, the other four give $\overline{C_3 \oplus C_5}$. Similarly, $(m, r) = (27, 6)$ yields two graceful labelings of $\overline{C_9}$. A *unique* labeling appears for $(m, r) = (35, 7)$; its graph is $\overline{C_3 \oplus C_7}$.

When $r = 8$ we must go up to $m = 48$. Here there are 14 graceful labelings, for eight different graphs. The most symmetrical solution, shown here, has a graph with 384 automorphisms.



(All of these computations are short; but other methods are needed for $r > 8$. See E. Pegg Jr., `math.stackexchange.com/questions/3246000` (2019), and OEIS A308722. Pegg conjectures that the smallest instances for $r = 2k > 2$ occur when $m = 3k^2$.)

94. A 2-regular graph with m edges is a disjoint union of cycles, having a total of m vertices. The number of graceful labelings for $m = 3, 4, \ldots, 16$, with $0 \mathrel{—} (m{-}1)$, is 1, 1, 0, 0, 7, 18, 0, 0, 175, 414, 0, 0, 7602, 20846. (Corollary E explains the zeros.)

It's easy to find the cyclic components of any given labeling; so we can identify isomorphic graphs among those labelings. There are $[z^m]\, 1/\prod_{n\ge3}(1 - z^n)$ different 2-regular graphs with m edges; hence the potential numbers of graceful 2-regular graphs, for those values of m, are respectively 1, 1, 0, 0, 2, 3, 0, 0, 6, 9, 0, 0, 17, 21. The actual numbers turn out to be 1, 1, 0, 0, 2, 3, 0, 0, 5, 8, 0, 0, 14, 19. Missing are $2C_3 \oplus C_5$ (that is, $C_3 \oplus C_3 \oplus C_5$); $4C_3$; $5C_3$, $3C_3 \oplus C_6$, $3C_5$; $3C_3 \oplus C_7$, $2C_3 \oplus 2C_5$.

[In *Utilitas Mathematica* **7** (1975), 263–279, A. Kotzig proved that tC_5 is ungraceful for all $t \ge 1$. And in *Congressus Numerantium* **44** (1984), 197–219, he showed that a graceful 2-regular graph with t odd components must have at least $t(t+2)$ vertices. These results account for all of the missing cases listed above, except for $3C_3 \oplus C_6$. On the other hand he showed that $C_3 \oplus C_5 \oplus \cdots \oplus C_{2t+1}$ *is* graceful, for all $t \ge 1$. And with J. Abrham, he also proved that $C_p \oplus C_q$ is graceful if and only if $(p+q) \bmod 4 \in \{0, 3\}$; see *Discrete Mathematics* **150** (1996), 3–15.]

Incidentally, a gracefully labeled 2-regular graph always leaves one label $\in [0\,..\,m]$ unused. The unused label was respectively (4, 5, ..., 12) in the case $m = 16$ exactly (311, 1547, 3208, 3510, 3651, 3532, 3241, 1554, 292) times.

complement symmetry
OEIS
α-graceful
unique
octahedron
C_n: cycle graph
symmetrical
Pegg Jr.
OEIS
cycles
Kotzig
Abrham

95. Now there are $m = 3t$ edges and $n = 2t$ nonisolated vertices, for $2 \le t \le 7$. The method of exercise 93 rapidly gives us graceful labelings galore, respectively (1, 5, 222, 22806, 2988280, 641731574) of them.

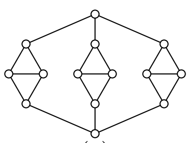
The main difficulty is to group them efficiently into equivalence classes of isomorphic graphs. One good way is to compute a "hash code" $h(G)$ for each graph G. Let r_1, r_2, … be pseudorandom integers in the range $0 \le r_j < 2^{30}$, and let $r_0 = 0$. For each vertex v, compute $h(v)$ as follows: Let $V_k(v)$ be the set of vertices at distance k from v, and let $d(v)$ be the maximum k with $V_k(v) \ne \emptyset$. Set $t_w \leftarrow 2r_k + 1$ for each $w \in V_k(v)$. Then, for $k = d(v)$, $d(v)-1$, …, 0, compute $t'_w = t_w \prod_{u \text{---} w}(2r_{2d(v)+1-k} - t_u)$ for all $w \in V_k(v)$, and set $t_w \leftarrow t'_w$ for all such w. Let $h(v)$ be the product of all those values t'_w, mod 2^{32}. (Notice that $h(v)$ is always odd, and $h(v) = 1$ when v is isolated.)

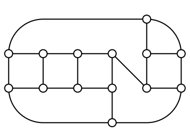
The hash code $h(G) = \left(\sum_v \lfloor h(v)/2 \rfloor\right) \bmod 2^{32}$, summed over all vertices v, now has the property that $h(G) = h(H)$ whenever graphs G and H are isomorphic. Furthermore, with trial and error we can find constants r_k for which $h(G) \ne h(H)$ whenever G and H are nonisomorphic cubic graphs with at most 14 nonisolated vertices.

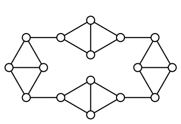
(The adjacency matrices for all connected cubic graphs with up to 24 vertices can be downloaded in a compact format from `houseofgraphs.org`, the "House of Graphs"; and the disconnected ones can be readily constructed from the connected ones. (See G. Brinkmann, K. Coolsaet, J. Goedgebeur, and H. Mélot, *Discrete Applied Math.* **161** (2013), 311–314.) For example, there are 509 connected cubic graphs with 14 vertices, and 540 altogether. In fact, the author's first try to choose random constants r_j actually was able to characterize uniquely every cubic graph with fewer than 20 vertices.)

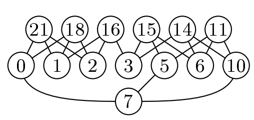
The bottom line is that *every cubic graph with at most* 14 *vertices is graceful, with only two exceptions:* $2K_4$ when $n = 8$ and $3K_4$ when $n = 12$. [A. Kotzig and J. Turgeon proved that the graph tK_n is graceful if and only if $t = 1$ and $n \le 4$; see *Colloquia Mathematica Societatis János Bolyai* **18** (1976), 697–703.] In fact, none of the *connected* cubic graphs are the least bit difficult to label; the two "least graceful" such graphs when $n = 14$ are graph (∗) below, with 9526 labelings and 96 automorphisms, and the Heawood graph 7–(57), with 10436 labelings and 336 automorphisms. (The disconnected graph $2K_4 \oplus (K_3 \square P_2)$, with 13824 automorphisms, has only 11 graceful labelings.) The "most graceful" of the 14-vertex cubics has 3762313 labelings(!) and only the identity automorphism; it's (∗∗) below.

Suppose we prespecify the labels $0 = l_0 < l_1 < \cdots < l_{n-1} = m$ that are to be used. Then a cubic graceful labeling is the solution to the MCC problem whose primary items are #1, …, #m and l_0, …, l_{n-1}, where the l's have multiplicity 3; the options are simply '#k l_i l_j' for $0 \le i < j < n$, where $k = l_j - l_i$. We can assume that $l_{n-2} = m-1$, and disallow '#$(m-1)$ 1 m'. It turns out that only 27028 of the $\binom{19}{11} = 75582$ choices for the l's have solutions. The one for labels $\{0, 1, 2, 3, 5, 6, 7, 10, 11, 14, 15, 16, 18, 21\}$ is unique (see (∗∗∗∗) below); but $\{0, 1, 2, 3, 5, 6, 10, 11, 16, 17, 18, 19, 20, 21\}$ has 455698 solutions.


(∗)


(∗∗)


(∗∗∗)

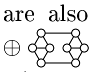


(∗∗∗∗)

96. With considerably more computation, the results of exercise 95 can be extended to the 204,154,267,353 graceful labelings of cubic graphs on 16 vertices. There are 4207 such graphs, of which 4060 are connected. The evidence is overwhelming: Each of

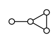
hash code
isomorphism clustering
adjacency matrices
House of Graphs
internet
Brinkmann
Coolsaet
Goedgebeur
Mélot
author
Kotzig
Turgeon
Heawood graph
MCC problem

the connected ones has at least 107,291 essentially different graceful labelings. (That "least graceful" example is (∗∗∗) above.) From this circumstantial evidence, the author conjectures confidently that every connected cubic graph is graceful.

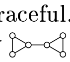
Furthermore, all 147 of the *disconnected* cubic graphs on 16 vertices are also graceful, except of course for $4K_4$. The closest to being ungraceful are $2K_4 \oplus$  (with 213 labelings) and $2K_4 \oplus P_2 \square P_2 \square P_2$ (with 1149). With only a bit of trepidation we may therefore conjecture that *every* cubic graph is graceful, except for $2K_4$, $3K_4$, ...

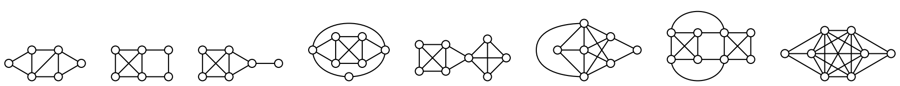
97. Backtracking via Theorem S, as in exercise 91, we can avoid most of the $m!/2$ cases by allowing at most 8 of the vertices $\{0, 1, \ldots, m\}$ to touch an edge. Thus we readily discover that the (1, 2, 7, 23, 122, 888, 11302) distinct graphs with $n =$ (2, 3, ..., 8) nonisolated vertices have respectively (1, 2, 13, 157, 3292, 110578, 5903888) different graceful labelings. (Complementary labelings are not considered different.)

All graphs with at most 8 nonisolated vertices can be found in the House of Graphs. And the hash function in answer 95, but with different r_j, works for them.

One of the seven graphs with $n = 4$ nonisolated vertices, $2K_2$, doesn't have enough edges to be graceful. But the text points out that the other six work out fine (indeed, uniquely for $K_{1,3}$, P_4, C_4, and K_4, and up to 5 ways with the paw ).

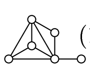
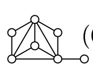
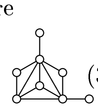
When $n = 5$, $K_2 \oplus P_3$ has too few edges; $K_2 \oplus K_3$ can't be labeled either; and Corollary E rules out C_5 and K_5, as well as $K_1 -\!\!- 2K_2$. The other 18 are graceful: $K_{1,4}$ uniquely, and the "dart" $K_1 -\!\!- (K_1 \oplus P_3)$ maximally (26 ways).

Cases $n =$ (6, 7, 8) lose respectively (4, 7, 19) graphs with too few edges, and (4, 20, 93) graphs that violate Corollary E. But they do include (109, 845, 11124) graceful graphs. Of course $K_{1,n-1}$ is always uniquely graceful. The other unique cases for $n = 6$ are $K_2 \oplus K_4$, $K_{3,3}$, $K_{2,2,2}$, and the double paw . The other unique cases for $n = 7$ are mostly disconnected: $P_2 \oplus L_{3,2}$, $P_3 \oplus C_4$, $C_3 \oplus P_4$, $C_3 \oplus C_4$, $C_3 \oplus L_{3,1}$, $P_3 \oplus K_4$, $K_3 \oplus K_{1,1,2}$, $K_2 \oplus K_5$; the connected one is $K_1 -\!\!- (2K_1 -\!\!- 2K_2)$. (Here $L_{m,n}$ denotes the "lollipop graph" on $m + n$ vertices, consisting of K_m and P_n joined by a bridge; $L_{3,1}$ is the paw.) There are 10 disconnected uniquely graceful graphs for $n = 8$: $K_2 \oplus C_6$, $2K_2 \oplus K_{1,1,2}$, $P_3 \oplus C_5$, $C_3 \oplus P_5$, $K_{1,3} \oplus L_{3,1}$, $2K_2 \oplus K_4$, $K_3 \oplus L_{4,1}$, $K_{1,3} \oplus K_4$, $P_3 \oplus K_5$, $K_3 \oplus (P_2 -\!\!- P_3)$. And the 19 connected ones likewise have lots of symmetry: $K_{1,7}$, G_{14}, $2K_1 -\!\!- 3K_2$, G_{16}, $4K_1 -\!\!- 2K_2$, $2K_1 -\!\!- (K_2 \oplus K_4)$, $K_2 -\!\!- 2K_3$, $K_1 -\!\!- G_{13}$, $K_1 -\!\!- (2K_1 -\!\!- (K_2 \oplus K_3))$, $2K_1 -\!\!- K_{3,3}$, $K_3 -\!\!- (K_1 \oplus 2K_2)$, $K_3 -\!\!- (P_2 \oplus P_3)$, $K_3 -\!\!- (2K_1 \oplus K_3)$, G_{21}, $K_1 -\!\!- G'_{14}$, $2K_1 -\!\!- G_9$, $K_2 -\!\!- G'_9$, $K_2 -\!\!- G''_9$, $K_3 -\!\!- (K_1 \oplus C_4)$, where G_m or G'_m or G''_m denotes a special graph with m edges:



G_9 G'_9 G''_9 G_{13} G_{14} G'_{14} G_{16} G_{21}

The champions for gracefulness with 6, 7, and 8 vertices are

 (126 labelings),  (680 labelings), and  (3778 labelings).

98. (a) No, because edge 11 ($3 -\!\!- 14$, `NC` — `SC`) doesn't touch edges 12–18 (see (33)).

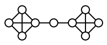
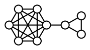
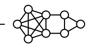
(b) 11067 (including the solution to Fig. 105(d)).

(c) A rooted labeling always defines a connected graph. We get n nonisolated vertices in respectively (864, 1122012, 148696974, 5469393230, 75003795230, 436515974020,

author
House of Graphs
paw
bowtie graph ($K_1 -\!\!- 2K_2$)
dart
K_n
$K_{m,n}$
P_n
C_n
$L_{m,n}$
lollipop graph
bridge
paw

1132397252122, 1296227076156, 605872421102, 94984144008, 2895168460) cases, for $7 \le n \le 17$. The total, 3649515044178, is approximately 17.4% of 16!.

(d) 1, 1, 1, 1, 2, 3, 1, 3, 3, 4, 5, 7, 3, 3, 15, 4. (See OEIS A338988 for further values. No pattern is evident. Does this sequence grow exponentially?)

99. (See exercise 97.) The *only* example with at most 8 vertices is $4K_1 — 2K_2$. (And the only examples with 9 vertices are , , K_1 — , and $K_1 — (2K_1 — (K_2 \oplus (2K_1 — K_2)))$; these are just four of the 259614 connected graceful graphs. The first of these is the only example with at most 14 edges.)

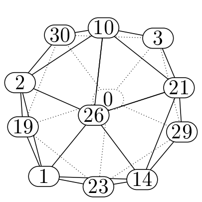
100. After numerous experiments, the author's most successful attempt is a backtrack program called BACK-GRACEFUL, based on Algorithm 7.2.2W (and available online). It uses a sparse-set structure to keep a list of all vertices, with labeled vertices at the left. To enable efficient bitwise tests, it maintains $ebits = \sum_k 2^k[\text{edge } k \text{ is labeled}]$; $rebits = \sum_k 2^{m-k}[\text{edge } k \text{ is labeled}]$; and $vbits = \sum_k 2^k[\text{no vertex is labeled } k]$. (For example, if v is an unlabeled vertex with a neighbor labeled k, we can AND the vector of permissible labels for v with $\neg((ebits \ll k) + (rebits \gg (m-k)))$.)

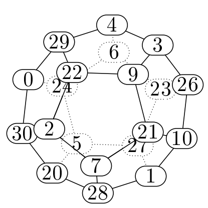
The current state is also maintained in four arrays lt, lu, vt, vu: If vertex v is unlabeled, $vt[v] = -1$ and $vu[v]$ is the number of v's unlabeled neighbors. But if v is labeled k, we have $vt[v] = k$, $lt[k] = v$; $lu[k]$ is the number of v's unlabeled neighbors, and $vu[v]$ is the value of $vu[v]$ when the label was assigned. If no vertex has been labeled k, $lt[k] = -1$ and $lu[k]$ is undefined.

The task at each level is to label a vertex for the longest currently unlabeled edge, unless some unlabeled vertex has only one viable label.

To enumerate all ways that might create an edge of length q, we run through all pairs $(j,k) = (0,q)$, $(1,q{+}1)$, …, $(m{-}q, m)$ such that either (i) $lu[j] > 0 > lu[k]$; or (ii) $lu[k] > 0 > lu[j]$; or (iii) $lu[j] < 0$ and $lu[k] < 0$. In case (i), we set $v \leftarrow lv[j]$, and for all $v — w$ with $lt[w] < 0$ we can label w with k. Case (ii) is similar. Case (iii) is the more difficult "unrooted" case [see exercise 98]: For all unlabeled v with $vu[v] > 0$, we prepare to label v with j now, and to label one of v's neighbors with k at the next level, if that succeeds. An attempted vertex labeling fails if it duplicates a previous edge label.

101. Each polyhedron has 30 edges and 120 automorphisms. Both questions were apparently answered correctly for the first time in October 2020, by B. Dobbelaere and T. Rokicki (working independently!). The icosahedron has only 12 vertices, and we easily find 24 distinct solutions, of which 5 include the triangle $0 — 30 — 29 — 0$ and 19 have the induced path $30 — 0 — 29$. All are rooted except for the solution shown.

(a) 

; (b) 

.

The dodecahedron, with 20 vertices to label, is much more challenging; it has 784,298,856 distinct labelings, of which 38,092,064 are rooted (4.9%). The algorithm of exercise 100 finds them in 25.3 teramems, with a 203-giganode search tree.

Notes: The solution shown is one of just 1882 for which all vertex labels lie in $[0\,..\,10] \cup [20\,..\,30]$. Since one of the edges has length 10, we cannot eliminate both 10 and 20. It turns out that *both* 10 and 20 must be used, and that exactly 9 of the labels

OEIS
author
backtrack program
downloadable programs
sparse-set
bitwise tests
AND
rooted
automorphisms
Dobbelaere
Rokicki
rooted

must be odd. Incidentally, to find those 1882, the XCC model of exercise 69 actually runs significantly faster than the supposedly "streamlined" algorithm of exercise 100.

102. See the online program BACK-GRACEFUL-ROOTED-RANDOMRESTARTS, developed by T. Rokicki and the author. As in exercise 100, it's based on Algorithm 7.2.2W. But for speed it considers only labelings that are "rooted" with respect to previously specified labels, and it uses simpler data structures to detect duplicate edges. It randomizes the table of legal moves at every level, and uses reluctant doubling (Eq. 7.2.2.2–(131)) to restart periodically in a new, randomly generated part of the search space.

103. There are four with 0 at the Y: $\begin{smallmatrix}2&16\\8&15\end{smallmatrix}\,0\,1$; $\begin{smallmatrix}2&16\\14&1\end{smallmatrix}\,0\,15$, $\begin{smallmatrix}8&16\\11&7\end{smallmatrix}\,0\,15$; $\begin{smallmatrix}8&15\\9&1\end{smallmatrix}\,0\,16$. There's one with 15 at the Y: $\begin{smallmatrix}6&9\\12&3\end{smallmatrix}\,15\,0$. There are nine with 16 at the Y, such as $\begin{smallmatrix}3&6\\5&10\end{smallmatrix}\,16\,0$. And 33 with other elements at the Y, such as $\begin{smallmatrix}4&1\\2&3\end{smallmatrix}\,5\,0$ and $\begin{smallmatrix}8&4\\3&6\end{smallmatrix}\,12\,0$. Total 47.

104. (a) There are $k+1$ components and k residues.

(b) If r is bad and $x \bmod k = r$, then we clearly can't set `LO[`k`]` $\leftarrow x$. But if r is good, at least one such x is OK.

(c) Say that x is a big vertex if $x+k>m$. There are g big good vertices, lying in $\le g$ components. The largest good vertices in the other good components are OK.

(d) The vertices $\{r, r+k, \ldots, r+pk\}$ can't be connected by p edges of lengths $>k$.

(e) The $k+1-G$ bad components account for at least $2(k+1-G)$ bad residues, by (d). Hence $g \le k-2(k+1-G)$ and we have $G-g \ge k+2-G$. If $G \le \frac{2}{3}(k+2)$ we have $G-g \ge (k+2)/3$; otherwise either g or $G-g$ is $\ge G/2 > (k+2)/3$. Thus $\lceil (k+2)/3 \rceil = \lfloor (k+4)/3 \rfloor$ is a valid lower bound in all cases, by (b) and (c). [Experiments for $m \le 20$ suggest that $t_k = \lfloor (k+3)/2 \rfloor - [k \text{ odd and } k = \lceil m/2 \rceil - 2 > 1]$ may in fact be valid.]

(f) When $k \ge m/2$, all edges connect small to big. The hint follows because the cycle containing x and $x+k$ includes the edges $y \mathbin{—} (x+k) \mathbin{—} x \mathbin{—} z$.

Let there be c unusable vertices, in C components. A component that contains $q>0$ unusable vertices $x_1 < \cdots < x_q$ therefore contains at least the $2q+2$ vertices $y_1 < x_1 < \cdots < x_q < x_1+k < \cdots < x_q+k < z_q$, and it contains at least $2q+1$ of the $m-k$ edges. Consequently $m-k \ge 2c+C$; and the number of usable vertices is $m+1-k-c \ge (m-k)/2+1+C/2 \ge 2+\lfloor (m-k)/2 \rfloor$, unless $C=c=0$.

[Altogether we get the superexponential lower bound $t_1 \ldots t_m = \Omega(m!/24^{m/2})$.]

105. (a) For example, when $n=4$ it's $\det\begin{pmatrix} x_1+x_1+x_2 & -x_1 & -x_2 \\ -x_1 & x_2+x_1+x_1 & -x_1 \\ -x_2 & -x_1 & x_3+x_2+x_1 \end{pmatrix}$.

(b) The sum of $s_2 \ldots s_{n-1} S(1, s_2, \ldots, s_{n-1})$ over all 2^{n-2} choices of $s_j = \pm 1$ is 2^{n-2} times the desired result. For example, when $n=4$ we have $[x_1x_2x_3]\,S(x_1,x_2,x_3) = (S(1,1,1) - S(1,1,-1) - S(1,-1,1) + S(1,-1,-1))/4$. [See OEIS A033472.]

106. Empirical investigations by D. Anick suggest that $\tau(n)/\tau(n-1)$ grows approximately as $a + bn + (-1)^n c/n$ for some constants a, b, c. If that is true, $\tau(n) = \exp\bigl(n \ln n - n \ln(e/b) + O(\log n)\bigr)$. The exact values for $n<30$ suggest further that $a \approx 0.19$, $b \approx 0.636$, and $c \approx 0.42$. But rigorous proofs are unknown. (This function $\tau(n)$ was introduced by A. Kotzig, who computed it by hand for $n \le 6$ in 1984.)

107. Suppose $1 \le e < 2^n$, where $2^n + e = (e_n \ldots e_1 e_0)_2$. Then the edge labeled e is between $x = (x_{n-1} \ldots x_1 x_0)_2$ and $x \mathbin{\&} (x-1)$, if $e_k = 1$ and $e_{k-1} = \cdots = e_0 = 0$ and $x_j = e_j \oplus [j>k] e_{j+1}$ for $0 \le j < n$. (This is in fact an α-labeling. Notice that $l(x)$ is essentially a left-right reflection of inverse Gray code, 7.2.1.1–(8).)

108. Notice that T_n, like P_n, has two automorphisms; so we divide the total number of graceful labelings by 4. This yields 30 and 988184 for T_3 and T_4; also approximately $4 \cdot 10^{18}$ and 10^{48} for T_5 and T_6, using ten million estimates with Algorithm 7.2.2E.

XCC model
online
downloadable programs
Rokicki
author
reluctant doubling
restart
OEIS
Anick
Kotzig
α-labeling
Gray code
automorphisms

109. (a) Suppose α has even parity and β has odd parity. Then $l(1\beta) - l(1\alpha) = l(0\beta) - l(0\alpha) - 2^{n-2} - 2a_{2^{n-2}}$, because $a_0 = 0$. Hence $L_1 = L_0 - 2^{n-2} - 2a_{2^{n-2}}$.

(b) Let $a_{2^k} = (k+2)2^{k-1}$. This choice makes $(a_0, a_1, \ldots) = (0, 1, 3, 4, 8, 9, \ldots)$, and we have $a_n = \sum_{k=1}^{n} 2^{\rho k}$ for all n. (It can be shown that $a_n = n + (e_1 2^{e_1} + \cdots + e_t 2^{e_t})/2$ when $n = 2^{e_1} + \cdots + 2^{e_t}$ with $e_1 > \cdots > e_t \geq 0$.) By part (a), $L_0 = L_1 + 2^{n-2} + 2a_{2^{n-2}} = L_1 + (n+1)2^{n-2}$. The other edges 0α — 1α have labels

$$\{m - k - a_k - a_{2^{n-1}-1-k} \mid 0 \leq k < 2^{n-1}\} \;=\; \{m - k - (n-1)2^{n-2} \mid 0 \leq k < 2^{n-1}\},$$

because $a_k + a_{2^{n-1}-1-k} = a_{2^{n-1}-1} = (n-1)2^{n-2}$. Thus $L_1 = \{1, 2, \ldots, (n-1)2^{n-2}\}$ by induction; and it all works, α-gracefully. [M. Maheo, *Discrete Mathematics* **29** (1980), 39–46; A. Kotzig, *Journal of Combinatorial Theory* **B31** (1981), 292–296.]

110. (a) $n = \sum_{k=0}^{r}(t_k - s_k + 1)$ vertices; $n - r - 1$ vertical plus $n - t_r - 1$ horizontal edges.

(b) Numbers in ovals don't change; in rectangles they're subtracted from 28.

(c) Use a rectangle for (x, y) when $x + y$ is odd. Label $(0, 0)$ with 0. For each edge, proceeding left to right and bottom to top, make the labels of its endpoints sum respectively to 0, 1, 2, ... (This will make the label in a rectangle equal to the one below it, and one less than the one above it, when those neighbors exist.)

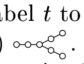
(d) Yes! In general let $\Sigma_0 = t_0$, $\delta_0 = 0$, and $\Sigma_{k+1} = \Sigma_k + t_k + t_{k+1} - 2s_{k+1} + 1$, $\delta_{k+1} = \Sigma_k - \delta_k - s_{k+1}$, for $0 \leq k < r$. Then the label of (x, y) corresponding to (i) is $\delta_x + \lfloor y/2 \rfloor$ when x is even, $\delta_x + \lceil y/2 \rceil$ when x is odd.

[This in fact is an instance of α-labeling as in exercise 111, where the u's are ovals and the v's are rectangles. A. Rosa presented a special case in Lemma 4.3 of his thesis.]

111. (a) We have $\overline{v_k} = m - v_k \geq m - (m - l) = l > u_j$. Hence all the complemented labels exceed all the uncomplemented ones, and $|u_k - \overline{v}_k| = m - v_k - u_k = m - k$ for all k.

(b) Since C_n has n edges, Corollary E tells us that $n \bmod 4$ must be 0 or 3. But a bipartite graph has no odd cycles; hence $n = 4k$. Conversely, the labels 0 — $\overline{1}$ — 1 — $\overline{2}$ — 2 — $\overline{3}$ — 4 — $\overline{4}$ — 5 — $\overline{5}$ — 6 — $\overline{0}$ — 0 for C_{12} reveal a general pattern that works for all $k > 0$. (The similar non-α pattern 0 — 11 — 1 — 10 — 2 — 9 — 4 — 8 — 5 — 7 — 0 for C_{11} shows that C_{4k+3} is at least graceful, for all $k \geq 0$.)

(c) Let $u_k^* = u_{m-1} - u_{m-1-k}$ and $v_k^* = v_{m-1} - v_{m-1-k}$. (Equivalently, change each vertex label t to $(l - 1 - t) \bmod (m + 1)$. Notice that $u_{m-1} = l - 1$ and $v_{m-1} = m - l$.)

(d) . [A. Rosa introduced α-graceful graphs, and solved these problems as well as exercise 115(b), in his original paper that introduced graceful graphs. His thesis (1965) credited A. Kotzig for part (b).]

112. (a) This is obvious, because $u_k < l \leq \overline{v_k}$.

(b) We can assume that the central vertex is labeled 0. Then there are two solutions, both with leaves labeled $\{1, 2, 4\}$.

(c) Suppose the edges of G are u_1 — v_1, ..., u_m — v_m, and the vertices of $K_{m,m}$ are $\{a_0, \ldots, a_{m-1}, b_0, \ldots, b_{m-1}\}$. We use a "cyclic" analog of the rainbow copies of K_{11} in Fig. 110(c): Let the edges of the kth copy be $\bigl(l(u_j)+k\bigr) \bmod m$ — $\bigl(l(v_j)+k\bigr) \bmod m$, for $1 \leq j, k \leq m$. (For example, the three copies of the path 0 — 3 — 1 — 2 in $K_{3,3}$ are a_1 — b_1 — a_2 — b_0, a_2 — b_2 — a_0 — b_1, a_0 — b_0 — a_1 — b_2.)

(d) Simply let the ith copy v_i of vertex v have label $l(v) + (i-1)m$. (Consequently, by (c), we can perfectly pack $t^2 m$ copies of G into $K_{tm,tm}$; also $(2tm+1)t$ into K_{2tm+1}.)

Historical notes: I. Cahit proposed the concept of ordered labeling in an unpublished research report at the University of Waterloo [CORR 80-47 (December 1980), 6 pages]. It was introduced independently by S. I. El-Zanati, M. J. Kenig, and C. Vanden Eynden, in *Australasian J. Combinatorics* **21** (2000), 275–285, who

parity
ruler function ρ
α-gracefully
Maheo
Kotzig
Rosa
cycle, graceful and α-graceful
Rosa
historical notes
Kotzig
Historical notes
Cahit
El-Zanati
Kenig
Vanden Eynden

also showed that the graph $S_{n,2}$ with $2n$ edges $0 \longrightarrow a_j \longrightarrow b_j$ for $1 \le j \le n$ has an ordered graceful labeling; that graph isn't α-graceful when $n > 2$.

113. (a) $\sum_{l=1}^{m} \prod_{k=0}^{m-1} (\min(k+1,l) - \max(0,k+l-m))$, since the choices for each k are independent and since $u_{m-1} = l-1$. (See OEIS A005193. Sheppard proved this when he introduced Theorem S). The values for $2 \le m \le 8$ are 2, 4, 10, 30, 106, 426, 1930.

(b) No simple formula is evident. The values are now 2, 4, 12, 40, 182, 906, 5404. When $m = 16$ there are 246,377,199,752, compared to 7,614,236,170 for (a).

Divide by 2 if complementary labelings are considered to be equivalent.

114. (a) If the elements of $K_{a,b}$ and $K_{c,d}$ are respectively u_i, v_j and x_k, y_l, the elements of $K_{a,b} \otimes K_{c,d}$ are $u_i x_k$, $u_i y_l$, $v_j x_k$, $v_j y_l$, for $1 \le i \le a$, $1 \le j \le b$, $1 \le k \le c$, $1 \le l \le d$. The edges are $u_i x_k \longrightarrow v_j y_l$, $u_i y_l \longrightarrow v_j x_k$, so the product is $K_{ac,bd} \oplus K_{ad,bc}$.

(b) (i) Think of the black or white squares of the $2m \times 2n$ chessboard, connected by bishop moves. Rotate by $90°$ to get either an $(m+n) \times (m+n-1)$ or $(m+n-1) \times (m+n)$ board, connected by rook moves, but with right triangles removed from the corners. These right triangles affect $m-1$ rows/columns at the upper left and lower right; they affect $n-1$ rows/columns at the lower left and upper right.

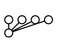
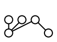
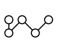
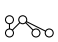
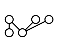
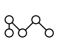
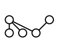
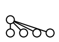
(ii) Now the board is $(m+n) \times (m+n)$, with n (not $n-1$) rows/columns affected at the upper left or lower right. Again the two graphs are isomorphic by transposition.

(iii) One of the graphs has $\lfloor (2m+1)(2n+1)/2 \rfloor$ vertices; it's an $(m+n) \times (m+n)$ board with $m-1$ and $n-1$ rows/columns affected at corners. The other, with one more vertex, is an $(m+n+1) \times (m+n+1)$ board with m and n rows/columns affected. [When $m = n$ these are the Aztec diamonds of orders n and $n + 1/2$.]

(iv) Both are generalized toruses (exercise 7–137), with offsets $(m, -m)$ and (n, n).

(v) The graph whose vertices are binary vectors $x_1 \ldots x_m y_1 \ldots y_n$ of given parity. Each vertex has mn neighbors: Complement one of the x's and one of the y's.

(c) Complementing labels interchanges parts; so we need only consider $(G \otimes H)'$. Let G's parts (U, V) have labels $l(u)$, $l(v)$, and let H's parts (X, Y) have labels $l(x)$, $l(y)$. The new labels $l(ux) = l(u) + ml(x)$, $l(vy) = l(v) + ml(y) - m$ work beautifully, where m is the number of edges in G. [H. S. Snevily, *Discrete Math.* **170** (1997), 185–194.]

115. (a) 000  001  010  011  100  101  110  111 

(b) Use labels 0, 1, ..., s for u_0 through u_s and $\overline{0}$, $\overline{1}$, ..., $\overline{t}$ for v_0 through v_t.

(c) There are m_{s+t+1} edges, where $m_0 = 0$ and $m_{i+1} = m_i + p_{s_i} + q_{t_i} + 1$.

(d) Assign the label $a_j + i$ to each element u_{ji} of U_j, and the label $\overline{b_k + i}$ to each element v_{ki} of V_k, where $a_0 = b_0 = 0$ and $a_{j+1} = a_j + p_j + q_{j'} + 1$, $b_{k+1} = b_k + p_{k''} + q_k + 1$; here $j' = \max\{i \mid u_j \longrightarrow v_i\}$ and $k'' = \max\{i \mid u_i \longrightarrow v_k\}$. The labels are distinct because $a_{j+1} > a_j + p_j$, $b_{k+1} > b_k + q_k$. These definitions ensure that $a_{s_i} + b_{t_i} = m_i$; hence the edges of the caterpillar between U_{s_i} and V_{t_i} receive the labels $m - m_i$, $m - m_i - 1$, ..., $m - m_{i+1} + 1$. When $i = s + t$ we have $s_i = s$, $t_i = t$; the final edge label is 1. In the example, $(a_0, a_1, a_2) = (0, 6, 11)$ and $(b_0, b_1, b_2) = (0, 4, 10)$; see below.

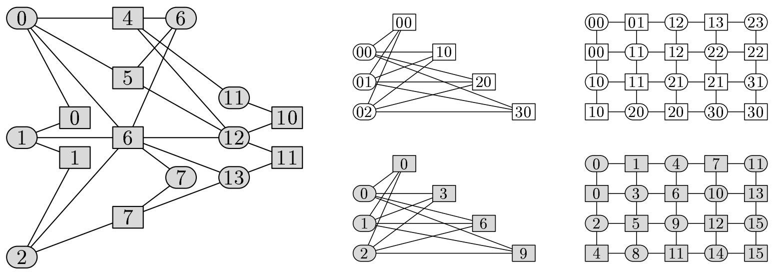
(e) Let $(s, t) = (0, r-1)$; this gives the caterpillar $K_{1,r}$, whose edges are $u_0 \longrightarrow v_0$, ..., $u_0 \longrightarrow v_{r-1}$. Then set $p_0 = n-1$ and $q_i = 0$ for $0 \le i < r$. (See the case $(3, 4)$ below.)

(f) Denote the grid points by (x, y) for $0 \le x < r$ and $0 \le y < n$. Let U_j be the points with $x + y = 2j$, and let V_k be the points with $x + y = 2k + 1$, as illustrated below for $n = 5$ and $r = 4$. The edges between $U_0 \longrightarrow V_0 \longrightarrow U_1 \longrightarrow V_1 \longrightarrow \cdots$ are staircase paths. (Hence this is a caterpillar net in which every caterpillar is simply a path. See B. D. Acharya and M. K. Gill, *Indian J. Math.* **23** (1981), 81–94.)

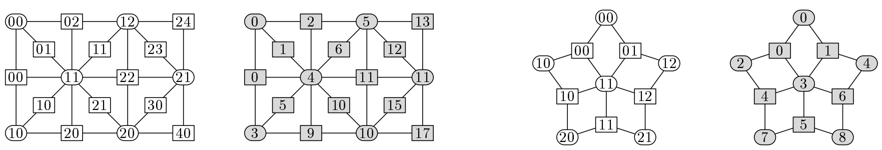
OEIS
Sheppard
chessboard
Aztec diamonds
generalized toruses
parity
Snevily
Acharya
Gill

Acharya
roots of unity
factorization
intervals

In the following illustrations, digits ji in an oval signify u_{ji}; digits ki in a rectangle signify v_{ki}; shaded nodes show final vertex labels; shaded rectangles are complemented:

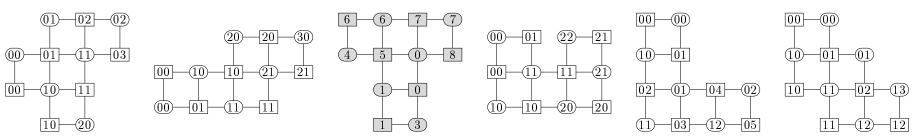


(g) Yes, they both are:



116. Exercise 110 applies to P, Q, V, W, and Z; but exercise 115 is stronger.

In fact, the skeletons of all but the T pentomino are caterpillar nets; the T does, however, have 1824 different α-graceful labelings. It's easy to decompose the others into small caterpillars, as in the decomposition of S below, thereby writing down a labeling quickly by hand — except that the (unique) decomposition of U is difficult to find. The R, V, and W also have surprising decompositions into rather large caterpillars:



[See B. D. Acharya, *Lecture Notes in Math.* **1073** (1984), 205–211.]

117. (a) $(\sum_{i=1}^{n} x^{a_i})(\sum_{j=1}^{r} x^{b_j}) = \sum_{k=0}^{m-1} x^k$ is an algebraic way to say that $\{a_1, \ldots, a_n\}$ and $\{b_1, \ldots, b_r\}$ are nonnegative integers whose nr sums $a_i + b_j$ yield $\{0, \ldots, m-1\}$.

(b) Because the mth roots of unity are $e^{2\pi ik/m}$, the complete factorization of $(1-x^m)/(1-x)$ over the real numbers is $(1+x)^{[m\text{ even}]} \cdot \prod_{k=1}^{\lceil m/2\rceil-1}(1 - 2x\cos\frac{2\pi k}{m} + x^2)$. And any product of palindromials is a palindromial.

(c) Let $G(x) = g_0 + \cdots + g_c x^c$ and $H(x) = h_0 + \cdots + h_d x^d$. Clearly $g_0 = h_0 = 1$. Let k be minimal with $0 < g_k < 1$ or $0 < h_k < 1$; say $0 < g_k < 1$. Then $h_k = 0$, because $g_{c-k} = g_k$ and $g_{c-k}h_k + g_c h_0 \le 1$. But $g_k h_0 + g_{k-1}h_1 + \cdots + g_0 h_k = 1$, and all terms but $g_k h_0$ are 0 or 1. Contradiction.

(d) Since $g_1 + h_1 = 1$ we may assume that $g_1 = 1$. Then the nonzero coefficients of G can be written as a union of disjoint intervals $[a_0 \mathinner{.\,.} a_0 + k_0) \cup [a_1 \mathinner{.\,.} a_1 + k_1) \cup \cdots \cup [a_t \mathinner{.\,.} a_t + k_t)$, where $a_0 = 0$ and $k_0 > 1$ and $a_{i+1} > a_i + k_i$. If we shift those intervals by s whenever h_s is 1, the union of all of the resulting disjoint sets is $[0 \mathinner{.\,.} m)$.

Let $k = k_0$. Clearly $h_k = 1$. And we must have $k_i \le k$ for $0 \le i \le t$, to avoid overlap after shifting by k. Moreover, if $k_i < k$ for some i, where i is minimal, there will be a short gap between $a_i + k_i$ and $a_i + k$ that cannot be covered by any subsequent shift without overlap. Hence all $k_i = k$, and $T(x) = x^{a_0} + \cdots + x^{a_t}$.

(e) We have $G(1) = n$, $F_k(1) = k$, and $T(0) = H(0) = 1$. So every nonzero term of T or H is a nonzero term of $T(x)H(x) = F_m(x)/F_k(x) = F_{m/k}(x^k)$.

(f) If $nr > 1$, every factorization counted by $A(n, r)$ comes from one that's counted by $A(n/k, r)$ or by $A(n, r/l)$, for some $k\backslash n$ or some $l\backslash r$. In particular, $A(p^e, q^f) = A(p^{e-1}, q^f)[e>0] + A(p^e, q^{f-1})[f>0] + [e=f=0]$. Hence $A(p^e, q^f) = \binom{e+f}{e}$.

(g) Let p_i denote the operation of dividing n by p_i, and let q_j denote the operation of dividing r by q_j. Then every permutation π of $\{p_1, p_2, q_1, q_2\}$ defines a factorization $F_m(x) = G_\pi(x)H_\pi(x)$, by the rules $G_{p_i\alpha}(x) = F_{p_i}(x)G_\alpha(x^{p_i}), H_{p_i\alpha}(x) = H_\alpha(x^{p_i})$; $G_{q_j\beta}(x) = G_\beta(x^{q_j}), H_{q_j\beta}(x) = F_{q_j}(x)H_\beta(x^{q_j})$; $G_\epsilon(x) = H_\epsilon(x) = 1$. For example, $G_{p_1q_2p_2q_1}(x) = F_{p_1}(x)F_{p_2}(x^{p_1q_2})$, $H_{p_1q_2p_2q_1}(x) = F_{q_2}(x^{p_1})F_{q_1}(x^{p_1q_2p_2})$.

But we must avoid double-counting, because the operations $\{p_1, p_2\}$ and $\{q_1, q_2\}$ commute pairwise. There are 14 equivalence classes of permutations: $p_1p_2q_1q_2 \equiv p_1p_2q_2q_1 \equiv p_2p_1q_1q_2 \equiv p_2p_1q_2q_1$, $p_1q_1p_2q_2$, $p_1q_1q_2p_2 \equiv p_1q_2q_1p_2$, $p_1q_2p_2q_1$, $p_2q_1p_1q_2$, $p_2q_1q_2p_1 \equiv p_2q_2q_1p_1$, $p_2q_2p_1q_1$, and seven more with $p \leftrightarrow q$. So $A(p_1p_2, q_1q_2) = 14$.

(h) The Möbius polynomial for variables $\{p_1, \ldots, p_s, q_1, \ldots, q_t\}$, when the p's and q's commute pairwise, is $(1-p_1)\ldots(1-p_s) + (1-q_1)\ldots(1-q_t) - 1$.

(i) $1/((1-q_1)\ldots(1-q_t) - p) = \sum_{e\ge 0} p^e(1-q_1)^{-1-e}\ldots(1-q_t)^{-1-e}$.

[See M. Krasner and B. Ranulac, *Comptes Rendus Acad. Sci.* **204** (Paris, 1937), 397–399, as well as V. Senderov and A. Spivak, *Kvant* **29**, 1 (January–February 1998), 10–18, for comments on parts (b)–(d). N. Beluhov contributed to parts (a), (e), (f), (g), and (i). Beluhov has also discovered the amazing identity $A(p_1^e p_2^e, q_1^e q_2^e) = \sum_k (-1)^{e+k}\binom{2e}{k}^4$ (!); see *Enumer. Combinatorics and Applic.* **2:1** (2022), #S2R6, 1–11.]

118. (a) Edges p through $2n$ are defined by the vertex labels already given. For the other $p-1$ edges we must choose the labels $2n-j$ or $2n-p+j$, for $1 \le j \le \lfloor p/2 \rfloor$; there are $2^{\lfloor p/2 \rfloor}$ solutions. (For example, when $n = 7$ there are two solutions with $\{14, 11\}$ in the second part, and four with $\{14, 9\}$. One of the former has $\{0, 1, 2, 6, 7, 8, 12\}$ in the first part; one of the latter has $\{0, 1, 2, 3, 4, 11, 13\}$.)

(b) The second part labels are $\{jn + k \mid 1 \le j < r\} \cup \{nr\}$. For example, $K_{7,7}$ can be labeled with $\{0, 1, 2, 3, 4, 47, 48\}$ and $\{9, 16, 23, 30, 37, 44, 49\}$.

119. Not when $n, r \le 23$, according to calculations by F. Stappers. (Is $K_{n,n}$ *uniquely* graceful when $n = 3k + 2$ is prime?)

120. Primary items $\{1, \ldots, m\}$ for the arc labels, and m primary items vw for the arcs $v \longrightarrow w$. Also n secondary items v for the vertices, and $q = m + 1$ secondary items $\{h_0, \ldots, h_m\}$ for the holders of arc labels. There are $(m+1)m^2$ options: '$((y-x) \bmod q)$ vw $v{:}x$ $w{:}y$ $h_x{:}v$ $h_y{:}w$', for each arc $v \longrightarrow w$ and each $x \ne y$ with $0 \le x, y \le m$.

(We can greatly reduce the number of solutions by forcing some vertex v to be labeled 0, and forcing some other vertex w to be labeled with a divisor of q.)

121. $a = 7$, $b = 5$. (Subtract 3, then multiply by the inverse of $5 - 3$.)

122. Using exercise 120 we quickly (14 Mμ) discover exactly 48 solutions with $l(000) = 0$ and $l(001) = 1$. Each of them belongs to a set of 12 that are mutually equivalent, via automorphisms and antiautomorphisms followed by possible addition and multiplication, just as labelings (d) and (f) are obtained from (b) in Fig. 109. The four essentially different solutions are represented, lexicographically least, by $(l(000),\ldots, l(111)) = (0,1,2,5,12,6,8,3)$, $(0,1,2,6,12,8,5,3)$, $(0,1,2,9,6,4,11,8)$, $(0,1,3,10,11,6,2,12)$.

123. (a) Let $d = \gcd\bigl(l(w) - l(v), q\bigr)$ and $q' = q/d$, so that $l(w) - l(v) = cd$ for some $c \perp q'$. There's a unique c' such that $0 < c' < q'$ and $cc' \equiv 1$ (modulo q').

permutation
Möbius polynomial
Krasner
Ranulac
Senderov
Spivak
Beluhov
amazing identity
Stappers
uniquely

There are d solutions to the simultaneous equations $\bigl(a\cdot l(v)+b\bigr) \bmod q = 0$ and $\bigl(a\cdot l(w)+b\bigr) \bmod q = d$, namely $a=a_k$ and $b=(-a_k\cdot l(v)) \bmod q$, where $a_k=c'+kq'$ and $0\le k<d$. Hence we want to prove that $a_k \perp q$ for at least one value of k.

Say that the prime p is "in d" if $p\backslash q$ but $p\nmid q'$. (For example, if $d=10$ and $q=60$, then only 5 is in d.) We can write $d=rd'$, where the prime factors of r are in d but those of d' are not. If p divides $\gcd(a_k,q)=\gcd(c'+kq',q)$ it must be in d; otherwise it would divide q' but not c'. Therefore $\gcd(a_k,q)=\gcd(a_k,r)$. And the values of $a_k \bmod r$ for $0\le k<d$ are d' copies of $\{0,1,\ldots,r-1\}$, because $q'\perp r$.

(b) Exactly $d'\varphi(r)=d\prod_{p\text{ in }d}(1-\frac{1}{p})$ graceful labelings are produced by that construction. Furthermore, different values of k give a different l': Let u and u' be the vertices for which $u\longrightarrow u'$ and $\bigl(l(u')-l(u)\bigr) \bmod q=1$. Then $\bigl(l'_k(u')-l'_k(u)\bigr) \bmod q=a_k$.

(c) It suffices to find the essentially different cases that are *normalized*, in the sense that $l(v)=0$ and $l(w)\backslash q$. Begin with the set of all normalized solutions (a), grouping the solutions for divisor d into equivalence classes of size $d\prod_{p\text{ in }d}(1-\frac{1}{p})$ as in (b). Then, for each automorphism or antiautomorphism α, apply α to a representative of each class. If the result is in a different class, after normalization by an affine transformation, merge the classes. Repeat until no more merging is possible. (We need only consider enough α's to generate them all under composition.)

124. (a) Denote a labeling by the tuple $l(\mathrm{a})l(\mathrm{b})l(\mathrm{c})l(\mathrm{d})$. If we choose $v=\mathrm{b}$ and $w=\mathrm{c}$, the initial affine equivalence classes in answer 123(c) turn out to be $\{1024\}$, $\{4021\}$ for $d=2$ and $\{1034,5032\}$, $\{2035,4031\}$ for $d=3$, since there are no solutions for $d=1$.

This digraph has two automorphisms, () and (b c). It also is self-converse, so it has two antiautomorphisms, one for each automorphism; they are (a d) and (a d)(b c).

Let $\alpha=(\mathrm{a\ d})$. Then $1024\alpha=4021$ and $2035\alpha=5032$; so the classes of equivalent labelings are $\{1024,4021\}$ and $\{1034,2035,4031,5032\}$ after the first step of merging.

Now let $\alpha=(\mathrm{b\ c})$. We have $1024\alpha=1204$, which normalizes affinely to 1024. So no further merging occurs, and there are just two essentially distinct classes of equivalent solutions. (We needn't try $\alpha=(\mathrm{a\ d})(\mathrm{b\ c})$, which is generated by the others.)

(Alternatively, we could have chosen $v=\mathrm{a}$ and $w=\mathrm{b}$, say. Then the initial classes would have been $\{0143\}$, $\{0153\}$, $\{0243\}$, $\{0253\}$. The antiautomorphism (a d) would have merged them to $\{0143,0253\}$ and $\{0153,0243\}$. The automorphism (b c) would then have made no further change.)

(b) Choose $v=\mathrm{a}$ and $w=\mathrm{d}$, say, getting six initial classes $\{043125\}$, $\{015243\}$, $\{031245\}$, $\{034215\}$, $\{045213\}$, $\{053241\}$. The antiautomorphism (a f)(b e)(c d) merges them to $\{034215,043125\}$, $\{015243\}$, $\{031245,053241\}$, $\{045213\}$; four classes only.

125. Set `FIRST`[l] $\leftarrow -1$ for $0\le l<q$. Then do the following steps for $l=1,2,\ldots,m$: Set $v\leftarrow$ `LO`[l], $w\leftarrow(v+l) \bmod q$, $t\leftarrow$ `FIRST`[v], `FIRST`[v] $\leftarrow w$, `NEXT`[l] $\leftarrow t$.

(A similar algorithm will create arrays `FIRSTP` and `NEXTP` with which all *predecessors* of any given vertex can be visited efficiently. We can also readily create `FIRST`, `NEXTL`, and `NEXTH` from the `LO` array of a graceful *undirected* graph.)

126. (a) Let $f(-1)=-1$, otherwise $f(x)=\bigl(a(x-b)\bigr) \bmod q$. Then `LO`$'$[$(al) \bmod q$] $=f($`LO`[l]$)$; `FIRST`$'$[$(a(l-b)) \bmod q$] $=f($`FIRST`[l]$)$; `NEXT`$'$[$(al) \bmod q$] $=f($`NEXT`[l]$)$; `NAME`$'$[$(a(l-b)) \bmod q$] $=$ `NAME`[l].

(b) `LO`, `FIRST`, and `NEXT` are unchanged; `NAME`$'$[l]α $=$ `NAME`[l].

(c) `LO`$'$[$q-l$] $=\bigl($`LO`[l]$+l\bigr) \bmod q$; `NAME`$'$[l]α $=$ `NAME`[l]; `FIRST`$'$ and `NEXT`$'$ must be computed from `LO`$'$ using exercise 125.

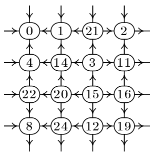
totient function
automorphisms
self-converse
antiautomorphisms
essentially distinct
`NEXTL`
graph representation

127. Now $q = 20$, and D^* has the same [anti]automorphisms as D. Choosing $v = 000$ and $w = 001$ in answer 123(c) yields respectively (46, 48, 14, 0, 0) affine classes for $d = (1, 2, 4, 5, 10)$; the classes for $d = 4$ are pairs of labelings, the others are singletons.

Automorphisms merge every class for $d > 1$ with at least one class for $d = 1$. So we can confine attention to the 23 labelings with $l(001) = 1$ and $l(010) < l(100)$.

An antiautomorphism finally leaves just seven classes: {`012acj5g`, `013cif28`, `016e745g`}, {`01649ehg`, `018aid54`, `0196edg4`}, {`0165icf8`, `01bec9ag`, `01becajg`}, {`0169ecjg`, `01bac568`, `01bac6f8`}, {`016acfb8`, `016j9ceg`, `0198e6b4`}, {`01358ife`, `014eb976`, `014hbje6`, `017fg56i`}, {`0135i8fe`, `01657fgi`, `01b9e476`, `01bjeh46`}. (Here the extended hexadecimal digits `0` through `j` encode the labels 0 through 19.)

128. Very much so, with millions and millions of labelings! Here's one of the 32 solutions for which $l(0000) = 0$, $l(0001) = 1$, $l(0010) = 2$, $l(0100) = 4$, $l(1000) = 8$, and $l(1111) = 15$, all found in 200 Gμ. By arranging the vertices of this interesting digraph as a Karnaugh map (see exercise 7.2.1.1–17), we can exhibit it as a "magical 4×4 torus."



129. (a) It suffices to consider tuples with $x_1 = 0$. Then $\mathcal{D}_2$ has two classes $\{00, 02\}$, $\{01\}^*$, and $\mathcal{D}_3$ has six: $\{000, 032\}$, $\{001, 011, 021, 031\}$, $\{002, 010, 022, 030\}^*$, $\{003, 033\}^*$, $\{012, 020\}$, $\{013, 023\}^*$. (Those marked with $*$ define a self-converse graceful digraph; the others define a converse pair. For example, $\{000, 032\}$ gives $K_1 \longrightarrow \overline{K_3}$, $\overline{K_3} \longrightarrow K_1$.)

(b) Use arithmetic mod q. If $a \perp q$ and $aa' = 1$, define $ax = y_1 \ldots y_m$ and $-ax^T = z_1 \ldots z_m$, where $y_l = a(x_{a'l} - x_{a'})$ and $z_l = 1 - l - y_l$. Reject x if $x > ax$ or $x > -ax^T$ lexicographically, for some $a \perp q$. The accepted tuples are inequivalent.

(c) The answer is $\sum_{a=1}^{m} [a \perp q] \sum_{b=0}^{m} \bigl(f(a,b,q) + g(a,b,q)\bigr)/(2q\varphi(q))$ by "Burnside's lemma," where $f(a,b,q)$ and $g(a,b,q)$ are respectively the number of x with $D(x) = aD(x) + b$ and $D(x)^T = aD(x) + b$. Let $t(\alpha,\beta,q) = \gcd(\alpha,q)[\gcd(\alpha,q)\backslash\beta]$; this is the number of $x \in [0\,..\,q)$ such that $\alpha x \equiv \beta$ (modulo q), when $\alpha, \beta \in [0\,..\,q)$.

Let $f(l,a,b,q) = \bigl(a^s l < l?\ 1\colon t(a^s - 1, -b(a^{s-1} + \cdots + a + 1), q)\bigr)$, where $s > 0$ is minimum with $a^s l \le l$. (All arithmetic is mod q.) Then $f(a,b,q) = \prod_{l=1}^{q-1} f(l,a,b,q)$.

Let $g(l,a,b,q) = \bigl((-a)^s l < l?\ 1\colon t(a^s - 1, -b(a^{s-1} + \cdots + a + 1) - l(s \bmod 2), q)\bigr)$, where $s > 0$ is minimum with $(-a)^s l \le l$. Then $g(a,b,q) = \prod_{l=1}^{q-1} g(l,a,b,q)$.

(For example, it's 12502550 when $m = 9$; see OEIS A341884. The totient function $\varphi(n)$ is asymptotically not much less than n. In fact, $\liminf_{n\to\infty} (\ln\ln n)\varphi(n)/n = e^{-\gamma}$; see Hardy and Wright, *An Introduction to the Theory of Numbers*, Theorem 328.)

130. (a) Since $l_{2k+1} - l_{2k} = 2k+1$ and $l_{2k+1} - l_{2k+2} = 2k+2$, these labels are actually graceful for the nonoriented path P_n. Modulo $q = n$, the edge labels 2, 4, …, $n-2$ become arc labels $n-2$, $n-4$, …, 2.

(b) Use the labels $l_{2k} = k$, $l_{2k+1} = r-1-k$ in the first half. Then define $l_{2r-1-k} = l_k + r + 1$. (This elegant construction is due to D. F. Hsu [*Lecture Notes in Math.* **824** (1980), 134–140], whose paper with G. S. Bloom [*Congressus Numerantium* **35** (1982), 91–103] introduced the notion of graceful digraphs and proved Theorem H.)

131. Let $l'(v) = (n+1)l(v)$ for $v \in D$, $l'(w_k) = k$ for the other vertices $\{w_1, \ldots, w_n\}$. [*SIAM Journal on Algebraic and Discrete Methods* **6** (1985), 519–536.]

132. $D = \vec{P_3}$, $l(v_0) = 2n+1$, $l(v_1) = 0$, $l(v_2) = n+1$, and $l(w_k) = k$ for $1 \le k \le n$.

133. Yes, because $K_{m,n}$ is α-graceful with labels $\{0, 1, \ldots, m-1\}$ in one part.

134. (Answer left to the reader: Enjoy! Consider also the analogs of exercises 91–95, as well as the behavior of *random* graceful digraph labelings as $m \to \infty$. Many results have been reported by F. Stappers at `archive.org/details/graceful_digraphs_6`.)

digits, extended hexadecimal
Karnaugh map
magical 4×4 torus
self-converse
Burnside's lemma
congruence enumeration
OEIS
totient function
γ
Hardy
Wright
Hsu
Bloom
Stappers

135. If D were graceful, its arc labels would sum to $1+\cdots+m = q(q-1)/2$. That sum is also congruent (modulo q) to $\sum_v (d^+(v)-d^-(v))l(v)$, which is even.

136. For each k with $1 \le k \le m$, we can reverse the orientations on the arcs labeled k and $m+1-k$. [See the paper cited in answer 131, which introduced digracefulness.]

137. (a) A, E, C, F, B, D, G, H, I, J, K, L. (Note that A is the transitive tournament $K_5^{\rightarrow}$.)

(b) C, G, H, I, J, K are not graceful; the other six are uniquely graceful. (The lexicographically smallest `LO[1]LO[2]`...`LO[10]` tables for A, B, D, L are respectively 0040210442, 0010770742, 0010210742, 0017214742; E = B^T; F = D^T. Each labeling can be obtained from any of the others by reversing pairs as in exercise 136.)

(c) The four unlabeled tournaments for $n=4$ are A′, B′, C′, D′, obtained by removing the bottom vertices of A, B, C, D. The self-converse D′ is ungraceful; the others are uniquely graceful, with `LO` tables 002102 and 001042 for A′ and B′; C'^T = B.

When $n=3$, A″ is uniquely graceful but B″ is the ungraceful $C_3^{\rightarrow}$.

(d) Let $v = q = \binom{n}{2}+1$. Given a graceful tournament on vertices $\{1,\ldots,n\}$, with labels $a_j = l(j)$, suppose arc $j \longrightarrow k$ is labeled l and arc $k' \longrightarrow j'$ is labeled $q-l$. Then $a_k \ominus a_j$ and $a_{k'} \ominus a_{j'}$ are two differences equal to l, so we have a cyclic $(v,n,2)$-difference set. (We'll have $j=k'$ and $k=j'$ when $l=q/2$, but never $j=j'$ and $k=k'$.) Conversely, by assigning labels from such a difference set, we get a graceful tournament if we define either ($j \longrightarrow k$ and $k' \longrightarrow j'$) or ($k \longrightarrow j$ and $j' \longrightarrow k'$) whenever $k \ominus j = k' \ominus j'$. [This connection was apparently first noted by Kumudakshi in her Ph.D. thesis (Mangalore: National Institute of Technology Karnataka, July 2016), Proposition 2.2.5.]

(e) These residues form a cycle (1 7 12 10 33 9 26 34 16) that defines a symmetrical graceful tournament, in which $u \longrightarrow v$ whenever v is one of the next four elements after u. (But the transitive tournament $K_9^{\rightarrow}$ is *not* graceful.) [In place of 7, R. D. Carmichael mentioned the equally good multiplier 16, on pages 437–438 of his *Introduction to the Theory of Groups of Finite Order* (1937); he probably learned about this remarkable difference set from someone else, so its origin is obscure. A computer search by D. M. Gordon has shown that no other cyclic difference sets with $\lambda = 2$ exist for $n \le 10^{10}$; see *J. Algebraic Combinatorics* **55** (2022), 109–115.]

138. Say G is weakly digraceful with *tolerance* t if it can be gracefully oriented using just $m+t$ arcs. Calculations by Filip Stappers show that, for all 1044 graphs with up to 7 vertices, exactly (1013, 26, 4, 1) require tolerance $t =$ (0, 1, 2, 3). (Only $3K_2$ needs tolerance 3; only $2K_2$, $L_{3,4}$, K_6, and K_7 need tolerance 2. For K_7 we can use the vertex labels $\{0,1,2,4,7,15,19\}$, mod 24, with all arcs $u \longrightarrow v$ going from $\min(u,v)$ to $\max(u,v)$ *except* that $2\longrightarrow 1$, $4\longrightarrow 2$, $7\longrightarrow 4$, $19\longrightarrow 15$, $15\longrightarrow 0$; the "tolerant" arcs $15\longrightarrow 7$ and $15\longrightarrow 1$ also pair up with their reversals $7\longrightarrow 15$ and $1\longrightarrow 15$.)

It seems likely that all connected graphs are weakly digraceful with *bounded* tolerance, because each modulus $q = m+t+1$ gives a "fresh start" for achieving gracefulness.

139. The arc labels between k and $k+1$ are $\pm(2k+1)$ (modulo q), where $q = 2n+1$, except for two values of k. The exceptional values are $k = \lfloor (n-1)/2 \rfloor$, when the labels are ± 1, and $k = n$, when they are $\pm(n-(-1)^n)$. Altogether, they are therefore $\Delta(n) = \{\pm 1, \pm 3, \ldots, \pm(2n-1)\}$, because the "missing" case $\pm(2k+1)$ for $k = \lfloor (n-1)/2 \rfloor$ turns out to be $\pm(n-(-1)^n)$. Finally, $\Delta(n)$ is the same as $\{\pm 1, \pm 2, \ldots, \pm n\}$ (modulo q). [*Discrete Mathematics* **261** (2003), 116.]

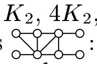
140. Regard T as rooted at v, with subtrees $T_1, \ldots, T_d$ where $|T_1| \ge \cdots \ge |T_d|$, and number the vertices $v_0, v_1, \ldots, v_m$ in preorder. Let $l(v_0) = 0$; and for $k = 1, \ldots, \lceil m/3 \rceil$ let $l(v_k)$ be the least positive integer such that $l(v_k) \ne l(v_j)$ and $|l(v_k) - l(\mathrm{parent}(v_k))| \ne$

historical notes
transitive tournament
self-converse
Kumudakshi
Carmichael
Gordon
tolerance
Stappers
lollipop $L_{m,n}$
preorder

$\bigl|l(v_j)-l(\mathrm{parent}(v_j))\bigr|$ for $1\le j<k$. At most $3(k-1)$ values are excluded, hence $l(v_k)\le m$. Let $C=\bigl\{\bigl|l(v_k)-l(\mathrm{parent}(v_k))\bigr| \bigm| 1\le k\le\lceil m/3\rceil\bigr\}$ be the "colors" used.

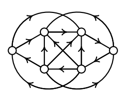
The remaining $m-\lceil m/3\rceil\le 2m/3$ vertices are leaves adjacent to v, by hypothesis. So we can label them with the *negatives* of the unused colors, $-(\{1,\ldots,m\}\setminus C)$.

141. The labels $\{0,1,2,5,12,23,29\}$ give all differences $\{\pm1,\pm2,\ldots,\pm18\}$ (modulo 37), with ±1, ±10, ±11 occurring twice. For (i), let $0 \mathrel{-\!\!\!/\!\!-} 1$, $2 \mathrel{-\!\!\!/\!\!-} 29$, $12 \mathrel{-\!\!\!/\!\!-} 23$; for (ii), let $1 \mathrel{-\!\!\!/\!\!-} 2 \mathrel{-\!\!\!/\!\!-} 12 \mathrel{-\!\!\!/\!\!-} 1$. For (iii), work modulo 41 and let $0 \mathrel{-\!\!\!/\!\!-} 36 \mathrel{-\!\!\!/\!\!-} 18 \mathrel{-\!\!\!/\!\!-} 31 \mathrel{-\!\!\!/\!\!-} 0$, $1 \mathrel{-\!\!\!/\!\!-} 2 \mathrel{-\!\!\!/\!\!-} 22 \mathrel{-\!\!\!/\!\!-} 28 \mathrel{-\!\!\!/\!\!-} 1$. [Each of these labelings is essentially unique. The other graphs on 7 and 8 vertices that are uniquely rainbow graceful are $\overline{3K_1\oplus 2K_2}$, $\overline{4K_1\oplus K_{1,3}}$, $\overline{3K_1\oplus K_{1,4}}$, $\overline{3K_1\oplus K_2\oplus C_3}$, $\overline{3K_1\oplus K_2\oplus P_3}$, K_8.]

142. P. Adams and J. Appleton (see S. I. El-Zanati and C. Vanden Eynden, *Mathematica Slovaca* **59** (2009), 1–18) found that $G^{\leftrightarrow}$ is graceful except in the following 18 cases: For $n=6$ vertices, $\overline{4K_1\oplus K_2}$, the complement of K_2. For $n=7$, the complements of $K_{3,3}$, $K_{1,5}$, K_2, and K_1. For $n=8$, the complements of $K_{4,4}$, $K_{3,4}$, $K_{2,6}$, $K_{1,6}$, $K_{1,5}$, $K_{2,2}$, $K_3\oplus K_2$, $4K_2$, K_3, $3K_2$, $2K_2$, $K_{1,2}$, and K_2. (The "most rainbow graceful" 8-vertex graph is : There are 41,636 essentially different ways to label it!)

[It turns out that 43 copies of K_7 *can* be packed perfectly into K_{43}, but not cyclically. On the other hand, 29 copies of $\overline{4K_1\oplus K_2}$ *cannot* be packed perfectly into K_{29}, cyclically or otherwise. It's the smallest example of an m-edge graph whose copies can't exactly cover K_{2m+1}. See S. Hartke, P. R. J. Östergård, D. Bryant, and S. I. El-Zanati, *Journal of Combinatorial Designs* **18** (2010), 94–104.]

143. No; $\overline{4K_1\oplus K_2}$ is digraceful (answer 138), yet not rainbow graceful (answer 142). (It has 156 essentially distinct graceful orientations, 18 of which are self-converse. The most graceful of these, with 5 labelings, is shown.)



144. (a) There are n^3-1 nonzero triples, in equivalence classes of size $n-1$, hence $(n^3-1)/(n-1)$ classes. Each class has a unique element whose first nonzero component is 1; thus $a_1=1$ in n^2 classes, $(a_1,a_2)=(0,1)$ in n, and $(a_1,a_2,a_3)=(0,0,1)$ in 1.

(b) $a_1+2a_3\equiv 0$ (modulo 3) $\iff a_1\equiv a_3$. So the answer is $\{(0,1,0),\ (1,0,1),\ (1,1,1),\ (1,2,1)\}$. (In general $a_1b_1+a_2b_2+a_3b_3=0$ has n^2-1 nonzero solutions (a_1,a_2,a_3) in F, belonging to $(n^2-1)/(n-1)$ classes, when $[b_1,b_2,b_3]$ is nonzero.)

(c) The nonzero vectors $[b_1,b_2,b_3]$, $[b_1',b_2',b_3']$ are linearly independent when one isn't a multiple of the other. In that case the homogeneous equations $a_1b_1+a_2b_2+a_3b_3=a_1b_1'+a_2b_2'+a_3b_3'=0$ have $n-1$ nonzero solutions (a_1,a_2,a_3), all equivalent.

[See 7–(57) for the case $n=2$; see also exercise 7–19.]

145. (a) It's an immediate consequence of the definitions; there are $m=\binom{n+1}{2}$ edges.

(b) If $\pi^3=c_1\pi^2+c_2\pi+c_3$, then $\pi^{3p}=c_1\pi^{2p}+c_2\pi^p+c_3$. Hence the other roots are π^p and π^{p^2}. [And $c_1=\pi+\pi^p+\pi^{p^2}$, $-c_2=\pi^{1+p}+\pi^{1+p^2}+\pi^{p+p^2}$, $c_3=\pi^{1+p+p^2}$.]

(c) Since $\pi^{k+1}=c_1\pi^k+c_2\pi^{k-1}+c_3\pi^{k-2}$, $a_1'=a_2+c_1a_1$, $a_2'=a_3+c_2a_1$, $a_3'=c_3a_1$.

(d) $b_1'=b_2$, $b_2'=b_3$, $b_3'=(b_1-c_1b_2-c_2b_3)/c_3$.

(e) Eschewing parentheses and commas, they are 001, 010, 100, 403, 132, 223, 031, 310, 304, 244, 241, 211, 411, 212, 421, 312, 324, 444, 042, 420, 302, 224, 041, 410, 202, 321, 414, 242, 221, 011, 110, 003. Since $\pi^{31}=3$, we have $\pi^{31+k}=3\pi^k$.

(f) Let $v=p^2+p+1$. Then $\{1,\pi^v,\pi^{2v},\ldots,\pi^{(p-2)v}\}=\{1,2,\ldots,p-1\}$, so the triples for $\{1,\pi,\ldots,\pi^{v-1}\}$ are all the points. The given labels L are the points of the line $[1,0,0]$. Hence the points of the line $[1,0,0]\alpha^k$ are $(L+k)\bmod v$, and we have a cyclic $(v,p+1,1)$-difference set. (For example, $L=\{0,1,6,18,22,29\}$ when $p=5$.)

(g) Let F be the field of p^{3e} elements, specified by a primitive polynomial modulo p, and let π be a root of f in F. Then the subfield F_0 of p^e elements is $\{0,1,\pi^v,\ldots,$

unique
Adams
Appleton
El-Zanati
Vanden Eynden
complement
exact cover
Hartke
Östergård
Bryant
El-Zanati
self-converse
linearly independent
homogeneous equations
primitive polynomial

$\pi^{(p^3-2)v}\}$, where $v = p^{2e} + p^e + 1$. The polynomial $f_0(x) = (x-\pi)(x-\pi^{p^e})(x-\pi^{p^{2e}}) = x^3 - c_1x^2 - c_2x - c_3$ is primitive for F and has coefficients in F_0. Proceed as before.

When $n = 8$ we can use $f(x) = x^9 - x^5 - 1$. Then $\omega = \pi^v = \pi^{73} = \pi^8 + \pi^7 + \pi^4 + \pi + 1$ is a primitive root for F_0, and we have $f_0(x) = x^3 - (\omega^2+1)x^2 - x - \omega$. Using octal notation with $\mathit{0} = 0$, $\mathit{1} = 1$, $\mathit{2} = \omega$, …, $\mathit{7} = \omega^2 + \omega + 1$, the points 1, π, π^2, …, π^{72} are *001*, *010*, *100*, *512*, *777*, *603*, *451*, *655*, *131*, *602*, *441*, *755*, *423*, *175*, *242*, *304*, *276*, *044*, …, *151*, and they yield the graceful rainbow labels $\{0, 1, 17, 39, 41, 44, 48, 54, 62\}$ for K_9.

[*Transactions of the Amer. Math. Soc.* **43** (1938), 377–385. T. P. Kirkman had discovered cyclic difference sets "by accident" for the projective planes of orders 2, 3, 4, 5, and 8, in *Trans. Hist. Soc. Lancashire and Cheshire* **9** (1857), 127–142. A famous conjecture that K_{n+1} is rainbow graceful if *and only if* n is a prime power has been verified for all $n \le 2\cdot 10^{10}$; see D. M. Gordon, *J. Algebraic Combinatorics* **55** (2022), 109–115.]

146. (a) It suffices to consider tuples with $x_1 = 0$. Then $\mathcal{R}_2$ has two classes {00, 01, 03, 04}, {02}, and $\mathcal{R}_3$ has eleven: {000, 011, 015, 050, 054, 065}, {001, 002, 024, 041, 063, 064}, {003, 026, 031, 034, 046, 062}, {004, 061}, {005, 013, 021, 044, 052, 060}, {006, 014, 030, 035, 051, 066}, {010, 055}, {012, 020, 022, 043, 045, 053}, {016, 025, 032, 033, 040, 056}, {023, 042}, {036}. For example, the first and fourth classes give $K_{1,3}$.

(b) Use arithmetic mod q. Reject x if $x > ax$ lexicographically for some $a \perp q$, where $ax = y_1 \ldots y_m$ is defined by first setting $z_{al} \leftarrow ax_l$ if $al \le m$, otherwise $z_{q-al} \leftarrow a(x_l + l)$; then $y_l = z_l - z_1$. The accepted tuples are inequivalent.

(c) It's $\sum_{a=1}^{2m}[a \perp q]\sum_{b=0}^{2m} f(a,b,q)/(q\varphi(q))$, where $f(a,b,q) = \prod_{l=1}^{m} f(l,a,b,q)$ and $f(l,a,b,q) = (a^s l = l?\ t(a^s-1, -b(a^{s-1}+\cdots+a+1), q)\colon q - a^s l = l?\ t(a^s-1, l-b(a^{s-1}+\cdots+a+1), q)\colon 1)$, where $s > 0$ is minimum with $a^s l \le l$ or $q - a^s l \le l$. (Compare with answer 129(c). We get 943532049 when $m = 9$; see OEIS A342357.)

147. This conjecture was introduced by S. I. El-Zanati, C. Vanden Eynden, and N. Punnim, *Australasian J. Combinatorics* **24** (2001), 209–219. In fact, they conjectured that *every bipartite graph G* with no isolated vertices has an "ordered graceful rainbow labeling," in which the smaller endpoint of every edge belongs to one part and the larger endpoint belongs to the other. (One such labeling for C_6 is (041327).)

148. True and true.

149. The unique answer is `chord` — `chore` — `chose` — `chase` — `chasm` — `charm` — `chard` — `chord`. (But one might argue that an induced cycle is always "chordless.")

150. Yes. One must check that $d(\texttt{cords},\texttt{costs}) = 3$ and $d(\texttt{colts},\texttt{carts}) = 3$ in `WORDS`(5757): The first is true because `corts` and `cosds` are nonwords, according to the Stanford GraphBase; the second is true because `corts` and `calts` are nonwords.

151. (a) $\binom{n_1}{2} + \binom{n_2}{2} + \cdots + \binom{n_r}{2}$.

(b) $n_1!\,n_2!\ldots n_r!\,t_2!\,t_3!\,t_4!\ldots$, when t_q of the n_k are equal to q.

(c) 4. (This question is too easy. Hamming distance is defined in exercise 7–23.)

(d) Suppose $x_1 \ldots x_r$ — $y_1 \ldots y_r$ because $x_j \ne y_j$, and $x_1 \ldots x_r$ — $z_1 \ldots z_r$ because $x_k \ne z_k$. Then $y_1 \ldots y_r$ — $z_1 \ldots z_r$ if and only if $j = k$.

(e) $K_{2,1,1}$. It contains two triangles that share an edge; hence the images of both triangles vary in only one constituent, by (d). But then all vertices are adjacent.

(f) Suppose we change coordinates k_0, k_1, …, k_4 as we go around the cycle. Then $k_0 \ne k_1 \ne \cdots \ne k_4 \ne k_0$, by (d). And each k_i must equal some k_j for $j \ne i$.

152. Every induced C_7 of a Hamming graph is equivalent to 000 — 100 — 110 — 111 — 121 — 021 — 001 — 000. So we can start by dividing `WORDS`(5757) into $\binom{5}{2} =$

octal notation
Kirkman
Gordon
Burnside's lemma
OEIS
El-Zanati
Vanden Eynden
Punnim
bipartite graph
ordered graceful rainbow labeling
unique answer
chordless
joke
Stanford GraphBase
Hamming distance
$K_{2,1,1}$

10 families of subgraphs in which two of the coordinates are constant. (The largest such subgraphs are `*a*e*`, `*a**s`, `*o**s`, and `***es`, with sizes 305, 316, 329, and 371.)

To find all solutions within each subgraph, count the frequency of each letter in each coordinate position. Choose the coordinates (i, j, k) that will contain respectively $(3, 2, 2)$ letters in the solution, with $j < k$. A word is "unsupported" if any of its letters in positions (i, j, k) have frequencies less than $(2, 3, 3)$. There must also be at least one letter, in each of coordinates (i, j, k), whose frequency *exceeds* $(2, 3, 3)$. Discard unsupported words (and update the frequencies) until all words are supported and all frequencies are satisfactory. Then visit the solutions, of which there are 69457.

A solution is isometric if and only if three specific five-letter strings, found as in answer 150, are nonwords. Exactly 6879 solutions survive this test — including just one that belongs to `WORDS`(1000), namely `beams` — `seams` — `seems` — `seeds` — `sends` — `bends` — `beads` — `beams`. (Furthermore, exactly (2628, 2088) of the 5757 words participate in at least one (induced, isometric) cycle; (225, 298) in only one of them. The champion words are `pares`, in 2543 induced cycles; `later`, in 233 isometric cycles.)

unsupported
backtrack
strict embedding
tied-path graphs

153. (a) To satisfy (i), permute the elements with coordinate k. To satisfy (ii), permute the coordinates according to their first use.

(b) Straightforward backtrack suffices, branching on the possible $f(v_i)$ adjacent to $f(v_{i'})$. Also ensure that, for all $0 \le j < i$ and $j \ne i'$, the Hamming distance d_H satisfies $[v_j \not\!\!{-}\, v_i] < d_H(f(v_i), f(v_j)) \le d(v_i, v_j)$.

154. (a) Yes. Any strict embedding of G also strictly embeds all G's induced subgraphs.

(b) If not connected, one of its components is nonembeddable (and induced).

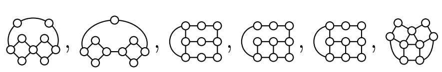
(c) True. Suppose $G \setminus v$ is disconnected, with induced components G' and G'', where G'' isn't embeddable. Then $G \setminus v'$ is connected for some v' in G'; it contains G''.

155. Let $(\mathcal{C}_n, \mathcal{H}_n, \mathcal{M}_n)$ be the n-vertex graphs that are respectively (connected, connected and Hamming embeddable, MNH). Clearly $\mathcal{H}_3 = \mathcal{C}_3$. Given lists of $\mathcal{C}_n$ for $4 \le n \le 9$, exercise 154 tells us that we can compute $\mathcal{H}_n$ and $\mathcal{M}_n$ as follows: Start with $\mathcal{H}_n$ and $\mathcal{M}_n$ empty. For each $G \in \mathcal{C}_n$, test if all n of its subgraphs $G \setminus v$ are either disconnected or in $\mathcal{H}_{n-1}$. If not, do nothing. Otherwise use exercise 153(b) to test if G has a Hamming embedding. If so, put G into $\mathcal{H}_n$; otherwise put G into $\mathcal{M}_n$.

The resulting sizes $(|\mathcal{H}_4|/|\mathcal{C}_4|, \ldots, |\mathcal{H}_9|/|\mathcal{C}_9|)$ turn out to be (5/6, 11/21, 36/112, 117/853, 469/11117, 2023/261080); and $(|\mathcal{M}_4|, \ldots, |\mathcal{M}_9|) = (1, 2, 0, 1, 1, 6)$.

The MNH graphs for $n \le 8$ all turn out to be "tied-path graphs," namely the graphs $P(n_1, \ldots, n_k)$ with $2{+}n_1{+}\cdots{+}n_k$ vertices and $k{+}n_1{+}\cdots{+}n_k$ edges that are obtained by tying together the endpoints of paths $P_{n_1+2}, \ldots, P_{n_k+2}$: $\mathcal{M}_4 = \{P(0, 1, 1)\}$; $\mathcal{M}_5 = \{P(1, 2),\ P(1, 1, 1)\}$; $\mathcal{M}_6 = \emptyset$; $\mathcal{M}_7 = \{P(1, 1, 3)\}$; $\mathcal{M}_8 = \{P(2, 2, 2)\}$.

If we knew only these results, we'd be tempted to conjecture falsely that $P(1, 1, 5)$ and $P(3, 3, 3)$ are MNH. But all such hopes are shattered by

$\mathcal{M}_9 = \Big\{$  $\Big\};$

we might still conjecture tentatively, however, that all MNH graphs are planar.

156. In a normalized embedding, say that i is "k's pioneer for c" if $i = \min\{j \mid x_{jk} = c\}$. Then 0 is every coordinate's pioneer for 0. But a positive i cannot be a *double* pioneer; v_i can't be breaking records in two different coordinates, because it differs from its parent in only one place. Let $p(k, c)$ be k's pioneer for c, if it exists.

We shall prove, by induction on $i > 0$, that at most one normalized label $l(v_i)$ is isometrically consistent with $l(v_0), \ldots, l(v_{i-1})$. Suppose we could legitimately set either $x_{ik} = a$ or $x_{ik} = b$, where $a < b$, and let $j = p(k, a)$. Then $j < i$, and $d(v_j, v_i)$ takes on two different values when we set $x_{ik} = a$ and $x_{ik} = b$. Contradiction.

Now suppose moves are legitimate in two different coordinates, $k < l$, so that if $(x_{i'k}, x_{i'l}) = (a, b)$ we could set (x_{ik}, x_{il}) to either (a', b) or (a, b'). If $a' > a$, let $j = p(k, a)$ and $t = x_{jl}$. Then $d(v_i, v_j) = \Delta + [a \neq a'] + [t \neq b] = \Delta + [a \neq a] + [t \neq b']$ for some Δ; consequently $1 + [t \neq b] = [t \neq b']$, and we must have $t = b$. Let $h = p(l, b)$ and $t' = x_{ih}$. Then $d(v_i, v_h) = \Delta' + [t' \neq a'] + [b \neq b] = \Delta' + [t' \neq a] + [b \neq b']$; consequently $[t' \neq a'] = [t' \neq a] + 1$ and $t' = a$. Hence $h = p(k, a) = j$, and j is a double pioneer! So $a = b = 0$. Finally let $g = p(k, 1)$. Then $x_{gl} = 0$; and $d(v_i, v_g) = \Delta'' + [1 \neq a'] + [0 \neq 0] = \Delta'' + [1 \neq 0] + [0 \neq b']$, a contradiction. A similar contradiction arises when $a' < a$.

So the desired algorithm is simplicity itself: To find $l(v_i)$, there are fewer than $2i$ candidates; for $0 \leq j < i$ we need $O(i)$ operations to test that $d\bigl(l(v_i), l(v_j)\bigr)$ is correct. If a candidate succeeds, we know $l(v_i)$, and no other candidates need be examined. If no candidate succeeds, there's no isometric embedding. Total time is $O(n^4)$, usually less.

[See *Discrete Applied Mathematics* **7** (1984), 221–225, also for exercise 157.]

157. (a) There are three kinds of vertices: corner (C, with degree 2); interior (I, with degree 4); other (O, with degree 3). There are four types of edges, which we may call CO, II, IO, OO. The relations OO $\bowtie$ OO, OO $\bowtie$ II, II $\bowtie$ II always hold. Each CO or IO is related to itself and to three others "parallel" to it.

(b) True. For example, $(0 \,\text{—}\, 1) \bowtie (1 \,\text{—}\, 2) \bowtie (2 \,\text{—}\, 3) \not\bowtie (0 \,\text{—}\, 1)$.

(c) Clearly $\bowtie$ is reflexive and symmetric. If $(u \,\text{—}\, v) \bowtie (u' \,\text{—}\, v') \bowtie (u'' \,\text{—}\, v'')$ in *any* isometric Hamming embedding, and if $u_k \neq v_k$, $u'_{k'} \neq v'_{k'}$, $u''_{k''} \neq v''_{k''}$, where u_k denotes the kth coordinate of $l(u)$, then $k = k' = k''$. And if the embedding is ternary, we must also have $\{u, v\} \cup \{u'', v''\} \neq \emptyset$, hence $(u \,\text{—}\, v) \bowtie (u'' \,\text{—}\, v'')$.

(d) Let there be r equivalence classes, and let $u^{(k)} \,\text{—}\, v^{(k)}$ represent class k. Assign label $l(w) = w_1 \ldots w_r$ to vertex w, where $w_k = \bigl(d(w, u^{(k)}) - d(w, v^{(k)})\bigr) \bmod 3$.

158. The graph with labels $\{00, 10, 20, 11, 21, 31\}$ answers (i); for (ii), add a seventh vertex labeled 30. Example (ii) shows that induced "minimal nonisometrically embeddable" subgraphs should *not* be used to prune the search for embeddable ones. But we still can exclude graphs with an induced MNH. Totals for $1 \leq n \leq 9$ are (1/1, 1/1, 2/2, 4/5, 9/11, 28/35, 86/111, 318/427, 1265/1742), where the denominators show *every* isometric embedding and the numerators show only the ternary ones.

159. (a) $\nu\bigl((b \oplus b')\,\&\,{\sim}(a \mid a')\bigr)$, the number of non-$*$ bits that differ.

(b) There are essentially only two other possibilities:

$$l(0) = 0000,\ l(1) = 1000,\ l(2) = 110{*},\ l(3) = {*}{*}11,\ l(4) = 0001;$$
$$l(0) = 000{*},\ l(1) = 100{*},\ l(2) = 1{*}10,\ l(3) = {*}111,\ l(4) = 010{*}.$$

(c) Let 5 be the top vertex, and let 6 and 7 be the two vertices inside the induced five-cycle. Use the labels $l(5) = 1{*}01$, $l(6) = 01{*}0$, $l(7) = {*}010$.

(d) If $v \neq r$, let $v' = \text{parent}(v)$. We want to prove that $\nu\bigl(l(u) \oplus l(v)\bigr) = d(u, v)$ for all u and v. If w is their nearest common ancestor, coordinates $(u, u', \ldots, u^{(d(u,w)-1)}, v^{(d(w,v)-1)}, \ldots, v', v)$ of $l(u)$ and $l(v)$ are respectively $(1, 1, \ldots, 1, 0, \ldots, 0, 0)$ and $(0, 0, \ldots, 0, 1, \ldots, 1, 1)$; other coordinates match. So there are $d(u, v)$ mismatches. (This construction is a special case of median labels; see 7.1.1–(63).)

(e) For example, suppose $d(u, w) = 4$ and $d(w, v) = 2$. Coordinates (u, u', u'', u''') are 1 in $l(u)$, non-1 in $l(v)$; coordinates (v', v) are non-1 in $l(u)$, 1 in $l(v)$; other coordinates are either both 1 or both non-1, so they contribute nothing to the "distance."

$\nu(x)$
sideways addition
nearest common ancestor
median labels

squashed cube
isometric embedding
Graham
Pollak
Winkler
3-PARTITION
strongly NP-complete
Garey
Johnson

Notice that coordinates (v', v) contribute $\frac{1}{2}(1+d(u,v)-d(u,v'))+\frac{1}{2}(1+d(u,v')-d(u,w)) = \frac{1}{2}(d(w,v)+d(u,v)-d(u,w))$. Similarly, coordinates (u, u', u'', u''') contribute $\frac{1}{2}(d(u,w)+d(u,v)-d(w,v))$. So the total "distance" is indeed $d(u,v)$.

For the Petersen graph, with vertices ij for $0 \le i < j < 5$ and root 01, we have

$$\begin{array}{rccccccccc} & 23 & 04 & 14 & 24 & 03 & 13 & 34 & 02 & 12\\ l(01) = & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ l(23) = & 1 & 0 & 0 & 0 & ? & ? & 0 & ? & ?\\ l(04) = & 1 & 1 & 0 & ? & ? & * & ? & ? & *\\ l(14) = & 1 & 0 & 1 & ? & * & ? & ? & * & ?\\ l(24) = & 0 & ? & ? & 1 & 0 & 0 & 0 & ? & ?\\ l(03) = & ? & ? & * & 1 & 1 & 0 & ? & ? & *\\ l(13) = & ? & * & ? & 1 & 0 & 1 & ? & * & ?\\ l(34) = & 0 & ? & ? & 0 & ? & ? & 1 & 0 & 0\\ l(02) = & ? & ? & * & ? & ? & * & 1 & 1 & 0\\ l(12) = & ? & * & ? & ? & * & ? & 1 & 0 & 1 \end{array} \implies \begin{array}{rccccccccc} & 23 & 04 & 14 & 24 & 03 & 13 & 34 & 02 & 12\\ l(01) = & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ l(23) = & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ l(04) = & 1 & 1 & 0 & 0 & * & * & 0 & * & *\\ l(14) = & 1 & 0 & 1 & 0 & * & * & 0 & * & *\\ l(24) = & 0 & * & * & 1 & 0 & 0 & 0 & 0 & 0\\ l(03) = & * & 0 & * & 1 & 1 & 0 & 0 & * & *\\ l(13) = & * & * & 0 & 1 & 0 & 1 & 0 & * & *\\ l(34) = & 0 & * & * & 0 & * & * & 1 & 0 & 0\\ l(02) = & * & 0 & * & * & 0 & * & 1 & 1 & 0\\ l(12) = & * & * & 0 & * & * & 0 & 1 & 0 & 1 \end{array}.$$

(f) Change '?' to ('$*$', '0') in u_v when $[u<v]+f(u,v)$ is (even, odd), where '$<$' is preorder and $f(u,v) = d(u,v)+d(u,r)+d(v,r)$. *Proof:* Let $p = d(u,w)$ and $q = d(w,v)$, and assume that $u < v$. Then the ancestors of u satisfy $u^{(k)} < v$ for $0 \le k < p$; similarly, $u < v^{(k)}$ for $0 \le k < q$. Define $x_k = f(u^{(k)}, v) \bmod 2$ for $0 \le k \le p$, and $x_{p+q-k} = f(u, v^{(k)}) \bmod 2$ for $0 \le k \le q$. Notice that $x_p = 0$, and $x_0 = x_{p+q}$. In $l(u)$ and $l(v)$ we have $u_{u^{(k)}} = 1$ and $v_{u^{(k)}} = {?}$ if and only if $x_k \ne x_{k+1}$, for $0 \le k < p$; similarly $u_{v^{(k)}} = {?}$ and $v_{v^{(k)}} = 1$ if and only if $x_{p+q-k} \ne x_{p+q-k-1}$, for $0 \le k < q$. So the number of ?s is the number of substrings '01' and '10' within x, say $2m$. If there are m' transitions '10' before the 0 at x_p, there are $m - m'$ transitions '01' after it.

Notes: If we shrink each subcube to a point, we get a "squashed cube." The subcube labels define an isometric embedding into a squashed cube — we can't get shorter paths by going outside the image and coming back again. (However, the computation of shortest distances between *unused* points of the squashed cube isn't easy.) The existence of a subcube representation with $n-1$ coordinates was conjectured by R. L. Graham and H. O. Pollak [*Bell System Tech. J.* **50** (1971), 2495–2519] and proved by P. M. Winkler [*Combinatorica* **3** (1983), 135–139].

160. False. (Maybe $G_1 = G_2 = K_2$, $H_1 = C_4$, $H_2 = K_1$.)

161. Because $G \sqsubseteq G'$ implies $G \subseteq G'$, (i), (iii), (v), and (vii) are obviously true. And (viii) clearly holds. But (ii), (iv), (vi) fail either when $G = G'$ or when $G' = G''$.

162. True. Suppose f embeds G into H, $u \not\!\!-\!\!\!- v$, $f(u) \mathrel{-\!\!-} f(v)$, and $u = u_0 \mathrel{-\!\!-} u_1 \mathrel{-\!\!-} \cdots \mathrel{-\!\!-} u_k = v$. Then $k > 1$, and $f(u_0) \mathrel{-\!\!-} f(u_1) \mathrel{-\!\!-} \cdots \mathrel{-\!\!-} f(u_k) \mathrel{-\!\!-} f(u)$ is a cycle.

163. The vertex of degree m must map to r; its neighbors must map to $\{x_{10}, \ldots, x_{m0}\}$. So each path P_{a_i} must be mapped to a_i vertices of $\{x_{j1}, \ldots, x_{jn}\}$ for some j. Those with the same j form a submultiset of sum $\le n$. So we get a suitable partition.

Conversely, such a partition yields an embedding. (And if $a_1 + \cdots + a_t = mn$ and $t = 3m$, we've solved the 3-PARTITION problem, which is strongly NP-complete. See M. R. Garey and D. S. Johnson, *Computers and Intractability* (1979), §4.2.2.)

164. (a) $5n$ vertices and $6n + \binom{n}{2}$ edges.

(b) If $v \mapsto f(v)$ is a strict embedding from G to H, then $(v,k) \mapsto (f(v),k)$ is easily seen to be an embedding from $q(G)$ to $q(H)$, by considering the three kinds of edges.

Conversely, assume that $(v,k) \mapsto \bigl(f(v,k), g(v,k)\bigr)$ is an embedding. For fixed v, let $w_k = f(v,k)$ and $r_k = g(v,k)$. If $w_k \ne w_{k+1}$ we must have $r_k = 0$ or $r_{k+1} = 0$. And if, say, r_0 is the only 0, we have $w_0 \ne w_1 = \cdots = w_4$ and $\{r_1, r_4\} = \{1, 3\}$, implying both $w_0 \mathrel{-\!\!-} w_1$ and $w_0 \not\!\!-\!\!\!- w_1$. A similar contradiction arises if r_k is the only 0. So

$(r_0,\ldots,r_4)$ must be a cyclic permutation of $(0,1,0,3,4)$ or $(0,1,0,4,3)$; but none of those is compatible with $(w_2,r_2) \mathrel{—} (w_4,r_4)$. Hence $f(v,k)=f(v)$ is independent of k. Now the image of $q(G)$ has $5n$ vertices and $6n+\binom{n}{2}$ edges; it must be an isomorphic copy.

165. If G has n vertices V, let $s(G)$ have n^2 vertices (v,w), where $(v,v) \mathrel{—} (v,w) \mathrel{—} (w,w)$ for all $v \neq w$, and $(v,w) \mathrel{—} (w,v)$ when $v \mathrel{—} w$. Let $t(G)$ be $s(G)$ together with additional vertices $\{v,w\}$ whenever $v \mathrel{—} w$; we have $(v,v) \mathrel{—} \{v,w\} \mathrel{—} (w,w)$ when $\{v,w\}$ exists. One can now prove that $G \subseteq H$ if and only if $s(G) \sqsubseteq t(H)$.

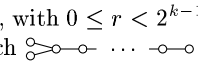
For example, if f is a strict embedding of $s(G)$, $f(v,v)$ must be a vertex of the form $(f(v),f(v))$, at least when $n>2$, because (v,v) has degree $2n-2$ in $s(G)$ and the other vertices of $t(H)$ have degree ≤ 3. Then $f(v,w)$ and $f(w,v)$ when $v \mathrel{—} w$ in G must be $(f(v),f(w))$ and $(f(w),f(v))$, since those are the only adjacent vertices in $t(H)$ that are neighbors of both $(f(v),f(v))$ and $(f(w),f(w))$. But when $v \neq w$ and $v \not\mathrel{—} w$, $\{f(v,w),\ f(w,v)\}$ can be any two of $(f(v),f(w))$, $(f(w),f(v))$, and possibly $\{f(v),f(w)\}$.

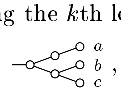
(Christine Solnon noticed that $s(G)$ has a huge number of automorphisms, because one can independently swap (v,w) with (w,v) when $v \neq w$. To avoid this problem she uses *directed* arcs $(v,v) \longrightarrow (v,w) \longrightarrow (w,w)$.)

166. (a) Suppose the given ISIP has edge labels L_j for $0 \leq j < J$. Define a labeled SIP on $\widehat{G}$ and $\widehat{H}$, the complete graphs on the vertices of G and H, giving their vertices the labels l_i and compatibilities they have in G and H. Also give their edges the existing labels L_j on existing edges, with the existing compatibilities; and let $L_j(u,v)=\Lambda$ when $u \not\mathrel{—} v$, where Λ is always compatible with Λ. Finally — and this is the key point — introduce a new edge label L_J, where $L_J(v,w)=[v \mathrel{—} w]$, compatible if and only if equal.

(b) Suppose the given SIP has labels l_i for $0 \leq i < I$ and L_j for $0 \leq j < J$. Introduce a new vertex label l_I, where $l_I(v)=[v \mathrel{—} u]$ for $v \in G \setminus u$ and $l_I(\hat{v})=[\hat{v} \mathrel{—} \hat{u}]$ for $\hat{v} \in H \setminus \hat{u}$; these labels are compatible if and only if $l_I(v) \leq l_I(\hat{v})$. Also introduce new vertex labels l_{I+1+j} for $0 \leq j < J$, where $l_{I+1+j}(v)=L_j(u,v)$ if $u \mathrel{—} v$, otherwise $l_{I+1+j}(v)=\Lambda$, using the compatibility relation of L_j and letting Λ be self-compatible.

(For directed graphs, however, we need more. Arc labels $L_j(v,w)$ are given when $v \longrightarrow w$. In part (a) let $L_J(v,w)=2[v \longrightarrow w]+[v \longleftarrow w]$. In part (b), let $l_I(v)=[v \longrightarrow u]$, $l_{I+1}(v)=[v \longleftarrow u]$, $l_{I+2+j}(v)=L_j(v,u)$ or Λ, $l_{I+2+J+j}(v)=L_j(u,v)$ or Λ.)

167. Given a 3SAT problem with m clauses, where every literal occurs exactly twice (exercise 7.2.2.2–208), construct G and H as follows: Start with the complete binary tree B_m with m leaves; if $m=2^k-r$, with $0 \leq r < 2^{k-1}$, there are r leaves on level $k-1$ and $m-r$ leaves on level k. Attach , a path of length $10m$ together with a 'Y' at one end, to the root of B_m, and call the result B_m^+. Then G is obtained from B_m^+ by replacing each leaf $-\!\circ$ by a path $-\!\circ\!-\!\circ\!-\!\circ$. Similarly, H is obtained from B_m^+ by replacing the kth leaf $-\!\circ$ by the graph

 where $\{a,b,c\}$ are the literals of clause k;

we also add nontree edges, two from each of a, b, c, to the vertices called respectively $\bar{a}$, $\bar{b}$, $\bar{c}$ in the other clauses. (These labels define the nontree edges, but don't appear in H.) Notice that G has $14m+1$ vertices, $14m$ edges; H has $17m+1$ vertices, $20m$ edges.

If the clauses are satisfiable, then $G \sqsubseteq H$, because we can match the "tip" of leaf k to a literal a, b, or c that satisfies clause k. Conversely, if $G \sqsubseteq H$, the 'Y' of G must correspond to the 'Y' of H, because the path of length $10m$ can't originate within B_m. Also the embedding of levels 0 through k must properly match up the r leaves on level $k-1$ and the $m-r$ leaves on level k. Thus the embedding will specify literals that satisfy each clause, never choosing both l and $\bar{l}$.

Solnon
automorphisms
complete graphs
directed graphs
complete binary tree

[This construction is based on an idea of C. Papadimitriou. On the other hand, E. Luks [*J. Computer and System Sciences* **25** (1982), 42–65] gave a polynomial-time algorithm to test *full* isomorphism between graphs of bounded degree. J. Matoušek and R. Thomas [*Discrete Math.* **108** (1992), 343–364] have shown how to solve $G \subseteq H$ and $G \sqsubseteq H$ in polynomial time if G has bounded degree and H has bounded treewidth.]

168. (a, b) Both equivalences are easily proved. Notice that all vertices of $\widehat{G}$ either have in-degree 0 (the original vertices of G) or in-degree 2 (the original edges of G); the embeddings must distinguish them too. (See T. Werth, M. Wörlein, A. Dreweke, I. Fischer, and M. Philippsen, in *Data Mining for Business Applications* (2009), 213.)

169. No. If $M = 2k+1$, use breadth-first searches to test if H contains a vertex u and two vertices $v \relbar\!\!\relbar w$ at distance k from u. A similar method works when $M = 2k+2$.

170. Yes, by including additional items in the option for v and V, namely

$$\{e \cdot E \mid e = (u \not\relbar v) \text{ and } E = (U \relbar\!\!\relbar V) \text{ for some } u \text{ and } U\}.$$

171. For SIP, there's a secondary item $uv \cdot UV$ for every arc $u \longrightarrow v$ in the pattern and every nonarc $U \not\longrightarrow V$ in the target; this item is inserted into the option for '$u\ U$' and the option for '$v\ V$' (and no other options). For ISIP, those options also get a secondary item $uv \cdot UV$ for every nonarc $u \not\longrightarrow v$ in the pattern and every arc $U \longrightarrow V$ in the target.

172. (a) If there are W_{mn} strict embeddings from $\mathcal{G}_m$ to $\mathcal{G}_n$, then $\mathrm{E}(W_{mn}) = n^{\underline{m}}/2^{\binom{m}{2}}$, because each of the $n^{\underline{m}}$ embedding functions f succeeds with probability $1/2^{\binom{m}{2}}$. When $m = 2\lg n + 1 + \delta$ we have $\binom{m}{2} \geq \binom{2\lg n}{2} + 2(1+\delta)\lg n = (m+\delta)\lg n$. Hence, by the first moment principle (MPR–(21)), $\Pr(\mathcal{G}_m \sqsubseteq \mathcal{G}_n) = \Pr(W_{mn} > 0) \leq \mathrm{E}(W_{mn}) \leq n^{-\delta}$.

(b) Clearly $\mathrm{E}(W_{mn}) \geq n^m 2^{-\binom{m}{2}}(1 - m^2/n)$; and when $m = 2\lg n + 1 - \delta$, one can show that $\mathrm{E}(W_{mn}^2) \leq \bigl(n^m/2^{\binom{m}{2}}\bigr)^2\bigl(1 + O(n^{-2\delta/3})\bigr)$. Hence, by the second moment principle, $\Pr(\mathcal{G}_m \sqsubseteq \mathcal{G}_n) \geq 1 - O(n^{-2\delta/3})$. [*J. Combin. Theory* **B160** (2023), 144–162.]

173. In general, assume that G and H are connected graphs with $G \subseteq H$, and that H can be disconnected into components H_1 and H_2 by cutting k edges. Then there must be a way to cut k edges from G in such a way that each resulting component can be embedded in either H_1 or H_2. (But (52) remains connected when any two edges are cut.)

174. `BRAIN83`(600) suffices for this, with $0 \mapsto 53$, $0{+} \mapsto 56$, $1{-} \mapsto 15$, $1 \mapsto 36$, $1{+} \mapsto 38$, $2{-} \mapsto 79$, $2 \mapsto 76$, $2{+} \mapsto 55$, $3{-} \mapsto 35$, $3 \mapsto 39$, $3{+} \mapsto 14$, $0{-} \mapsto 77$.

175. Yes: $12 \cdot 3$ ways in `BRAIN83`(370), found in 3.7 Tμ (but none in `BRAIN83`(360)).

176. (J_5 is 3-regular.) Not into `BRAIN83`(600); but $20 \cdot 86$ ways into `BRAIN83`(700).

177. Require $f(0{+}) < f(v)$ for $v \in \{1{-}, 1{+}, 2{-}, 2{+}, 3{-}, 3{+}, 0{-}\}$. ("Pairwise ordering," exercise 7.2.2.1–20, makes options still longer but needs only $3 + 1.5$ Gμ to find the 9 essentially different embeddings into `BRAIN83`(300).)

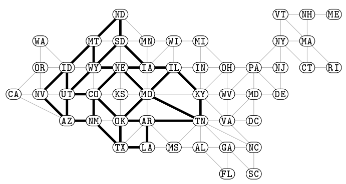
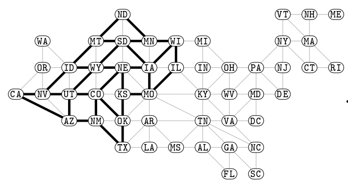
178. (a) The same result holds if 'deg' is replaced by 'd' in the definition of s, where d is *any* supplemental labeling function. *Proof:* Let v's neighbors in G be $v_1, \ldots, v_p$, where $d(v_1) \geq \cdots \geq d(v_p)$; similarly, let $f(v)$'s neighbors in H be $w_1, \ldots, w_q$, where $d(w_1) \geq \cdots \geq d(w_q)$ and $q \geq p$. Given $k \leq p$, there are indices $1 \leq i_1 < \cdots < i_k \leq q$, depending on k, such that $\{f(v_1), \ldots, f(v_k)\} = \{w_{i_1}, \ldots, w_{i_k}\}$. Let j be the index with $w_{i_k} = f(v_j)$; then $d(v_k) \leq d(v_j) \leq d(w_{i_k}) \leq d(w_k)$. [See S. Zampelli, Y. Deville, and C. Solnon, *Constraints* **15** (2010), 327–353.]

(b) (Solution by C. Solnon.) For $1 \leq k \leq p$, let v have a_k neighbors of degree k; also let w have b_k neighbors of degree k, or of degree $\geq k$ when $k = p$. Then check whether or not $b_p \ldots b_1$ majorizes $a_p \ldots a_1$, namely whether or not $b_p + \cdots + b_k \geq a_p + \cdots + a_k$ for $p \geq k \geq 1$. (Compare with Algorithm 5.2D and exercise 7.2.1.4–54.)

Papadimitriou
Luks
isomorphism between graphs
bounded degree
Matoušek
Thomas
treewidth
Werth
Wörlein
Dreweke
Fischer
Philippsen
breadth-first searches
first moment principle
second moment principle
cutting
Pairwise ordering
supplemental labeling function
Zampelli
Deville
Solnon
majorizes

[This partial ordering of multisets is a distributive lattice. When restricted to multisets of at most r positive integers, all $\le s$, it's the lattice $L(r,s)$ of partitions into at most r parts $\le s$, of which there are $[q^k]\binom{r+s}{r}_q$ partitions of k by 7.2.1.4–(51).]

179. `02` $\mapsto$ `MS` would force `01` $\mapsto$ `AL` and `03` $\mapsto$ `AL`. `02` $\mapsto$ `TX` would force `01` $\mapsto$ `NM` and `03` $\mapsto$ `NM`. Now `02` $\mapsto$ `LA` limits the domains of `01` and `03` to $\{$`MS`, `TX`$\}$; and that forces both `00` $\mapsto$ `NM` and `04` $\mapsto$ `NM`. (`AL` has no neighbors in `h`, so we can't map `00` $\mapsto$ `AL`.)

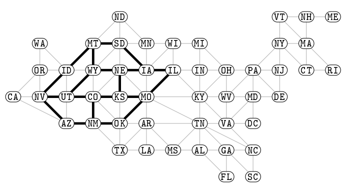
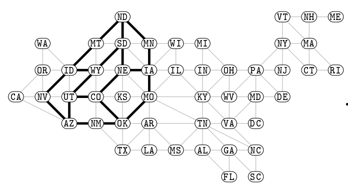
180.  and  .

181. (a) One of $4\cdot 12$ embeddings for $P_2 \,\square\, P_{12}$ is $\begin{pmatrix}\texttt{CA OR ID WY NE IA WI MI OH WV PA NJ}\\ \texttt{AZ NV UT CO KS MO IL IN KY VA MD DE}\end{pmatrix}$.

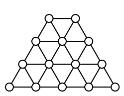
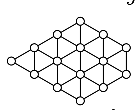
(b) And $\begin{pmatrix}\texttt{OR ID WY NE IA IL}\\ \texttt{NV UT CO KS MO KY}\\ \texttt{CA AZ NM OK AR TN}\end{pmatrix}$ is one of $4\cdot 9$ for $P_3 \,\square\, P_6$.

182. $P_2 \,\square\, P_2$ (in just $8\cdot 3$ ways, including $\{$`CO`, `NE`, `MO`, `OK`$\}$); $P_3 \,\square\, P_0$.

183. There are $10\cdot 7$ ways, including for instance

 and  .

(And there are $12\cdot 19$ ways to embed six pentagons that surround a *hexagon*.)

184. There are unique embeddings  $\sqsubseteq$ `USA` and  $\sqsubseteq$ `USA` of *simplex*$(4,4,4,3,0,0,0)$ and *simplex*$(5,5,3,3,0,0,0)$. (Put `NV` in the left corner.)

185. Let $M(v)$ be the mate of vertex v in the given matching, so that $M(x_i) = y_{j_i}$ and $M(y_{j_i}) = x_i$. Also let $M(y_j) = \perp$ if $j \notin \{j_1,\ldots,j_m\}$. Suppose there's also *another* feasible matching, with mate function m, in which $m(x_i) = y_j$ (hence j isn't removable).

Let $u_0 = x_i$, $v_0 = y_j$, and $u_1 = M(v_0)$. If $u_k \ne \perp$, let $v_k = m(u_k)$ and $u_{k+1} = M(v_k)$. If $u_k = u_0$, this sequence will be periodic, and $u_k \longrightarrow v_{k-1} \longrightarrow u_{k-1} \longrightarrow \cdots \longrightarrow v_0 \longrightarrow u_0$ will be a path in T; hence x_i and y_j will be in the same strong component.

But if $u_k = \perp$, let $v_{-1} = M(u_0)$ and $u_{-1} = m(v_{-1})$. If $u_{-l} \ne \perp$, let $v_{-l-1} = M(u_{-l})$ and $u_{-l-1} = m(v_{-l-1})$. Eventually we'll have $u_{-l} = \perp$, and a path $u_k \longrightarrow v_{k-1} \longrightarrow u_{k-1} \longrightarrow \cdots \longrightarrow v_{-l} \longrightarrow u_{-l}$; so y_j and $\perp$ will be in the same strong component.

Conversely, if there's an oriented path $x_i \longrightarrow \cdots \longrightarrow y_j \longrightarrow x_i$ or $\perp \longrightarrow \cdots \longrightarrow y_j \longrightarrow x_i$ in T, we can convert the given matching to a feasible matching with x_i — y_j by reversing each edge of that path. Hence j isn't removable.

186. (a) [This is Philip Hall's theorem, *J. London Math. Soc.* **10** (1935), 26–30, where Hall sets are featured. When x_1 — y_{j_1}, ..., x_m — y_{j_m} is such a matching, the sequence $j_1 \ldots j_m$ is called a "system of distinct representatives." Group theorists and algebraists use the term "Hall set" for quite a different concept — due jointly to Philip

partial ordering of multisets
distributive lattice
lattice
$L(r,s)$
partitions
q-nomial coeffs
Philip Hall
historical notes
system of distinct representatives
distinct representatives
Hall set

and Marshall Hall; see *Proc. Amer. Math. Soc.* **1** (1950), 575–581.] The condition is certainly necessary. If the algorithm fails, its final dag supplies an I with $|D(I)| < |I|$.

(b, c) If j is removable from x_i's domain, there's no matching in the subgraph with x_i and y_j deleted. So there's a subset $I \subseteq \{1,\ldots,m\} \setminus i$ with $|D'(I)| < |I|$, where D' is the subdomain in the subgraph. Thus $|D(I)| \le |I|$; by feasibility, $|D(I)| = |I|$.

(d) Because all values in $D(I)$ must be used as the images of I's variables.

(e) Let A, B, C be disjoint subsets of $\{1,\ldots,n\}$, with $a = |A|$, $b = |B|$, $c = |C|$, $a' = |D(A)|$, $b' = |D(B) \setminus D(A)|$, $c' = |D(C) \setminus D(A)|$, $a' + b' = a + b$, $a' + c' = a + c$. By feasibility we have $a' \ge a$ and $a' + b' + c' \ge |D(A \cup B \cup C)| \ge a + b + c$. Therefore $2a' + b' + c' \ge 2a + b + c = 2a' + b' + c'$, hence $a' = a$ and $|D(A \cup B \cup C)| = a + b + c$.

(f) This structure is a consequence of parts (b) and (d); I_1 through I_r are the minimal nonempty Hall sets. (Consequently the problem now has $r+1$ independent sets of variables $\{x_i \mid i \in I_j\}$, each of which has the all-different constraint only within its subdomain $D(I_j)$; moreover, perfect matchings are required, except between I_0 and $D(I_0)$.

(g) Each I_j is the set of x's belonging to some strong component, with $j = 0$ when that component also contains $\perp$. (Notice that I_0 might be $\emptyset$. There might be more than $r+1$ strong components, but only because $\{y_j\}$ is a singleton strong component when $D(i) = \{j\}$ is a singleton domain.)

Historical notes: Chapter 7 of C. Berge's book *Graphs and Hypergraphs* (1973) surveys the theory of alternating paths, which allows us to understand the family of all maximum matchings. Minimal nonempty Hall sets correspond to connected bipartite graphs for which every edge is part of a perfect matching. Such graphs are called "elementary bipartite" by L. Lovász and M. D. Plummer [*Matching Theory* (1986), Chapter 4], who have traced the concept back to D. Kőnig [*Mathematikai és Természettudományi Értesítő* **33** (1915), 221–229]. One of many interesting properties of such graphs, noted in their exercise 4.1.5, can be paraphrased as follows: "Let F be a loopfree digraph on vertices $\{x_1,\ldots,x_n\}$, and let G be the bigraph on $\{x_1,\ldots,x_n\}$, $\{y_1,\ldots,y_n\}$ whose edges are x_i — y_i for $1 \le i \le n$ and x_i — y_j whenever $x_i \longrightarrow y_j$ in F. Then F is strongly connected if and only if G is elementary."

J.-C. Régin [*Proc. Nat. Conf. on Artificial Intelligence* **12** (1994), 362–367] developed the algorithm of exercise 185 after discovering that every removable element of an all-different constraint can be identified from a single computation of strong components. Subsequent refinements of his algorithm were surveyed carefully and investigated empirically by I. P. Gent, I. Miguel, and P. Nightingale [*Artificial Intelligence* **172** (2008), 1973–2000], who noted gains in efficiency after strong components I_j have been identified as in (f) and used for GAD filtering on the smaller domains $D(I_j)$.

187. Given a matching, let T simply be the digraph on vertices $\{y_1,\ldots,y_n\}$ with arcs $\{y_{j_i} \longrightarrow y_k \mid x_i \longrightarrow y_k \text{ and } k \ne j_i\}$. Then $k \ne j_i$ is removable from D_i if and only if y_k and y_{j_i} belong to different strong components. (We're essentially identifying x_i with y_{j_i}.)

188. (a) The domain $D_{\rm a}$ of a must be a target vertex with a predecessor of out-degree ≥ 4; so $D_{\rm a} = \{6, 8, 13, 14, 15, 16\}$. And $D_{\rm d}$ is the set of targets with a predecessor having out-degree ≥ 1, in-degree ≥ 1, and at least two neighbors, namely $\{3, 6, 8, 12, 13, 14, 15, 16\}$. But $D_{\rm b} = \{3\}$ is the only target with a predecessor of bi-degree ≥ 1, where "bi-degree" is the number of two-way ($\leftrightarrow$) edges; $D_{\rm c} = \{8\}$ is the only target with bi-degree ≥ 1 and out-degree ≥ 2. Similarly, $D_{\rm e} = D_{\rm a}$; $D_{\rm f} = \{0\}$; $D_{\rm g} = \{14\}$.

After the forced assignments $\mathtt{b} \mapsto 3$, $\mathtt{c} \mapsto 8$, $\mathtt{f} \mapsto 0$, $\mathtt{g} \mapsto 14$, the remaining domains reduce to $D_{\rm a} = D_{\rm e} = \{6, 13, 15, 16\}$; $D_{\rm d} = D_{\rm a} \cup \{12\}$. LAD filtering now tells us that $\mathtt{e} \not\mapsto 16$, because $D_{\rm d}$ doesn't contain 16's successor (3). Similarly, $\mathtt{d} \not\mapsto 13$; $\mathtt{d} \not\mapsto 16$.

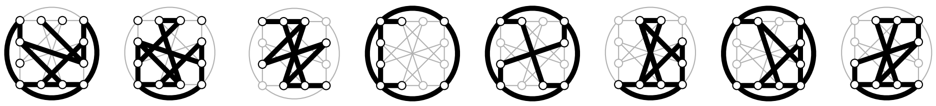
Marshall Hall
pigeonhole principle
perfect matchings
Historical notes
Berge
perfect matching
elementary bipartite
Lovász
Plummer
Kőnig
bigraph
strongly connected
Régin
Gent
Miguel
Nightingale
bi-degree

line graph
automorphisms
domain

So we branch on e, and there are three cases: If $\mathtt{e} \mapsto 6$, then $\mathtt{d} \mapsto 15$ and we discover two solutions, $\mathtt{a} \mapsto 13$ or 16. If $\mathtt{e} \mapsto 13$, then $\mathtt{d} \mapsto 12$ and $\mathtt{a} \mapsto 6$, 15, or 16. If $\mathtt{e} \mapsto 15$, then $\mathtt{d} \mapsto 6$ and $\mathtt{a} \mapsto 13$ or 16.

(b) With strict embedding the initial domain $D_\mathtt{e}$ is reduced to $\{8, 13, 16\}$. Only two of the previous solutions survive: $(\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e},\mathtt{f},\mathtt{g}) \mapsto (6 \text{ or } 15, 3, 8, 12, 13, 0, 14)$.

189. Yes; but there are three essentially different ways to delete two edges. If the edges are adjacent — at distance 1 in the line graph — there are $32{\cdot}4$ embeddings, such as (P1) below. If at distance 2, (P2) is one of $16 \cdot 7$ embeddings. At distance 3 there are none.



(P1) (P2) (i) (ii) (iv) (v) (vi) (vii)

190. Respectively $8 \cdot 1$, $32 \cdot 1$, 0, $16 \cdot 1$, $16 \cdot 3$, $64 \cdot 1$, $8 \cdot 1$ strict embeddings. (Notice that in case (iv), the pattern has 8 automorphisms, the target has 8, and the image has 4. So we get $(8 \cdot 8)/4 = 16$ different embedding functions f.)

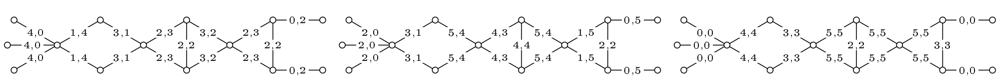
191. Spectacularly false. For example, if $H = G \,—\, K_1$ then $H^{\le 2}$ is a complete graph.

192. The degree of 12 in $G^{\le 2}$ is 11. So we can exclude 22 vertices whose degree in $H^{\le 2}$ is 10 or less: {AZ, CA, CT, DC, DE, FL, GA, LA, MA, ME, MI, MN, ND, NH, NJ, NV, OR, RI, SC, TX, VT, WA}. (The text's original method didn't exclude AZ or TX; its supplemental edge labels ℓ_G, ℓ_H did exclude all of these except MN and NV, and picked off also NM and WI.)

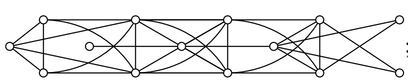
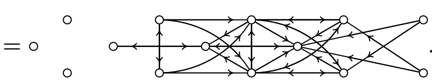
193. $d_G^{P_{k+1}}(v)$ is the number of simple paths of length exactly k that begin at v. (Thus when $k = 1$, $d_G^{P_2}(v) = \deg(v)$.) Consequently v's degree in $G^{\le 2}$ is $d_G^{P_2}(v) + [d_G^{P_3}(v) > 0]$.

194. Symmetrically equivalent vertices have the same label. Left to right, they are: (i) (4, 2, 7, 6, 8, 8, 5, 2); (ii) (0, 20, 2, 12, 6, 12, 6, 0); (iii) (2, 6, 6, 16, 12, 10, 7, 5); (iv) (0, 8, 7, 16, 12, 20, 8, 0); (v) (0, 0, 22, 16, 24, 34, 4, 0); (vi) (0, 0, 0, 2, 4, 4, 2, 2); (vii) (0, 2, 2, 4, 2, 2, 0, 0); (viii) (0, 0, 0, 0, 2, 0, 0, 0); (ix) (0, 0, 0, 2, 0, 2, 0, 0).

195. Here 'a, b' stands for the label left-to-right, then right-to-left:



196. Graph (i) is undirected, because s and t are symmetrically placed.

(i) =  ; (ii) =  .

197. Indeed, if $v \mapsto$ MO and v' is diagonally adjacent to v, we can't have $v' \mapsto$ AL, GA, LA, MN, NC, NM, OH, WV, or WY, even though those states are at distance 2 from MO, because no appropriate 4-cycle connects them to MO.

198. Only 15 vertices v of $H =$ USA have at least 4 neighbors in $H^{S,2}$, namely {AR, CO, IA, IL, KS, KY, MO, NE, NV, OK, SD, TN, UT, WV, WY}. Furthermore, if say $\mathtt{11} \mapsto$ NV, then $\mathtt{12} \mapsto$ AZ, CA, ID, OR, or UT; hence $\mathtt{12} \mapsto$ UT. Similarly $\mathtt{11} \mapsto$ NV implies $\mathtt{21} \mapsto$ UT, a contradiction. An analogous contradiction rules out $\mathtt{11} \mapsto$ WV.

199. One part has the neighbors u' of u in G (either $u \longrightarrow u'$ or $u \longleftarrow u'$ or both). The other part has the neighbors v' of v. There's a potential match between u' and v' if and only if all of the following conditions hold: (i) v' is in the current domain of u'. (ii) If $u \longrightarrow u'$ in G then $v \longrightarrow v'$ in H. (iii) If $u \longleftarrow u'$ in G then $v \longleftarrow v'$ in H. (iv) For each supplemental pair label that we've computed, satisfying (64), $\ell_G(u,u') \le \ell_H(v,v')$ and $\ell_G(u',u) \le \ell_H(v',v)$. And if G is to be *strictly* embedded into H, we also have two more conditions: (v) If $u \not\longrightarrow u'$ in G then $v \not\longrightarrow v'$ in H. (vi) If $u \not\longleftarrow u'$ in G then $v \not\longleftarrow v'$ in H.

Condition (i) implies that $d_G(u') \le d_H(v')$ for every supplemental label that we've computed, because we used those labels to initialize the domains.

This bipartite matching problem arises not only for the original pattern graph G and the original target graph H, but also (and independently) for every pair of supplemental graphs G^Σ and H^Σ that we know are solutions to (65).

200. Count the number of *strict* embeddings $S \sqsubseteq G$ that map $v \mapsto s$ and possibly $w \mapsto s$, in a motif S with designated vertices s and possibly t. (In particular, when S is $\overline{K}_2$ on the vertices s and t, the complementary graph $G^\Sigma = \overline{G}$ is supplementary.)

201. (a) Choose a vertex p in each connected component, and use breadth-first search to list the elements $p^{(1)}p^{(2)}\ldots$ reachable from p in increasing order of distance, starting with p itself. Concatenate those lists. (Some choices are much better than others.)

(b) (This data structure is a special case of a sparse-set representation.) Maintain also the inverse permutation $u_1 \ldots u_n$ so that, if the target vertices are $\{1,\ldots,n\}$, we have $t_j = k$ if and only if $u_k = j$. Initially $t_j = u_j = j$ for $1 \le j \le n$. When assigning $f(p_{l+1}) = k$, first set $j \leftarrow u_k$, $l \leftarrow l+1$, $k' \leftarrow t_l$, $t_l \leftarrow k$, $t_j \leftarrow k'$, $u_k \leftarrow l$, $u_{k'} \leftarrow j$. Then for each neighbor k'' of k, set $j \leftarrow u_{k''}$ and, if $j > s_l$, set $s_l \leftarrow s_l+1$, $k' \leftarrow t_{s_l}$, $t_{s_l} \leftarrow k''$, $t_j \leftarrow k'$, $u_{k'} \leftarrow j$, $u_{k''} \leftarrow s_l$. Finally, if $s_l < r_l$, set $l \leftarrow l-1$. (That assignment to k cannot be part of a solution, so we must backtrack. No changes to the t and u arrays need to be made when backtracking.)

(c) Yes; this condition is weaker than LAD filtering. (Notice that $q = q_l$ is fixed and can be computed in advance; also a target vertex k is near if and only if $u_k \le s_l$.)

(d) Yes, in the ISIP (strict embedding); again $q = q'_l$ is fixed. But no, in the SIP.

[These heuristics are used by the SIP and ISIP solvers VF2 and VF3 to prune the backtrack tree. See V. Carletti, L. P. Cordella, P. Foggia, A. Saggese, C. Sansone, and M. Vento, *IEEE Trans.* **PAMI-26** (2004), 1367–1372; **PAMI-40** (2018), 804–818.]

202. At step j, H is a Hall set, based on domains different from D_j. [See C. McCreesh and P. Prosser, *LNCS* **9255** (2015), 300–301.]

203. First assume for convenience that the target graph has $n \le 64$ vertices, that all graphs are undirected, and that there are at most 7 supplemental graphs (thus at most 8 altogether). Represent the pattern by an $m \times m$ matrix A_{uv} of bytes; the individual bits of A_{uv} tell us which of the 8 pattern graphs have $u \,—\, v$. Each target graph H^S is represented by n octabytes $H^S_{v'}$; bit u' of $H^S_{v'}$ is 1 if and only if $u' \,—\, v'$ in H^S.

To assign $v \mapsto v'$, first set $D_v \leftarrow \{v'\}$, and mark it "final" so that it won't participate at deeper levels of the search. Then, for every pattern vertex $u \ne v$, we must set $D_u \leftarrow D_u \mathbin{\&} H^S_{v'}$ whenever A_{uv} tells us that $u \,—\, v$ in H^S; we simply set $D_u \leftarrow D_u \setminus \{v'\}$ if $A_{uv} = 0$. (For strict embedding, also set $D_u \leftarrow D_u \mathbin{\&} {\sim}H^S_{v'}$.)

The resulting domains should now be refined further as in exercise 202. That algorithm is readily extended to recognize quickly whether or not at least one nonfinal domain has been reduced to size 1; if so, we repeat the process with a new v and v'.

If the target graph has $n > 64$ vertices, a similar procedure can be carried out with $\lceil n/64 \rceil$ octabytes per domain and with $\lceil n/64 \rceil$ octabytes in place of each $H^S_{v'}$. If the graphs are directed, byte A_{uv} should represent $u \longrightarrow v$ in the pattern graphs, and bit u' of $H^S_{v'}$ should represent $u' \longrightarrow v'$ in H^S. The transposed target graphs should also be represented separately, so that bit u' of $H^{ST}_{v'}$ represents $v' \longrightarrow u'$ in H^S. If A_{vu} tells us that $v \longrightarrow u$ in H^S, we should set $D_u \leftarrow D_u \mathbin{\&} H^{ST}_{v'}$.

initialize the domains
complementary graph
breadth-first search
data structure
sparse-set representation
LAD filtering
VF2
VF3
Carletti
Cordella
Foggia
Saggese
Sansone
Vento
Hall set
McCreesh
Prosser
strict embedding

[*Historical notes:* Bitwise domain filtering was recommended by J. R. Ullmann in one of the first papers about SIP solving, *JACM* **23** (1976), 31–42. See also J. J. McGregor, *Information Sciences* **19** (1979), 229–250, as well as Ullmann's subsequent paper in *ACM J. Experimental Algorithmics* **15** (2011), 1.6:1–1.6:64. C. McCreesh has reported (unpublished) that the state-of-the-art Glasgow solver, c. 2020, spends roughly 1/3 of its time doing bitwise propagation, 1/4 doing relaxed GAD filtering, 1/6 copying domains from one level to the next, and 1/10 choosing the variable on which to branch.]

204. In the following code, D_k is the octabyte in address `dom` $+\,8k$. Sorting is achieved by making byte `START[`i`]` point to the first domain of size i; `NEXT[`k`]` points to the next domain of the same size. The assembler code '`start GREG @ ;next GREG @+64 ;dom GREG @+128`' appears somewhere in the `Data_Segment`, so that we can address those arrays conveniently. Bucket m receives all domains of size $\ge m$, because they can be treated in any order. Symbols `t`, `u`, `h`, `i`, `j`, `k`, `kk` denote registers \$255, \$0, \$1, \$2, \$3, \$4, \$5.

```
Sort SET   j,0          j ← 0.
     SET   i,56         i ← 56.
1H   STOU  j,start,i    START[i..i+7] ← 0.
     SUB   i,i,8        i ← i − 8.
     PBNN  i,1B         Repeat while i ≥ 0.
     CMP   t,i,0        t ← −1.
     STB   t,m,start    START[m] ← −1.
     SET   k,m          k ← m.
1H   8ADDU kk,k,0       kk ← 8k.
     LDOU  t,kk,dom     t ← D_k.
     SADD  t,t,0        t ← |D_k|.
     CMP   i,t,m
     CSP   t,i,m        If t > m set t ← m.
     LDB   i,start,t
     STB   i,next,k     NEXT[k] ← START[t].
     STB   k,start,t    START[t] ← k.
     SUB   k,k,1        k ← k − 1.
     PBP   k,1B         Loop while k > 0.
DoIt SET   u,0          u ← ∅.
     SET   h,0          h ← ∅.
     SET   i,0          i ← 0. (j = 0)
     LDB   k,start,0    k ← START[0].
     PBZ   k,2F         No domain empty?
1H   INCL  j,1          j ← j + 1.
     8ADDU kk,k,0
     LDOU  t,kk,dom     t ← D_k.
     OR    u,u,t        U ← U ∪ t.
     ANDN  t,t,h        t ← t \ H.
     BZ    t,Unfeas     To Unfeas if t = ∅.
     STOU  t,kk,dom     D_k ← t.
     SADD  t,u,0        t ← |U|.
     CMP   t,t,j        t ← sign(t − j).
     BN    t,Unfeas     To Unfeas if |U| < j.
     CSZ   h,t,u        If |U| = j set H ← U.
     LDB   k,k,next     k ← NEXT[k].
     BP    k,1B         Repeat loop if k > 0.
     BN    k,Feas       We're done if k < 0.
2H   INCL  i,1          i ← i + 1.
     LDB   k,i,start    k ← START[i].
     PBP   k,1B         Repeat loop if k > 0.
     PBZ   k,2B         Increase size if k = 0.  ▮
Feas ...
```

The total time is approximately $(8\mu + 30\upsilon)m + 10\mu + 38\upsilon$.

Complete GAD filtering can also be done with bitwise manipulation, but the algorithms are considerably more complicated and time-consuming. See P. Van Kessel and C.-G. Quimper, *Proceedings of the AAAI Conference* **26** (2012), 577–583.

205. (a) The problem is to find knight paths $p_1 \ldots p_m$ and $q_1 \ldots q_n$ so that the mn cells $p_i + q_j$ lie in a chessboard and are distinct. There are respectively (2, 13, 16, 3) essentially different solutions for $(m, n) = (2, 22)$, $(3, 12)$, $(4, 7)$, $(6, 6)$; examples appear in (i)–(vi) of Fig. A–14. The symmetrical constructions (iv) and (v) show that $P_{n-2} \mathbin{\square} P_{n-2} \subseteq N_n$ for all $n \ge 4$, indeed in at least two different ways when n is even. Case (vi) is delightfully "symmetrical" although it has no nontrivial automorphism: It arises from 64 different embedding functions f, while cases (iv) and (v) arise from only 16 each.

(b) Every extremal solution is shown in (vii)–(xiv) of Fig. A–14.

(c) Case (xv) is one of three essentially different solutions for $n = 20$.

(d) Case (xvi) is the essentially unique solution for $n = 8$.

(e) Case (xvii) is one of two essentially different solutions for $n = 8$.

(f) Case (xviii) is the essentially unique embedding for $n = 2$, and it's strict.

[Incidentally, the 4-cube $P_2 \mathbin{\square} P_2 \mathbin{\square} P_2 \mathbin{\square} P_2$, which is also $C_4 \mathbin{\square} C_4$, is *uniquely* embeddable in N_n for all $n \ge 7$, and that embedding is in fact strict.]

Historical notes
Bitwise domain filtering
Ullmann
McGregor
McCreesh
copying domains
OR
ANDN
Van Kessel
Quimper
geek art
automorphism
4-cube

backtrack search

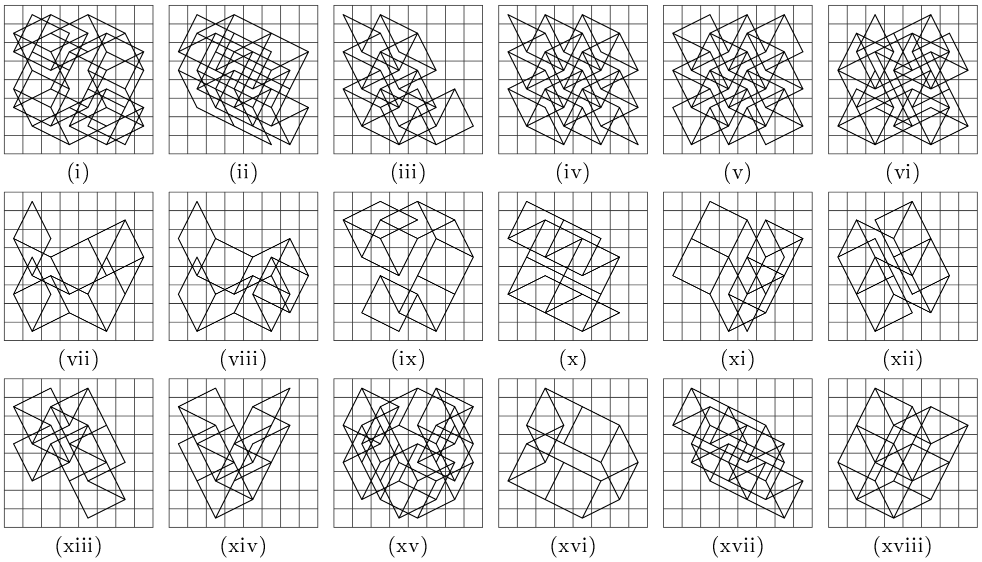


Fig. A–14. A gallery of knight's grids in a chessboard.

206. Although SIP solvers use sophisticated techniques like filtering and supplemental labels, the special geometry of these problems means that a specially tuned backtrack search can be significantly faster. For example, suppose t is given, as well as a fixed knight path $p_1 \ldots p_m$. Instead of mapping a pattern vertex into a fixed vertex of the target graph N_t, we can map q_1 to the origin and backtrack over all knight paths $q_1 q_2 \ldots$ for which the points $p_i + q_j$ are distinct and fit into a $t \times t$ region of the plane. That avoids $\Theta(t^2)$ near-similar branches at the top levels of the search tree.

We have $\bigl(f_2(3), \ldots, f_2(11)\bigr) = (1, 2, 7, 10, 15, 22, 29, 36, 46)$; and $f_2(12) \ge 57$, because of the knight path $q_1 \ldots q_{57}$ in Fig. A–15. Using somewhat similar paths one can prove that $f_2(t) = t^2/2 - O(t)$, with most of the cells q_j on "even" rows.

When $m = 3$ we can compute exact results a bit further: $\bigl(f_3(3), \ldots, f_3(14)\bigr) = (1, 1, 3, 5, 9, 12, 16, 20, 27, 33, 39, 48)$; and $f_3(15) \ge 55$ because of a knight path $q_1 \ldots q_{55}$ that sticks to cells (i, j) with $(i + j) \bmod 3$ fixed. Using such paths together with a "crooked path" $p_1 p_2 p_3$ one can show that $f_3(t) = t^2/3 - O(t)$. However, $f_3(14) = 48$ is obtained with a "straight" $p_1 p_2 p_3$ and a completely mysterious path $q_1 \ldots q_{48}$.

When $m = 4$ we have $\bigl(f_4(3), \ldots, f_4(17)\bigr) = (1, 1, 2, 4, 5, 7, 10, 15, 18, 22, 25, 34, 37, 43, 52)$, and $f_4(18) \ge 61$. In this case the optimum solutions for $13 \le n \le 16$ all occur when $p_1 p_2 p_3 p_4$ is the zigzag path shown as '**1** 1 1 1'; such solutions prove that $f_4(t) = t^2/4 - O(t)$. However, the zigzag path yields only $f_4(17) \ge 49$. Hence the straight path wins when $t = 17$, and the sequence $f_4(t)$ remains mysterious.

Turning now to induced subgraphs, $\bigl(\overline{f}_2(3), \ldots, \overline{f}_2(18)\bigr) = (1, 2, 5, 8, 8, 10, 12, 15, 19, 24, 28, 32, 36, 40, 46, 52)$; also $\bigl(\overline{f}_3(3), \ldots, \overline{f}_3(24)\bigr) = (1, 1, 3, 4, 5, 6, 7, 10, 11, 12, 14, 16, 20, 21, 25, 28, 32, 34, 41, 44, 49, 53)$; furthermore $\bigl(\overline{f}_4(3), \ldots, \overline{f}_4(36)\bigr) = (1, 1, 2, 4, 4, 5, 6, 8, 8, 10, 12, 12, 14, 15, 17, 18, 20, 20, 22, 24, 25, 26, 28, 29, 31, 32, 34, 35, 37, 38, 40, 41, 43, 44)$. It appears that $\lim_{t\to\infty} \overline{f}_2(t)/t^2 = \alpha_2$ and $\lim_{t\to\infty} \overline{f}_3(t)/t^2 = \alpha_3$ for some (unknown) positive constants α_2 and α_3. But $\overline{f}_4(t) = O(t)$, because none of the paths $p_1 p_2 p_3 p_4$ allow us to "turn a corner."

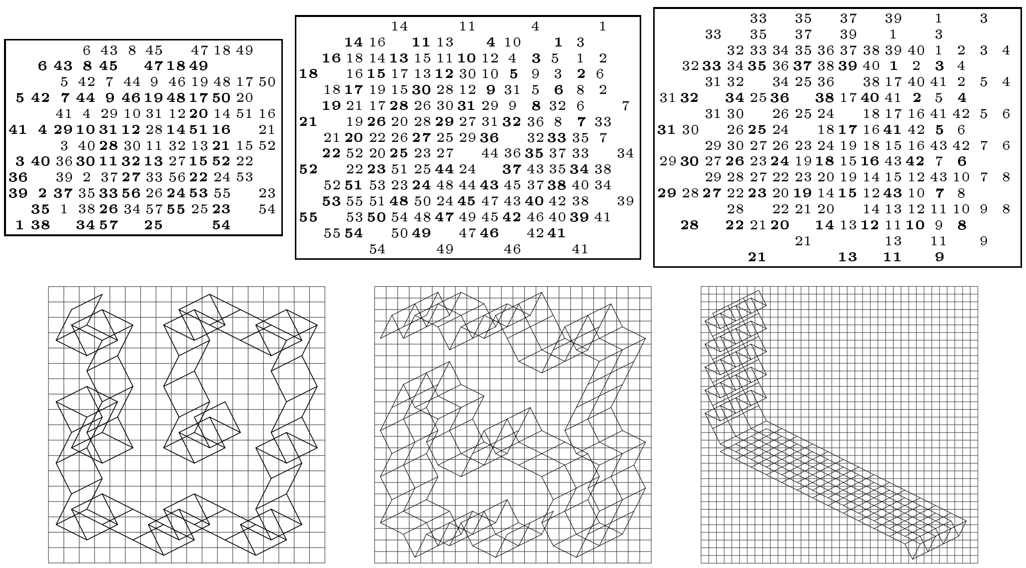


Fig. A–15. Champion knight's grids on larger boards.

McCreesh
Glasgow solver
restarts
symmetrical solutions
Solnon
Rokicki
horizontal and vertical symmetry
axial symmetry
4-fold symmetry
90-degree rotation
central symmetry
Beluhov
Lo Shu
magic square
Dürer
axial symmetry

207. C. McCreesh found the first solution below in 2019, in 446 seconds using 160 parallel threads, on the Glasgow solver (which incorporates random restarts). It seems probable that millions of solutions exist, but a person has to be lucky to find them.

The problem is essentially to label each cell ij of a chessboard with the name of another cell xy, so that when two cells are a knight move apart their labels are a queen move apart. (For example, the knight-move neighbors of the cell labeled 36 in the first solution are labeled 06, 63, 47, and 66.) In problems such as this it's often easier (and fun) to look for *symmetrical solutions*, because such solutions have many fewer variables. For example, we can impose further constraints: (i) if $ij \mapsto xy$ then $ji \mapsto yx$; (ii) if $ij \mapsto xy$ then $i\bar{j} \mapsto x\bar{y}$, where $\bar{y} = 7 - y$; (iii) if $ij \mapsto xy$ then $\bar{i}j \mapsto \bar{x}y$. C. Solnon discovered in 2021 that condition (i) cannot be satisfied. But T. Rokicki found that there are exactly $8 \cdot 4$ ways to satisfy both (ii) and (iii), as in the second solution below, thus achieving "axial symmetry" (see exercise 7.2.2.1–386). He showed furthermore that exactly $8 \cdot 14$ solutions have the other kind of 4-fold symmetry, under 90-degree rotation, as in the third solution; the constraint in this case is (iv) if $ij \mapsto xy$ then $j\bar{i} \mapsto y\bar{x}$. Also exactly $4 \cdot 23$ solutions, like the fourth, satisfy (ii) but not (iii). And $32 \cdot 991$ have central symmetry: (v) if $ij \mapsto xy$ then $\bar{i}\bar{j} \mapsto \bar{x}\bar{y}$, but not (ii) or (iii).

60 30 36 44 74 15 64 14	12 11 06 05 02 01 16 15	67 50 32 41 30 07 34 71	10 40 71 61 66 76 47 17
06 41 65 33 66 17 75 24	51 00 13 41 46 14 07 56	33 63 60 57 12 74 31 02	11 72 60 43 44 67 75 16
10 63 00 47 35 34 11 67	10 42 55 04 03 52 45 17	00 37 23 52 61 35 01 24	62 21 41 77 70 46 26 65
05 56 32 61 77 57 13 31	50 24 30 43 44 37 23 57	73 51 66 15 56 22 72 13	63 00 73 24 23 74 07 64
62 70 27 55 43 71 37 02	20 54 40 33 34 47 53 27	64 05 55 21 62 11 26 04	51 42 27 30 37 20 45 56
45 50 52 76 07 22 53 73	60 32 25 74 73 22 35 67	53 76 42 16 25 54 40 77	05 03 33 53 54 34 04 02
72 26 25 23 51 46 01 04	21 70 63 31 36 64 77 26	75 46 03 65 20 17 14 44	22 57 32 36 31 35 50 25
20 12 54 16 21 03 40 42	62 61 76 75 72 71 66 65	06 43 70 47 36 45 27 10	12 06 13 55 52 14 01 15

208. N. Beluhov notes that the 3×3 "Lo Shu" magic square may actually be regarded as an embedding of N_3 into $P_3 \,\square\, P_3$; and the famous magic square in Albrecht Dürer's *Melencolia I* is an embedding of N_4 into Q_4, with axial symmetry!

Without reducing for symmetry, there are 44176 embeddings for $n = 3$, 171569126 for $n = 4$, and zillions for $5 \le n \le 7$. Restricting to solutions with central symmetry, these counts become (80, 66624, 69200, 1599680, 48560, 32000), for $3 \le n \le 8$.

Surprisingly, no 4-way symmetry is possible for $9 \le n \le 30$. In fact Rokicki found that there are only $32 \cdot 2$ symmetrical solutions for $n = 9$, all with central symmetry. [Solnon has discovered that *unsymmetrical* solutions can be obtained quite quickly, with a dynamically weighted improvement of the MRV heuristic, at least for $9 \le n \le 12$!]

209. (Solution by N. Beluhov.) Of course N_n has very few edges when n is small, so the task is easy; (1, 24, 1296, 69120) embeddings solve the problem when $n = (1, 2, 3, 4)$.

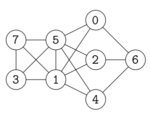
When $n = 5$, there are exactly 28800 embeddings. In fact, they are the mappings $ij \mapsto p((2i+j) \bmod 5)q((3i+j) \bmod 5)$ and their transposes, when p and q are arbitrary permutations of $\{0, 1, 2, 3, 4\}$. Those maps also embed the toroidal 5×5 knight moves.

But it's impossible when $n > 5$, because knight edges of the same slope must map onto rook edges of the same slope. (This is true in each "knight rhombus," and we can connect moves of the same slope by chains of such rhombuses.) And without loss of generality, knight edges of at least two distinct slopes map onto horizontal rook edges.

(And in general, the $n \times n$ graph of every skew free (p, q)-leaper is embeddable in the $n \times n$ rook graph for $n = p^2 + q^2$, but not for larger n.)

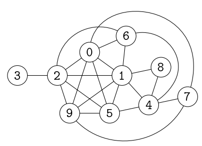
210. If solvable, there would be headline news: We could name 75 American collegiate football teams who played each other in 1990 if and only if 75 corresponding characters encountered each other in the first half of Victor Hugo's *Les Misérables* (1862)! But unfortunately this one is *not* solvable. Indeed, 95 of the target teams belong to one of eleven "conferences"; and they play almost everybody in their own conference. So the largest independent set among those teams has at most 1+1+1+1+1+1+1+1+1+2+2+2 members. Since at most 8 of the remaining 25 teams are independent, the target graph has at most 23 independent vertices. But the pattern graph has 27 *isolated* vertices.

211. (a) The unique solution is nicely symmetric. One interesting way to find it is to consider a Boolean function on $\binom{8}{2} = 28$ variables x_{uv}, one for each potential edge u — v. The function that characterizes 4-universal graphs H is $\bigwedge_{G \in \mathcal{G}_4} S(G)$, where $\mathcal{G}_4$ is the set of all 4-vertex graphs and $S(G) = [G \sqsubseteq H]$. For example, when $G = L(3, 1)$ we have $S(\circ\!\!-\!\!\circ\!\!<) = \bigvee_{tuvw} x_{tu}x_{tv}x_{tw}x_{uv}\bar{x}_{uw}\bar{x}_{vw}$, which is an OR taken over all $8 \cdot 7 \cdot 6 \cdot 5 = 1680$ ordered quadruples of vertices $tuvw$.



Many simplifications are possible, because H must contain a 4-vertex clique C as well as an independent set I of size 4, having just one vertex in common with C. The eighth vertex must not be adjacent to all of $C \setminus I$, but adjacent to at least one of $I \setminus C$. That leaves only 11 unspecified variables x_{uv}; the resulting BDD has only 1019 nodes and can be computed in only 4 megamems.

(b) It turns out that exactly 90 distinct 4-universal 8-vertex graphs can be strictly embedded in a 5-universal 10-vertex graph — but *not* the graph of (a). This example becomes 4-universal when we delete vertices 8 and 9; further deletion of $\{5, 6, 7\}$ gives the bull.



The Boolean function for all 5-universal graphs in $\mathcal{G}_{10}$, analogous to the one in part (a), has $\binom{10}{2} - 22 = 23$ variables and a BDD of size 3803(!), computed in 2.5 Gμ.

[*Historical notes:* J. W. Moon introduced n-universal graphs in *Proc. Glasgow Math. Assoc.* **7** (1965), 32–33. He defined $\lambda(n)$ as the minimum number of vertices in such a graph, and showed that $2^{(n-1)/2} < \lambda(n) < 1.1n2^{(n-1)/2}$. N. Alon sharpened this to $\lambda(n) = 2^{(n-1)/2}(1 + O(n^{-1/2}(\log n)^{3/2}))$ in *Geometric and Functional Analysis* **27**

Rokicki
Solnon
MRV heuristic
Beluhov
(p, q)-leaper
leaper
football teams
Hugo
Les Misérables
independent set
isolated
unique solution
Boolean function
lollipop
paw
clique
BDD
Historical notes
Moon
Alon

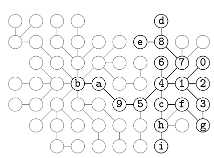
(2017), 1–32. Exact values for small n were computed by J. Trimble [arXiv:2109.00075 [math.CO] (2021), 22 pages], who found $(\lambda(1),\ldots,\lambda(6)) = (1, 3, 5, 8, 10, 14)$ and $16 \le \lambda(7) \le 18$. The minimum number of edges in an n-universal graph is (0, 1, 4, 11, 21) for $1 \le n \le 5$; the smallest known examples for $n = 6$ and 7, due respectively to F. Stappers and J. Trimble, have respectively 45 and 77 edges. T. Zhang and S. Szeider showed in *LIPIcs* **280** (2023), 39:1–39:20, that $\lambda(7) > 16$.]

212. In fact we can obtain each V_{j+1} by "promoting" a vertex of V_j: $V_1 = 0125$ $(K_{1,1,2})$; $V_2 = 0126$ (C_4); $V_3 = 0136$ (P_4); $V_4 = 0236$ $(P_3 \oplus K_1)$; $V_5 = 0237$ $(K_2 \oplus 2K_1)$; $V_6 = 0247$ $(\overline{K_4})$; $V_7 = 1247$ $(K_{1,3})$; $V_8 = 1347$ $(L(3,1))$; $V_9 = 1357$ (K_4); $V_{10} = 1367$ $(C_3 \oplus K_1)$; $V_{11} = 2367$ $(2K_2)$. *Exercise:* Make and post an animated video of this. [See suggestions by Filip Stappers (`https://archive.org/details/gray-4-universal/`) and Ho Boon Suan (`https://www.youtube.com/watch?v=KelZOGPr3Zw`).]

An interesting CSP now suggests itself: Given a digraph in which each vertex v has a given color $c(v) \in \{1,\ldots,d\}$, we seek an oriented path $v_1 \longrightarrow v_2 \longrightarrow \cdots \longrightarrow v_d$ such that each color occurs once in $\{c(v_1), c(v_2), \ldots, c(v_d)\}$. Let's call this the *rainbow path problem.* There's a nice way to formulate it as an XCC: Let there be $3d$ primary items x, $x+$, $x-$ for $1 \le x \le d$, together with a secondary item v for each vertex v; we also have two special primary items $\perp$ and $\top$. If the vertices colored x are $v_1, \ldots, v_t$, there are $3t$ options 'x $v_1{:}\delta_{1s}$ $\ldots$ $v_t{:}\delta_{ts}$', '$\perp$ $v_s{:}1$ $x-$', '$x+$ $v_s{:}1$ $\top$', for $1 \le s \le t$. Also, for each arc $v \longrightarrow v'$ with $c(v) \ne c(v')$, there's an option '$c(v)+$ $v{:}1$ $v'{:}1$ $c(v')-$'.

This exercise is the special case where each v is a 4-element subset of $\{0,\ldots,7\}$ and $c(v)$ is the corresponding induced subgraph; $v \longrightarrow v'$ if and only if v' increases an element of v by 1. The associated XCC has 105 items, 341 options, and 22 solutions, found in 3 megamems. (But we were lucky, because there are $8! = 40320$ ways to label the vertices of H and only 4224 of them yield solutions.)

213. (a) The answer is unique, except for permutation of $\{\mathtt{0},\mathtt{2},\mathtt{3}\}$:



(b) Yes. Subtree S_r has nodes $\{\mathtt{7},\mathtt{8},\mathtt{d},\mathtt{e}\}$; subtree T_e has nodes $\{\mathtt{w},\mathtt{x},\mathtt{y},\mathtt{z},\mathtt{A},\mathtt{B}\}$; map $\mathtt{7} \mapsto \mathtt{w}$, $\mathtt{8} \mapsto \mathtt{x}$, $\mathtt{d} \mapsto \mathtt{y}$, $\mathtt{e} \mapsto \mathtt{z}$. (This example uses an extended hexadecimal code in which the letters $[\mathtt{a}\,.\,.\,\mathtt{z}]$ denote $[10\,.\,.\,35]$ and the letters $[\mathtt{A}\,.\,.\,\mathtt{Z}]$ denote $[36\,.\,.\,61]$.)

(c) Let $e_j = {}^{\,v}_{w_j}$, for $1 \le j \le l$. The stated embedding is possible if and only if there are analogous embeddings of $S_{r_1}, \ldots, S_{r_k}$ into some k distinct subtrees T_{e_j}.

(d) The condition in (c) is that there's a matching of size k in the graph with k boys, l girls, and $b_i \longrightarrow g_j \iff \mathtt{sol}[r_i][e_j]$.

(e) Let v's neighbors be $\{w_0,\ldots,w_l\}$; define e_j as in (c), but for $0 \le j \le l$. Now consider the graph of (d), but with $l+1$ girls. The embedding for $e = {}^{u}_{v}$ is possible when $u = w_j \iff$ there's a matching of size k with g_j *unmatched.* And Algorithm 7.5.1H has the beautiful property that such a matching exists $\iff g_j \in \{\mathtt{QUEUE[0]},\ldots,\mathtt{QUEUE}[q-1]\}$ when that algorithm terminates with no free boys. (This brilliant idea saves us a factor of n. See Theorem 3.4 in Matula's paper, *Annals of Discrete Math.* **2** (1978), 91–106.)

(f) Assign integers $[0\,.\,.\,2n-2)$ to the arcs e of T so that (i) all arcs $e = {}^{u}_{v}$ with the same value of v are consecutive, and (ii) if $\deg(e) < \deg(e')$ then $e < e'$. (Here $\deg(e)$ means $\deg(v)$ when $e = {}^{u}_{v}$.) For $1 \le d \le n$, set $\mathtt{THRESH}[d]$ to the number of arcs with $\deg(e) < d$. If $e = {}^{u}_{v}$ and $e' = {}^{v}_{u}$, set $\mathtt{UERT}[e] \leftarrow u$, $\mathtt{VERT}[e] \leftarrow v$, $\mathtt{DUAL}[e] \leftarrow e'$.

The heart of the computation is *solve*(r), a recursive procedure to set $\mathtt{sol}[q][e]$ for all arcs e and all descendants q of r, where r is a node of S. Here's how it works:

Trimble
Stappers
Trimble
Zhang
Szeider
internet
video
Stappers
Ho Boon Suan
CSP
rainbow path problem
XCC
hexadecimal code, extended
unmatched girl in maximum matching
recursive procedure

author
downloadable programs

If r is a leaf, simply set `sol[`r`][`e`]` $\leftarrow 1$ for $0 \le e < 2n-2$. Otherwise suppose r_1, $\ldots$, r_k are r's children, and $solve(r_i)$ for $1 \le i \le k$. We start with `sol[`r`][`e`]` $\leftarrow 0$ for $0 \le e < 2n-2$. Then we set $d \leftarrow k+1$, $e \leftarrow$ `THRESH[`d`]`, and do the following while $e < 2n-2$: While $e =$ `THRESH[`$d+1$`]` set $d \leftarrow d+1$; set up a bipartite matching problem (see below), and use its solution to fix `sol[`r`][`$e+j$`]` for $0 \le j < d$; then set $e \leftarrow e+d$. (One can abort, concluding that $S \not\subseteq T$, if $solve(r)$ never sets any `sol[`r`][`e`]` $\leftarrow 1$.)

The bipartite graph for r and $[e \,..\, e+d)$ has k boys $b_i = r_i$ (the children of r) and $d > k$ girls $g_j = e+j$, with b_i — g_j if and only if `sol[`b_i`][DUAL[`g_j`]]` $= 1$. However, several special cases are important: If b_i — g_j for *no* j, there's no perfect matching and we don't bother to look for one. If b_i — g_j for *all* j, we omit boy b_i from the graph. (That happens often, for example whenever r_i is a leaf.) So we're left with $k' \le k$ boys, where k' is at most the inner degree of r. If $k' = 0$ (every boy matches every girl), we set `sol[`r`][`$e+j$`]` $\leftarrow 1$ for $0 \le j < d$. Otherwise if Algorithm 7.5.1H terminates with $f = d - k'$, and with q girls in its queue, we set `sol[`r`][QUEUE[`j`]]` $\leftarrow 1$ for $0 \le j < q$.

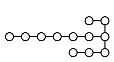
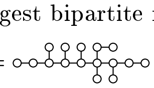
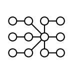
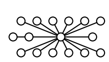
Finally, $S \subseteq T$ if and only if `sol[1][`e`]` $= 1$ for at least one arc e.

Here's the `sol` matrix for the trees of (a):

```
u 11796cclfhnuxxwwCCHOONpRpML1435686aclkmfifo5sjtCQuGFOtVpRJSqMrUqTO32sNuwzyGWP5Lr5b975ked5nhguBAxuHED5KJItFCwv4qpjfc6
v 028abdemgiovyzABDEQPWVISKUT33447799kkllhhnnjjssHHFFGGNNJJRRrrMMLL111tttxxxOOOqqq6666ccccffffwwwwCCCCppppuuuuu5555555
1 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001110000000000000000
2 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
3 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
4 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001010000000
5 00000000000000000000000000010110101010101011111011101010101010101101111011011111011111110111011101111111111111111111
6 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
7 00000000000000000000000000001100101010001010110010110010100010001000110011000011011101110111111101110111111111111111
8 00000000000000000000000000000000000000000000000000000000000000000111111111111111111111111111111111111111111111111111
9 00000000000000000000000000011110101110101011111011111011101110101101111011011111111111111111111111111111111111111111
a 00000000000000000000000000011111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
b 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
c 00000000000000000000000000000000000000000000000000000000000000000000111000000111111100111111011000110101111111111111
d 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
e 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
f 00000000000000000000000000011111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
g 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
h 00000000000000000000000000011111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
i 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
```

The author's online program MATULA includes additional matrices `solx` and `soly`, which record the `MATE` and `QUEUE` information of subproblem solutions, so that an actual embedding of S into T can be exhibited when one exists.

214. (a) The average running time is less than 2 kilomems, and the standard deviation is very small. There are exactly 516399 pairs with $S \subseteq T$ (4.85%). Tree S_1 is embeddable the most (2016 Ts); tree $S = K_{1,12}$ is embeddable the least (31 Ts); tree T_1 has the most embedded subtrees (74 Ss); trees $T = P_{16}$ and $K_{1,15}$ have the fewest (1 S); trees S_2 and T_2 lead to the largest bipartite matching problem (5 boys, 14 girls).

$S_1 =$  $\subseteq$ max; $T_1 =$  $\supseteq$ max; $S_2 =$  $\not\subseteq$ $T_2 =$ .

(b) These tests, which take about 250 kilomems, find $S \subseteq T$ slightly more than half of the time, and rarely need bipartite matching with more than 3 boys.

(c) These tests take about 10 megamems, and find a unique embedding about 30% of the time. (About 5% of the time there are five or more.) The matching problems usually all have fewer than 5 boys and fewer than 9 girls.

215. If D has n vertices, there's a solution if and only if $\vec{K}_{n-k} \sqsubseteq D$.

216. Let there be a primary item v for each variable v, and let D_v be v's domain. Let there be a secondary item $u_i v_j$ for all elements of $N = \{(u,v,i,j) \mid u < v,\ i \in D_u,\ j \in D_v,\ (u,v) = (i,j) \text{ disallowed}\}$. There's one option for each v and each $j \in D_v$:

$$‘v \ \ \textstyle\bigvee_{(u,v,i,j)\in N} u_i v_j \ \ \bigvee_{(v,u,j,i)\in N} v_j u_i’.$$

Prestwich
Klavžar
ternary-to-binary encodings
unit clauses
weakened encoding

For radio coloring we can in fact do better. Let $D_v = [0\,..\,d)$ for all v, and $N = \{(u,v,i,j) \mid u \mathrel{—} v,\ u < v,\ |i-j| < 2\}$; introduce also a secondary item v_j for each v and j, meaning that v has a neighbor colored j. The option for v and j is then

$$\text{‘}v\ \ \textstyle\bigvee_{u — v}\, u_j\ \ \bigvee_{(u,v,i,j)\in N}\, u_i v_j\ \ \bigvee_{(v,u,j,i)\in N}\, v_j u_i\text{’}.$$

217. (a) 608 ($d = 7$); (b) 95520 ($d = 10$); (c) 3311464 ($d = 12$); (d) 401800 ($d = 11$).

218. (a) Yes. For example, $(\bar u_2 \vee \bar v_2)$ says that we don't have $u = v = 2$. (Like the binary encoding, it allows all pairs of binary bits in each variable's representation. But it's better, because it lumps 11 together with 10 instead of with 00.)

(b) $(000, 001, 01{*}, 1{*}{*})$. (See also the variable-length example in 6.2.2–(33).)

(c) Omit 000. (S. Prestwich introduced this alternative in order to study encodings that have many bit patterns assigned to a single value.)

219. $v_2v_1 = (00, 01, 10, 11)$ means $v = (0, 1, 2, 1 \text{ or } 2)$. The allowable $u_2u_1v_2v_1$ are 0001, 0010, 0011, 0100, 0110, 1000, 1001, 1100; hence $u \ne v$. (See also answer 240.)

220. Direct: (vars, clauses, totlits) $= (3V, 4V + 3E, 9V + 6E)$. Multivalued: $(3V, V{+}3E, 3V{+}6E)$. Log or Order: $(2V, V{+}3E, 2V{+}8E)$. Binary: $(2V, 6E, 24E)$. Support: $(3V, 4V + 6E, 9V + 18E)$. Weakened: $(3V, V + 3E, 3V + 12E)$. Reduced: $(2V, 3E, 8E)$. Prefix: $(2V, 3E, 10E)$. Curiously, the multivalued encoding has fewer total literals than the reduced encoding when $E > \frac{3}{2}V$, although it has more variables and more clauses.

221. By induction on n, the colors at the corners are uniquely determined: Given the colors of vertices $01\ldots1$ and $02\ldots2$, there are two ways to 3-color each of the subgaskets $1{*}\ldots{*}$ and $2{*}\ldots{*}$; but three of those four possibilities fail to hook up. [S. Klavžar, *Taiwanese Journal of Mathematics* **12** (2008), 513–522.]

222. True. There are two 3-colorings when $n = 1$. And any 3-coloring $\subseteq S_n^{(3)}$ with equal colors at two corners can be extended to a 3-coloring $\subseteq S_{n+1}^{(3)}$, in one or two ways.

223. The hint follows by induction. Consider three ternary-to-binary encodings

$$\begin{array}{lll} 0\rho = \bar 1; & 0\sigma = 1; & 0\tau = \bar 1;\\ 1\rho = \bar 1; & 1\sigma = \bar 1; & 1\tau = 1;\\ 2\rho = 1; & 2\sigma = \bar 1; & 2\tau = \bar 1; \end{array} \qquad \text{and let } a_1\ldots a_{n-1} \mapsto \Bigl(\bigl((a_1\rho)\ldots(a_{n-1}\rho)\bigr)_2,\ \bigl((1a_1\sigma)\ldots(a_{n-1}\sigma)\bigr)_2,\ \bigl((a_1\tau)\ldots(a_{n-1}\tau)\bigr)_2\Bigr).$$

For example, $1202 \mapsto \bigl((\bar1 1\bar1 1)_2, (1\bar1\bar1 1\bar1)_2, (1\bar1\bar1\bar1)_2\bigr) = (-5, 5, 1)$. It's easy to verify that $\alpha \mapsto (x, y, z)$ implies that x, y, and z are odd numbers with $x + y + z = 1$. Conversely, one can go back from such (x, y, z) to α, but only if α is a ternary vector $a_1 \ldots a_{n-1}$.

To go from the representation of triangle α to its three vertices $\alpha0$, $\alpha1$, or $\alpha2$, add respectively $(-1, -1, 1)$, $(-1, 1, -1)$, or $(1, -1, -1)$. The corner points are $0\ldots00 \mapsto (-2^n, 2^{n+1}, -2^n)$; $1\ldots11 \mapsto (-2^n, 0, 2^n)$; and $2\ldots22 \mapsto (2^n, 0, -2^n)$.

224. Assert the unit clauses $(\bar u_1)$, $(\bar u_2)$, (v_1), $(\bar v_2)$, $(\bar w_1)$, (w_2). In the weakened encoding, also assert $(\bar v_0)$ and $(\bar w_0)$.

225. Here are typical running times for Algorithm 7.2.2.2C, in units of 10^n mems:

	without clique hints								*with clique hints*								
	3	4	5	6	7	8	9	n	3	4	5	6	7	8	9	10	11
Dir	1.0	4.3	4.7	4.1	5.3	7.6	12.0		1.2	4.0	4.3	3.7	3.5	4.5	7.9		
Mul	0.9	4.6	4.3	4.1	4.8	7.1	13.1		1.2	4.1	3.4	3.8	4.3	4.9	7.5		
Log	0.9	3.1	3.9	3.9	3.2	4.9	8.1		1.1	4.0	3.6	3.0	2.8	3.4	5.8	10.7	23.5

Bin	28.9	48.2	70.3	78.5	89.7											
Sup	2.4	6.4	6.9	5.8	5.9	6.9	13.3	2.8	7.8	7.0	6.2	6.0	6.8	9.6		
Wea	9.1	14.9	14.5	14.7	16.9	23.6	27.0									
Red	0.8	3.9	3.3	3.4	3.2	3.7	8.6	1.0	3.8	3.8	2.7	2.6	3.1	6.5	12.9	24.0
Pre	3.4	7.7	7.5	5.7	6.4	7.5	11.2	1.4	2.8	3.4	3.0	2.3	3.0	4.8	7.3	14.6

Tests were omitted when there was little chance of success. Since runtime depends on random choices, each experiment was done nine times; the median is shown. (For example, the values for Pre when $n = 11$ ranged from 11.2 to 16.1.) Algorithm 7.2.2.2L gives very similar results when $n \le 5$. But for $n \ge 6$ it often loses focus and takes forever.

226. For $n > 1$ let X_n be the search tree size for 3-coloring $S_n^{(3)}$ when distinct colors have already been specified for vertices $0\ldots00$ and $0\ldots01$; also let Y_n be similar, but for vertices $0\ldots00$ and $1\ldots11$; and let Z_n be similar, but for $\widehat{S}_n^{(3)}$ instead of $S_n^{(3)}$. Then $X_2 = Y_2 = 5$ and $Z_2 = 2$. Also $X_{n+1} = X_n + (X_n + Y_n - 1) + (X_n + 2Z_n)$; $Y_{n+1} = 1 + (X_n + Y_n - 1 + Y_n - 1) + (X_n + 2Z_n)$; $Z_{n+1} = X_n + Y_n - 1 + 2Z_n$. Via generating functions we find $X_{n+1} = Y_{n+1} = 2^{n-1}L_{2n+1} + 1$ and $Z_{n+1} = 2^{n-1}L_{2n} - 1$, where $L_m = F_{m+1} + F_{m-1}$ is a Lucas number. The answer, if we decide to save a factor of 6 by prespecifying the colors of $0\ldots00$ and $0\ldots01$, is $Z_n = \lfloor (3+\sqrt{5}\,)^{n-1}/2 \rfloor$.

227. For all $n > 2$, Filip Stappers has constructed 2^{n-1} sets of $2^{n-1}+1$ removable edges as follows: Each set includes the two "tip" edges that touch vertex $2\ldots2$, plus $2^{n-1}-1$ non-tip edges. When $n = 2$ the non-tip edge is $00 \mathbin{—} 01$, which can also be written $11 \mathbin{—} 01$; these two forms have different progeny. When $n > 2$, replace each non-tip edge $\alpha a \mathbin{—} \alpha b$ for $n-1$ by two non-tip edges $\alpha aa \mathbin{—} \alpha ac$, $\alpha bb \mathbin{—} \alpha bc$, where $c \notin \{a, b\}$; also add either $2\ldots200 \mathbin{—} 2\ldots201$ or $2\ldots211 \mathbin{—} 2\ldots210$ as a further non-tip edge.

For example, when $n = 4$ we get eight sets of nine removable edges, such as: $0000 \mathbin{—} 0001$, $0022 \mathbin{—} 0021$, $0111 \mathbin{—} 0110$, $0122 \mathbin{—} 0120$, $2000 \mathbin{—} 2002$, $2011 \mathbin{—} 2012$, $2200 \mathbin{—} 2201$, $2202 \mathbin{—} 2222$, $2212 \mathbin{—} 2222$. (This construction actually produces *all* of the largest removable sets when $n \le 4$. Is that conjecture actually true for all n?)

228. (a, c) $(u_1 \vee v_1 \vee w_1) \wedge (u_2 \vee v_2 \vee w_2)$.
(b) $(u_0 \vee v_0 \vee w_0) \wedge (u_1 \vee v_1 \vee w_1) \wedge (\bar{u}_1 \vee \bar{v}_1 \vee \bar{w}_1) \wedge (u_2 \vee v_2 \vee w_2)$.
(d) $(u_1 \vee v_1 \vee w_1) \wedge (\bar{u}_1 \vee \bar{v}_1 \vee \bar{w}_1) \wedge (u_2 \vee v_2 \vee w_2)$.

229. If $(\neg 0202_2)$ and $(\neg 0222_2)$, the clique hint $(0202_2 \vee 0212_2 \vee 0222_2)$ implies (0212_2). Hence $(\neg 0201_2)$; and $(0022_2 \vee 0201_2 \vee 0202_2)$ implies (0022_2), contradicting (0002_2).

230. The construction of exercise 7.2.2.1–117(b) nicely sets up an exact cover problem with $\lceil 3^n/2 \rceil$ primary items, 3^n secondary items, and $\lceil 3^{n+1}/2 \rceil - 5$ options, each of size at most 3. (The colors of the first three vertices are forced.) And Algorithm 7.2.2.1X nicely proves uncolorability, with a search tree of size $Z_n = O(5.24^n)$ (see exercise 226). Indeed, only a few links need to dance at every node of that tree.

But there's a catch! The author's implementation of step X3, which was used for many of the experiments in Section 7.2.2.1, looks at the `LEN` fields of *every* uncovered primary item, when choosing the item for branching by MRV (see exercise 7.2.2.1–9). So his implementation incurs a cost of $\Omega(3^n/2 - l)$ for each node at level l. That's foolish — because in this problem all the `LEN` fields are at most 3!

A better implementation of step X3 solves the problem with only $O(n)$ steps of computation at each node, while making precisely the same choices: We maintain heap-ordered lists of all uncovered primary items that have a given length. (See Algorithm 5.2.3H.) Then we simply choose the smallest item in the nonempty list of least `LEN`. With that improvement the running time for $\widehat{S}_n^{(3)}$ is $\approx (1.7n + 12)(3 + \sqrt{5}\,)^n \mu$.

focus
recurrence relations
generating functions
Lucas number
Fibonacci numbers
Stappers
dance
author
`LEN` fields
primary item
MRV
heap-ordered lists

In this particular problem it turns out to be very important to choose the smallest item each time; otherwise the algorithm gets lost and exercise 226 does not apply. Notice that the secondary items (three per clique) could actually be made primary; surprisingly, however, that changes the order of exploration and messes everything up.

231. The same question can of course be asked for $\widehat{S}_n^{(d)}$ and $\overline{S}_n^{(d)}$.

232. We observed in Section 7.2.2.2 that Algorithm 7.2.2.2L is hopelessly slow for this problem. The clique-hinted runtimes shown here are in units of q^2 kilomems. One of several surprises in this experiment is that the weakened encoding performs much better than expected, especially when q is small.

q	29	49	99	199	399	799	1599	3199
Dir	58	44	46	40	52	56	40	41
Mul	96	60	53	36	57	99	62	
Log	35	24	24	28	29	33	19	17
Sup	148	169	111	99	109	152	105	
Red	53	29	31	27	34	37	25	22
Wea	32	35	41	55	66	51	39	58
Pre	47	31	35	35	77	77	35	38

233. (The author hopes that some reader will supply a good answer. His best so far is to remove nine edges, such as these: $a_0 — b_1$, $e_0 — d_1$, $f_0 — c_1$, $a_1 — b_1$, $b_1 — c_1$, $b_1 — d_1$, $c_1 — d_1$, $c_1 — e_1$, $d_1 — f_1$.)

234. $013 = 031$; $113 = 131$; $123 = 132$; $133 = 311$; $213 = 231$; $312 = 321$; $313 = 331$.

235. A path of length 2^{n-1}. (And $\overline{S}_n^{(2)}$ is a $(2^{n-1}+1)$-cycle.)

236. True: Consider the vertices $a_1 \ldots a_n$ with $a_j < d$ for all j. (And we can independently remap the coordinates of those vertices in $d'^{\underline{d}} = d'(d'-1)\ldots(d'-d+1)$ ways.)

237. The pure vertices $j \ldots j$ for $0 \le j < d$ ($S_n^{(d)}$), $2 \le j < d$ ($\widehat{S}_n^{(d)}$), $1 \le j < d$ ($\overline{S}_n^{(d)}$).

238. (a) $d^n(d-1)/2$ clique edges; $(d^n - d)/2$ nonclique edges.

(b) Contract all the nonclique edges.

(c) Add a loop to each pure vertex $j \ldots j$, then take the line graph.

[The graphs $s_n^{(d)}$ for arbitrary d were introduced by S. Klavžar and U. Milutinović, in *Czechoslovak Mathematical Journal* **47** (1997), 95–104; a few years later, M. Jakovac, in *Ars Combinatoria* **116** (2014), 395–405, introduced $S_n^{(d)}$. For a comprehensive survey of graph-theoretical properties satisfied by these and similar graphs, see A. M. Hinz, S. Klavžar, and S. S. Zemljič, *Discrete Applied Mathematics* **217** (2017), 565–600.]

239. Each of d^n vertex labels receives a color, and each color c appears d^{n-1} times—once in every clique. And c appears an even number of times on the impure labels, since they're paired up. So its pure appearances are congruent to d^{n-1} (modulo 2).

Incidentally, a d-coloring of $S_2^{(d)}$ is essentially a self-transpose $d \times d$ latin square.

240. Each variable v must be represented individually. Direct and Support: d Boolean variables $v_j = [v = j]$, with the at-least-one clause $(v_0 \vee \cdots \vee v_{d-1})$ and $\binom{d}{2}$ at-most-one clauses $\bar{v}_i \vee \bar{v}_j$. Multivalued and Weakened: Omit those at-most-one clauses. (If $v_j = 1$ and $v_k = 0$ for $j < k < d$ in the weakened encoding, $v = j$.) Log: $l = \lceil \lg d \rceil$ variables v_1, v_2, v_4, $\ldots$, denoting $v = (\ldots v_4 v_2 v_1)_2$. Assert clauses of length l to exclude the cases $d \le v < 2^l$. (Those clauses can often be shortened; for example, to exclude $v > 4$ when $d = 5$ it suffices to assert $(\bar{v}_4 \vee \bar{v}_2)$ and $(\bar{v}_4 \vee \bar{v}_1)$.) Prefix: Again $\lceil \lg d \rceil$ variables, but there are no constraints; $v = j$ is represented by the path to the jth leaf in the complete binary tree with j external nodes. For example, the five values when $d = 5$ are represented by $v_4 v_2 v_1 = 000$, 001, $01{*}$, $10{*}$, $11{*}$, effectively lumping together the binary values $\{2,3\}$, $\{4,5\}$, $\{6,7\}$. Reduced: $d-1$ variables $v_j = [v = j]$ for $0 < j < d$. Order: $d-1$ variables $v^j = [v \ge j]$ for $0 < j < d$; assert $(\bar{v}^j \vee v^{j-1})$ for $1 < j < d$.

We also must assert clauses to prohibit $u = j$ and $v = j$. Direct, Multivalued: $(\bar{u}_j \vee \bar{v}_j)$. Reduced: Same, but assert $(u_1 \vee \cdots \vee u_{d-1} \vee v_1 \vee \cdots \vee v_{d-1})$ when $j = 0$.

weakened encoding
author
pure vertices
contraction of a graph
line graph
Klavžar
Milutinović
Jakovac
Hinz
Zemljič
self-transpose
latin square
complete binary tree

Log: Assert a clause of length $2l$ from the binary representation of j; for example, when $l = 3$ and $j = 4$, assert $(\bar{u}_4 \vee u_2 \vee u_1 \vee \bar{v}_4 \vee v_2 \vee v_1)$. (However, that clause can be shortened to $(\bar{u}_4 \vee u_1 \vee \bar{v}_4 \vee v_1)$ when $d = 6$, and to $(\bar{u}_4 \vee \bar{v}_4)$ when $d = 5$.) Support: Assert $(\bar{u}_j \vee v_1 \vee \cdots \vee v_{j-1} \vee v_{j+1} \vee \cdots \vee v_{d-1})$, and the same with $u \leftrightarrow v$. Weakened: Assert $(\bar{u}_j \vee u_{j+1} \vee \cdots \vee u_{d-1} \vee \bar{v}_j \vee v_{j+1} \vee \cdots \vee v_{d-1})$. Prefix: Assert a clause of length $2l$ or $2l-2$ based on the path to leaf j. For example, when $d = 5$ and j corresponds to $\{4,5\}$, assert $(\bar{u}_4 \vee u_2 \vee \bar{v}_4 \vee v_2)$. (See exercise 7.2.2.2–391(c).) Order: Assert $(\bar{u}^j \vee u^{j+1} \vee \bar{v}^j \vee v^{j+1})$; but omit $\bar{u}^0$, $\bar{v}^0$, u^d, v^d (which are always false). SAT

241. We assume that all domain sizes are d, and that we want to assert all possible hints when the underlying constraint graph has a d-clique $\{v^{(1)}, \ldots, v^{(d)}\}$. Let v_k be one of the Boolean variables representing vertex v. If we know that $v_k = 1$ for at least one v in any c-clique, where $3 \le c \le d$, we can assert the positive clause $(v^{(i_1)} \vee \cdots \vee v^{(i_c)})$ for all $\binom{d}{c}$ subsets $\{i_1, \ldots, i_c\} \subseteq \{1, \ldots, d\}$. Similarly, if we know that $v_k = 0$ for at least one such v, we can assert the negative clause $(\bar{v}^{(i_1)} \vee \cdots \vee \bar{v}^{(i_c)})$ for all such subsets.

Let's assume, for example, that the vertices $\{u, v, w, x, y\}$ form a clique when $d = 5$. Direct, Multivalued, Support, and Reduced have positive hints $(u_j \vee v_j \vee w_j \vee x_j \vee y_j)$ for $0 \le j < d$; we must, however, omit $j = 0$ in the reduced encoding, where v_0 doesn't exist. Log encoding, likewise, has $(u_4 \vee v_4 \vee w_4 \vee x_4 \vee y_4)$; and when $j \in \{1, 2\}$ it also has five positive clauses for $c = 4$, namely $(u_j \vee v_j \vee w_j \vee x_j)$, …, $(v_j \vee w_j \vee x_j \vee y_j)$, as well as ten negative clauses for $c = 3$, such as $(\bar{u}_j \vee \bar{v}_j \vee \bar{w}_j)$. Thus, Log has $1+5+5+10+10 = 31$ hints altogether, for every 5-clique(!). Order has even more: Positive for $cj \in \{32, 43, 54\}$ and negative for $cj \in \{33, 42, 51\}$, totalling $10 + 5 + 1 + 10 + 5 + 1 = 32$. (Examples are the hints $(\bar{u}^1 \vee \bar{v}^1 \vee \bar{w}^1 \vee \bar{x}^1 \vee \bar{y}^1)$, $(\bar{u}^2 \vee \bar{v}^2 \vee \bar{w}^2 \vee \bar{x}^2)$, and $(u^3 \vee v^3 \vee w^3)$.) And Prefix has positive hints for $cj \in \{42, 44, 51\}$, negative hints for $cj \in \{33, 34, 51\}$, also totalling 32. Finally, Weakened has positive hints for $cj \in \{50, 51, 52, 53, 54\}$, negative hints for $cj \in \{33, 42\}$.

242. With hints for 17 8-cliques (7 rows, 8 columns, and two long diagonals; the top row is already forced), the time for Algorithm 7.2.2.2C to prove unsatisfiability goes down dramatically, from 9813 Mμ to 0.8 Mμ (median of nine runs) — better than K6!

243. Suppose we have a 4-coloring h, with $h(a_1 \ldots a_n) \in \{0, 1, 2, 3\}$ for all vertices $a_1 \ldots a_n$. If π is any permutation of $\{1, 2, 3\}$, let $0\pi = 0$. Then $h'(a_1 \ldots a_n) = h((a_1\pi) \ldots (a_n\pi))\pi^-$ is a 4-coloring; and $h'(0 \ldots 0j) = h(0 \ldots 0(j\pi))\pi^- = j\pi\pi^- = j$.

Consequently we can assume without loss of generality that $h(0 \ldots 011) = 0$. Let v_k be the vertex $a_1 \ldots a_n$ such that $k = (a_1 \ldots a_n)_2$. Then the sequence $h(v_1)$, $h(v_3)$, $h(v_5)$, …, $h(v_{2^n-1})$ begins 1, 0, and ends with 2 or 3. So there's a first odd index j with $h(v_j) > 1$, and we can assume without loss of generality that $h(v_j) = 2$.

We could exploit this when backtracking to save a factor of at least 3. But if we are using SAT, the assertions $(0 \ldots 011_0)$ and $(\neg 0 \ldots 101_3)$ *don't* actually give any speedup.

244. (Prefix = Log when $d = 4$.) To avoid decimal points in the table below, the running times are given in units of 10^{2n-4} mems, rounded to two significant digits.

	$\overline{S}_3^{(4)}$	$\overline{S}_4^{(4)}$	$\overline{S}_5^{(4)}$	$\overline{S}_6^{(4)}$	$\overline{S}_7^{(4)}$
Dir	580	200	24	12	13
Mul	580	130	33	18	17
Log	5900	2600	440	62	48
Sup	2100	800	250	85	24
Wea	140000	8900	1700	290	450
Red	9100	5000	200	20	24
Ord	680	130	26	11	16

	$\widehat{S}_3^{(5)}$	$\widehat{S}_4^{(5)}$	$\widehat{S}_5^{(5)}$	$\widehat{S}_6^{(5)}$
Dir	460	650	160	25
Mul	22000	2500	250	30
Log	8400	6700	1600	
Sup	8700	3900	480	
Pre	6100	4700	1200	1600
Red	16000	1800	180	17
Ord	2500	1700	320	150

Kissat
Intel Xeon computer
Biere
clique
permutation
pigeonhole principle
domain inconsistency
direct representation
$\wp$, tautology
clique hints
multivalued encoding
van Dongen
benchmarks
competition

245. Similarly, the table entries above are in units of 10^{2n-2} mems. Reasons for the sterling performance of the direct encoding when $n=3$, and for the poor performance of the prefix encoding when $n=6$, are unknown.

246. Here are the clique-hint runtimes for Kissat 2022-light on an Intel Xeon computer, model E5-2620 v4 2.1GHz, reported by Armin Biere. (The units for $\widehat{S}_n^{(3)}$, $\overline{S}_n^{(4)}$, $\widehat{S}_n^{(5)}$ are respectively 10^{n-11}, 10^{n-7}, and 10^{n-5} sec; the units for $L_q = L(J_q)$ are q^2 μsec. Algorithm 7.2.2.2C is totally eclipsed on the $\widehat{S}_n^{(3)}$ and $\overline{S}_n^{(4)}$ benchmarks!)

	$\widehat{S}_9^{(3)}$	$\widehat{S}_{10}^{(3)}$	$\widehat{S}_{11}^{(3)}$	$\widehat{S}_{12}^{(3)}$	$\overline{S}_6^{(4)}$	$\overline{S}_7^{(4)}$	$\overline{S}_8^{(4)}$	$\widehat{S}_5^{(5)}$	$\widehat{S}_6^{(5)}$	$\widehat{S}_7^{(5)}$	$\widehat{S}_8^{(5)}$	L_{1023}	L_{2047}	L_{4095}	L_{8191}
Dir	170	84	45	26	6	7	9	8	6	28	31	62	57	49	15
Mul	150	81	45	24	10	11	19	34	40	37		28	23	17	13
Log	110	60	32	18	54	38	27	40	39	38		23	18	14	10
Sup	180	73	54	28	25	31	33	47	57	59		20	15	10	6
Red	100	59	35	18	17	100	46	13	11	11	14	21	20	14	10
Wea	120	85	50	29	170	210	340					32	19	14	11
Pre	60	44	33	21	54	38	27	55	74	76		30	24	27	19
Ord	110	60	32	18	7	14	51	22	26	30	25	23	18	14	10

247. (a) Each of the n clusters $\{x_{ij} \mid 0 \le j < n\}$ is an n-clique, so their values must be a permutation of the domain. If $i>0$ and $j>0$, $x_{i0}<2$ implies $x_{ij}\ge 2$; hence $x_{i0}\ge 2$. So the $n-1$ variables $\{x_{01},\ldots,x_{0(n-1)}\}$ have only $n-2$ available values.

(b) Since there really are only n^2-n+1 variables, by (iii), we can identify x_{i0} with x_{0i}. Let there be $2n^2-n+1$ primary items x_{ij} and v_{ij} for $0\le i,j<n$, omitting x_{0j} when $j>0$. Introduce $2(n-1)^2$ secondary items a_{ij} and b_{ij} for $0<i,j<n$, in order to forbid $(x_{i0},x_{ij})=(0,1)$ and $(1,0)$. There's an option containing x_{ij} and v_{ik} for each $0\le i,j,k<n$ except when $i=0$ and $j>0$. If $i>0$ and $j=0$ that option contains also v_{0k}, as well as $a_{ij'}$ for $0<j'<n$ when $k=0$, and $b_{ij'}$ for $0<j'<n$ when $k=1$. If $i>0$ and $j>0$ it contains also b_{ij} when $k=0$ or a_{ij} when $k=1$.

The running time for Algorithm 7.2.2.1X is approximately proportional to $(n-1)!$, if the primary items have their natural order; for example, it's 105 Mμ when $n=8$ and 90 Gμ when $n=12$. But the time is much, much longer when they're randomly ordered (e.g., 1880 Gμ when $n=7$). On the other hand, Algorithm 7.2.2.1P quickly proves unsatisfiability in $\Theta(n^4)$ steps, because the domains of x_{ij} and v_{ij} are inconsistent. For example, it needs only 22 Mμ to remove all options when $n=32$.

(c) Use, for instance, the direct representation, with $x_{ijk}=[x_{ij}=k]$; identify x_{i0k} with x_{0ik}. The clauses for clique i are $A_i\wedge B_i\wedge C_i\wedge D_i$ for $0\le i<n$, where

$$A_i = \textstyle\bigwedge_{j=0}^{(n-1)[i\ne 0]}\Bigl(\bigl(\bigvee_{k=0}^{n-1} x_{ijk}\bigr)\wedge \bigwedge_{0\le k<k'<n}(\bar x_{ijk}\vee \bar x_{ijk'})\Bigr) \qquad \text{[domain constraints]};$$

$$B_i = \textstyle\bigwedge_{0\le j<j'<n}\bigwedge_{k=0}^{n-1}\bigl(i>0?\ (\bar x_{ijk}\vee \bar x_{ij'k})\colon (\bar x_{j0k}\vee \bar x_{j'0k})\bigr) \qquad \text{[clique constraints]};$$

$$C_i = \textstyle\bigwedge_{k=0}^{n-1}\bigl(i>0?\ \bigl(\bigvee_{j=0}^{n-1} x_{ijk}\bigr)\colon \bigl(\bigvee_{j=0}^{n-1} x_{j0k}\bigr)\bigr) \qquad \text{[clique hints]};$$

$$D_i = \bigl(i>0?\ \textstyle\bigwedge_{j=1}^{n-1}\bigl((\bar x_{i00}\vee \bar x_{ij1})\wedge(\bar x_{i01}\vee \bar x_{ij0})\bigr)\colon \wp\bigr) \qquad \text{[constraint (ii)]}.$$

Thanks to the clique hints, classical SAT solvers handle this problem quite well. For example, in nine runs for $n=32$ with different random seeds, the median time for Algorithm 7.2.2.2L was 59 Mμ, and Algorithm 7.2.2.2C needed only 2.4 Mμ. But without the clique hints the runtime is exponential—for example 270 Gμ with 7.2.2.2C for $n=11$. The multivalued encoding does poorly too (280 Gμ), even with clique hints.

[This problem was introduced by M. R. C. van Dongen as one of the benchmarks for the 2nd international CSP solver competition in 2006. In the competition, of course, only the variables, domains, and constraints were given, and variable names were randomized. A mechanical solver wouldn't be able to deduce unsatisfiability efficiently without somehow understanding the clique structure, and introducing something like the v_{ij} items of (b) or the hints of (c).]

248. Changing the notation to gain symmetry, let's encode '$u+v \geq 2^n-1+t$', where $u = (u_{n-1}\ldots u_0)_2$ and $v = (v_{n-1}\ldots v_0)_2$. It's the same problem, since $\bar{u} = (\bar{u}_{n-1}\ldots\bar{u}_0)_2 = 2^n-1-u$. There are no constraints if $t \leq 1-2^n$; there are no solutions if $t \geq 2^n$.

For all $n>0$ and $1-2^n < t < 2^n$, let $a_{n,t}$ be an auxiliary variable and construct the following clauses: (i) $(\bar{a}_{n,t} \vee u_{n-1} \vee v_{n-1})$ if $0 \leq t < 2^{n-1}$; (i′) $(\bar{a}_{n,t} \vee u_{n-1} \vee v_{n-1} \vee a_{n-1,t+2^{n-1}})$ if $t<0$; (ii) $(\bar{a}_{n,t} \vee u_{n-1} \vee a_{n-1,t})$; (iii) $(\bar{a}_{n,t} \vee v_{n-1} \vee a_{n-1,t})$; (iv) $(\bar{a}_{n,t} \vee a_{n-1,t-2^{n-1}})$, if $t>1$ and $n>1$. (In cases (ii) and (iii), omit $a_{n-1,t}$ if $t \geq 2^{n-1}$.) Then $u+v \geq 2^n-1+t$ if and only if u and v satisfy these clauses with $a_{n,t}=1$, for some values of the other auxiliary variables.

(We can remove $\bar{a}_{n,t}$, and all clauses that contain pure literals of the form $\bar{a}_{n',t'}$.)

For instance, $t=-1$ encodes '$u \leq v+1$': $(\bar{u}_8 \vee v_8 \vee a_{3,7})$, $(\bar{u}_8 \vee a_{3,-1})$, $(v_8 \vee a_{3,-1})$, $(\bar{a}_{3,7} \vee \bar{u}_4)$, $(\bar{a}_{3,7} \vee v_4)$, $(\bar{a}_{3,7} \vee a_{2,3})$, $(\bar{a}_{3,-1} \vee \bar{u}_4 \vee v_4 \vee a_{2,3})$, $(\bar{a}_{3,-1} \vee \bar{u}_4 \vee a_{2,-1})$, $(\bar{a}_{3,-1} \vee v_4 \vee a_{2,-1})$, $(\bar{a}_{2,3} \vee \bar{u}_2)$, $(\bar{a}_{2,3} \vee v_2)$, $(\bar{a}_{2,3} \vee a_{1,1})$, $(\bar{a}_{2,-1} \vee \bar{u}_2 \vee v_2 \vee a_{1,1})$, $(\bar{a}_{1,1} \vee \bar{u}_1)$, $(\bar{a}_{1,1} \vee v_1)$. And '$u \leq v-2$' is $(\bar{u}_8 \vee v_8)$, $(\bar{u}_8 \vee a_{3,2})$, $(v_8 \vee a_{3,2})$, $(a_{3,-6})$, $(\bar{a}_{3,2} \vee \bar{u}_4 \vee v_4)$, $(\bar{a}_{3,2} \vee \bar{u}_4 \vee a_{2,2})$, $(\bar{a}_{3,2} \vee v_4 \vee a_{2,2})$, $(\bar{a}_{3,2} \vee a_{2,-2})$, $(\bar{a}_{3,-6} \vee \bar{u}_4 \vee v_4 \vee a_{2,-2})$, $(\bar{a}_{2,2} \vee \bar{u}_2)$, $(\bar{a}_{2,2} \vee v_2)$, $(\bar{a}_{2,2} \vee a_{1,0})$, $(\bar{a}_{2,-2} \vee \bar{u}_2 \vee v_2 \vee a_{1,0})$, $(\bar{a}_{1,0} \vee \bar{u}_1 \vee v_1)$.

249. The shortest "covering" is $(\bar{u}_0 \vee v_0 \vee \bar{w}_1) \wedge (u_0 \vee w_1) \wedge (\bar{u}_1 \vee v_2) \wedge (\bar{u}_2 \vee v_1) \wedge (v_1 \vee \bar{w}_2)$.

250. Besides the at-least-one and at-most-one clauses, the direct encoding has preclusion clauses $(\bar{u}_0 \vee \bar{v}_2) \wedge (\bar{u}_1 \vee \bar{v}_0) \wedge (\bar{u}_1 \vee \bar{v}_1) \wedge (\bar{u}_2 \vee \bar{v}_1) \wedge (\bar{u}_2 \vee \bar{v}_2)$, while the support encoding has $(\bar{u}_0 \vee v_0 \vee v_1) \wedge (\bar{u}_1 \vee v_2) \wedge (\bar{u}_2 \vee v_0) \wedge (\bar{v}_0 \vee u_0 \vee u_2) \wedge (\bar{v}_1 \vee u_0) \wedge (\bar{v}_2 \vee u_1)$.

251. $(R_{00} \vee R_{01} \vee R_{12} \vee R_{20}) \wedge (\overline{R}_{00} \vee u_0) \wedge (\overline{R}_{00} \vee v_0) \wedge (\overline{R}_{01} \vee u_0) \wedge (\overline{R}_{01} \vee v_1) \wedge (\overline{R}_{12} \vee u_1) \wedge (\overline{R}_{12} \vee v_2) \wedge (\overline{R}_{20} \vee u_2) \wedge (\overline{R}_{20} \vee v_0) \wedge (\bar{u}_0 \vee R_{00} \vee R_{01}) \wedge (\bar{u}_1 \vee R_{12}) \wedge (\bar{u}_2 \vee R_{20}) \wedge (\bar{v}_0 \vee R_{00} \vee R_{20}) \wedge (\bar{v}_1 \vee R_{01}) \wedge (\bar{v}_2 \vee R_{12})$ and the at-least-one, at-most-one clauses for u and v.

252. After deducing u_0, v_0, w_0, we have (for example) $\bar{w}_1$; hence $\overline{R}_{001}$.

253. (a) There are $N = d_1\ldots d_k - G$ clauses of length k, hence Nk literals altogether.

(b) The clause exemplified by (80) has G literals; the Gk clauses like the left of (81) each have 2; the $d_1+\cdots+d_k$ clauses like the right of (81) have a total of $d_1+\cdots+d_k+Gk$. So the grand total is $(3k+1)G+d_1+\cdots+d_k$.

254. Consider a general relation R as in exercise 253, with Boolean variables v_{ja} for $1 \leq j \leq k$ and $0 \leq a < d_j$. Then $R(a_1,\ldots,a_k)$ is true if and only if every preclusion clause is satisfied with v_{ja_j} true for $1 \leq j \leq k$ and the other Boolean variables arbitrary. (The reduced encoding without at-most-one is the "multivalued encoding"; see Table 2.)

255. Let C_a be the clause for $a \in D_u$, and let $C = \bigvee\{u_a \mid a \in D_u\}$ be u's at-least-one clause. Given $b \in D_v$, resolve C with each C_a for which $ab \notin R = R(u,v)$; this gives $C' = U_b \vee V_b$, where $U_b = \bigvee\{u_a \mid ab \in R\}$, $V_b = \bigvee\{v_c \mid c \in R'_b\}$, and $R'_b = \{c \mid ac \in R$ for some a with $ab \notin R\}$. If $R'_b \neq \emptyset$, we get the desired clause $(\bar{v}_b \vee U_b)$ by resolving C' with $(\bar{u}_c \vee \bar{u}_b)$ for each $c \in R'_b$. Otherwise the desired clause is subsumed by U_b, which can be obtained by resolving C with C_a for all $a \in D_u$ that have no support in R.

(The other half of the clauses are, however, important for *unit* resolution.)

256. Form the 27-bit vectors for the set of all 2^9 truth tables a_i on (x_1,x_2,x_3) that define binary relations on (x_1,x_2); also similar vectors b_j and c_k for (x_1,x_3) and (x_2,x_3). The number of distinct a_i & b_j & c_k is 1614530, which is $\approx 1.2\%$ of 2^{27}. (The answer to the analogous question for domain size 2 is 166, by exercise 7.2.2.2–191.)

auxiliary variable
pure literals
covering
preclusion clauses
multivalued encoding
truth tables
bitwise AND

257. There are 111618 classes; they form 55809 pairs, because the complements of equivalent relations are equivalent. One of the pairs has classes of size 1 (the empty relation and the full relation). Another pair has classes of size 9 (for example, '$x = 0$' and '$x \neq 0$'). Another has classes of size 12 ('$(x{\pm}y{\pm}z) \bmod 3 = \text{const}$' or '$(x{\pm}y{\pm}z) \bmod 3 \neq \text{const}$', analogous to the parity relations mod 2). Then there's size 18 (like '$x = y$' or '$x \neq y$'). The class containing '$x = y = z = 0$' is one of 20 classes of size 27. The class containing 'x, y, and z are distinct' is one of 4 classes of size 36; so is the class containing '$x = y = z$'. Exactly 12722 classes have size 648; 96726 classes have the max size, 1296.

The 1614530 decomposable relations in answer 256 form 1841 equivalence classes. Those classes are *not* closed under complementation; for example, '$\langle xyz\rangle = 1$', whose class has size 108, is decomposable; but '$\langle xyz\rangle \neq 1$' differs in six places (x, y, z) from the intersection of its projections onto $\{x, y\}$, $\{x, z\}$, $\{y, z\}$. Altogether 6034 of the classes, and 6496994 of the relations ($\approx 4.8\%$), are within 1 of that intersection; 65623736 relations are within 5. Only the class that contains '$(x+y+z) \bmod 3 = 0$' is at distance 18.

258. Yes; it's not difficult to prove that $R(u, v, w) = P(u, v) \wedge P(u, w) \wedge P(v, w)$, where $P(u, v) = (\max(u, v) \geq c) \wedge (\min(u, v) \leq c)$.

259. `hells`, `shart`, and `trice`. (But `hells` and `trice` are in `WORDS`(3500).)

260. (a) If $a_1 \ldots a_k \in R$ there's a solution with $v_{1a_1} = \cdots = v_{ka_k} = 1$.

(b) All clauses are satisfied, and the value of every literal has been unambiguously forced. Furthermore exactly one v_{ja_j} is true for each j.

(c) If v_a becomes false, D_v loses the value a. If v_a becomes true, all $v_{a'}$ for $a' \neq a$ become false; we're left with the support clauses for a relation on the variables $\neq v$.

(d) The current relation R' has at least two elements in D_v.

(e, f) The arguments in (a), (b), (c) remain valid.

Historical note: F. Rossi, C. J. Petrie, and V. Dhar [*ECAI* **9** (1990), 550–556] described the "hidden variable" trick as part of the CSP folklore; U. Montanari had alluded to it on page 105 of his paper of 1974.

261. Introduce secondary items w_2x_2, w_1y_0, $\ldots$, y_2z_1 for the excluded pairs. The options are then 'w w_0z_2', 'w w_1y_0', 'w w_2x_2 w_2y_0'; 'x x_0z_0 x_0z_2', 'x x_1y_1 x_1y_2', 'x w_2x_2 x_2y_2'; 'y w_1y_0 w_2y_0 y_0z_0 y_0z_1', 'y x_1y_1 y_1z_0 y_1z_1', 'y x_1y_2 x_2y_2 y_2z_1'; 'z x_0z_0 y_0z_0 y_1z_0', 'z y_0z_1 y_1z_1 y_2z_1', 'z w_0z_2 x_0z_2'.

262. Now there are six primary items, $\{wx, wy, wz, xy, xz, yz\}$, while $\{w, x, y, z\}$ are secondary. There are $8 + 7 + 8 + 6 + 7 + 4$ options, listing the "positive" tuples. For example, the options for wx are 'wx $w{:}0$ $x{:}0$', 'wx $w{:}0$ $x{:}1$', $\ldots$, 'wx $w{:}2$ $x{:}1$'; the options for yz are 'yz $y{:}0$ $z{:}2$', 'yz $y{:}1$ $z{:}2$', 'yz $y{:}2$ $z{:}0$', 'yz $y{:}2$ $z{:}2$'. (By contrast, answer 261 used the "negative" tuples that were expressly forbidden in (87). In this instance, negative beats positive.)

263. (a) True. An inactive variable has been assigned the (unique) value in its domain.

(b) False. Any or all variables in a given problem might have a singleton domain.

(c) False. An empty domain is always weakly viable (indeed, viable), because Definition V is satisfied vacuously. If a domain becomes empty while maintaining forward consistency, we are justified in backtracking immediately; but that may be inconvenient. Sometimes it's best to wait for the next level of search to discover an empty domain.

(d) Even more false than (c)! An active variable with empty domain cannot appear in the same constraint as an active variable with nonempty domain.

(e) True, unless there are unary constraints — which must match the domains.

(f, g, h) True. The only constraint still involves two or more active variables.

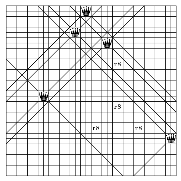
parity
all-different
median of three
projections
Historical note
Rossi
Petrie
Dhar
hidden variable
Montanari
positive versus negative table constraints
negative versus positive table constraints

(i, j) False. But would be true if D_z were reduced to $\{2,3\}$.

wipeout
Weigel
fourfold symmetry
Lecoutre

264. If the current partial solution of a coloring problem is FC but not DC, some active binding (v,a) is unviable. Hence v is adjacent to a vertex w with $D_w \subseteq \{a\}$; and w is active (by FC). So we'll remove a from D_v when we maintain FC after assigning $w = a$.

265. Placing a queen in some row or column reduces the number of unattacked cells in another row or column by at most 3. Thus no wipeout is possible until some domain has size ≤ 3.



But five queens placed as shown leave r_8 with only two free cells, allowing DC to forbid four potential placements. (Incidentally, these five placements appear in 37 solutions of the full problem.)

266. No; Peter Weigel has shown that exactly $8 \cdot 89 + 2 \cdot 3 = 718$ foursomes *cannot* be completed. The three solutions with fourfold symmetry are obtained by placing a queen in (row, col) $= (1,2)$ or $(3,7)$ or $(7,8)$, then rotating by $90°$, $180°$, $270°$.

267. Sometimes the case $w = v$ is necessary. If, for example, $S_c = \{u,v\}$ and $\texttt{STAMP}(u) > \texttt{STAMP}(c)$, the change to D_u might have caused v to lose all support in c.

But we can introduce a new variable q, setting $q \leftarrow 0$ at the beginning of step D4, also setting $q \leftarrow 1$ at the beginning of D6 if $\texttt{STAMP}(w) > \texttt{STAMP}(c)$. Then step D4 needs to do step D6 for $w = v$ only if c is unary or $q = 1$ after all other choices of $w \in S_c$ have been tried. [See C. Lecoutre, *Constraint Networks* (2009), Algorithm 9.]

268. We use the following data structures for clauses c and bindings β:

$\texttt{BIND}(c)$ is the binding for which clause c lists potential supports;
$\texttt{POS}(c)$ is the $\texttt{MEM}$ location for the current support of $\texttt{BIND}(c)$;
$\texttt{IN}(\beta)$, where $\beta = (v,a)$, is 1 if a is in v's current domain, otherwise 0;
$\texttt{LAST}(\beta)$ is the final clause c such that $\texttt{MEM[POS}(c)\texttt{]} = \beta$;
$\texttt{PREV}(c)$ is the previous clause c' such that $\texttt{MEM[POS}(c')\texttt{]} = \texttt{MEM[POS}(c)\texttt{]}$;
$\texttt{START}(c)$ is the $\texttt{MEM}$ location just preceding clause c.

A stack S_0, S_1, $\ldots$ holds bindings that will soon be removed from their current domains.

H1. [Initialize.] Set $\texttt{LAST}(\beta) \leftarrow 0$ and $\texttt{IN}(\beta) \leftarrow 1$ for all bindings β. Also set $c \leftarrow l \leftarrow s \leftarrow 0$, so that the table of clauses, $\texttt{MEM}$, and the stack are initially empty. Then, for each binding $\beta = (v,a)$ and for each constraint $R(v,w)$ that involves v, generate a potential clause as follows: Let $\{b_1,\ldots,b_k\}$ be the values of w such that $ab_j \in R(v,w)$. If $k = |D_w|$, do nothing (the relation doesn't constrain β). Otherwise if $k > 0$, set $c \leftarrow c+1$, $\texttt{BIND}(c) \leftarrow \beta$, $\texttt{START}(c) \leftarrow l$, $\texttt{MEM[}l{+}j\texttt{]} \leftarrow (w,b_j)$ for $1 \le j \le k$, $l \leftarrow l+k$, $\texttt{POS}(c) \leftarrow l$, $\alpha \leftarrow \texttt{MEM[}l\texttt{]}$, $\texttt{PREV}(c) \leftarrow \texttt{LAST}(\alpha)$, and $\texttt{LAST}(\alpha) \leftarrow c$. [See (89).] Otherwise if $\texttt{IN}(\beta) = 1$, set $\texttt{IN}(\beta) \leftarrow 0$, $S_s \leftarrow \beta$, $s \leftarrow s+1$.

H2. [Prepare to loop.] Terminate the algorithm if $s = 0$ (because all bindings with $\texttt{IN}(\beta) = 1$ are supported). Otherwise set $s \leftarrow s-1$, $\beta \leftarrow S_s$, and $c \leftarrow \texttt{LAST}(\beta)$. (We need to find supports for bindings previously supported by β.)

H3. [Done with loop?] If $c = 0$, return to H2. Otherwise set $c' \leftarrow \texttt{PREV}(c)$, $\beta \leftarrow \texttt{BIND}(c)$, and let $\beta = (w,b)$. Go to H6 if $\texttt{IN}(\beta) = 0$ (because we've already deleted b from w's domain and don't need support for it). Otherwise set $k \leftarrow \texttt{POS}(c) - 1$.

H4. [Done with c?] If $k = \texttt{START}(c)$, go to H5. Otherwise set $\alpha \leftarrow \texttt{MEM[}k\texttt{]}$. If $\texttt{IN}(\alpha) = 0$, set $k \leftarrow k-1$ and repeat this step. Otherwise set $\texttt{POS}(c) \leftarrow k$, $\texttt{PREV}(c) \leftarrow \texttt{LAST}(\alpha)$, $\texttt{LAST}(\alpha) \leftarrow c$, and go to H6.

H5. [Remove binding β.] Set $\texttt{IN}(\beta) \leftarrow 0$ and remove b from the domain of w. If w's domain is now empty, terminate and report unsatisfiability. Otherwise set $S_s \leftarrow \beta$ and $s \leftarrow s+1$.

H6. [Loop on c.] Set $c \leftarrow c'$ and return to step H3. ▌

269. Consider, for example, the 4-ary relation $wxyz \in \{0101, 1210, 2110\}$, where $D_w = D_x = D_y = D_z = \{0,1,2\}$. We can set up 12 dual Horn clauses analogous to (88): $\overline{x_1y_0z_1} \Rightarrow \bar{w}_0$, $\overline{x_2y_1z_0} \Rightarrow \bar{w}_1$, $\overline{x_1y_1z_0} \Rightarrow \bar{w}_2$; $\Rightarrow \bar{x}_0$, $\overline{w_0y_0z_1} \wedge \overline{w_2y_1z_0} \Rightarrow \bar{x}_1$, $\overline{w_1y_1z_0} \Rightarrow \bar{x}_2$; $\ldots$; $\Rightarrow \bar{z}_2$; here $x_ay_bz_c$ is a "compound" Boolean variable meaning $x_a \wedge y_b \wedge z_c$. Additional clauses such as $\bar{x}_a \Rightarrow \overline{x_ay_bz_c}$, $\bar{y}_b \Rightarrow \overline{x_ay_bz_c}$, $\bar{z}_c \Rightarrow \overline{x_ay_bz_c}$, complete the set.

The algorithm of exercise 268 is extended to allow "hyperbindings" β such as $\{(x,a),(y,b),(z,c)\}$ as well as ordinary bindings, and to build the table of all necessary clauses in the extension of step H1. In general, a k-ary constraint yields $|D_v|$ support clauses whose left-hand sides involve $(k-1)$-ary compound Booleans, for each v in the scope of the relation. And there are $k-1$ clauses for each $(k-1)$-ary compound Boolean, having that compound on the right and a simple Boolean on the left.

270. Each "variable" of $\mathcal{P}^*$ is either a variable v of $\mathcal{P}$ or a constraint c of $\mathcal{P}$ (sometimes called a "hidden variable"). Each "constraint" of $\mathcal{P}^*$ is a relation between some ordinary variable v and some hidden variable c with $v \in S_c$. (Thus the graph of constraints in $\mathcal{P}^*$ is bipartite; it represents the hypergraph whose hyperedges are $\mathcal{P}$'s scopes S_c, just as the bipartite Heawood graph represents the Fano hypergraph in 7–(57).) Each domain D_v is the same in $\mathcal{P}$ and $\mathcal{P}^*$; each domain D_{c_i} is the set of c_i's k_i-tuples.

Domain consistency is especially easy to understand when all constraints are binary, because a binary constraint can be represented as a Boolean matrix. *Domain consistency holds if and only if none of the Boolean matrices has an all-zero row or column.*

Consider a 4-ary constraint such as '$wxyz \in \{0101, 0122, 1100, 1212, 2020, 2211\}$'. The Boolean matrix that relates this constraint to the ternary variable x is

$$\begin{array}{c c} & \begin{array}{cccccc} 0101 & 0122 & 1100 & 1212 & 2020 & 2211 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}.$$

Notice that each column of such a matrix contains exactly one 1. In general, if $v \in S_c$ and $a \in D_v$, the number of 1s in the row for $v=a$ in the matrix that relates v to c is the number of supports for $v{=}a$ in c. An all-zero row is equivalent to having no support.

(The construction in this exercise provides an alternative solution to exercise 269.)

271. Let the variables of $\mathcal{P}^D$ be the *hidden* variables of $\mathcal{P}$, namely $\mathcal{P}$'s constraints, where we require the tuples of hidden variable c to match the tuples of every other hidden variable c' wherever their scopes overlap. (There's a constraint between c and c' if and only if $S_c \cap S_{c'} \neq \emptyset$.)

Suppose, for example, that $\mathcal{P}$ is the CSP with four binary variables $\{w,x,y,z\}$ and the following two ternary constraints:

$$c = \text{`}w+x+y=1\text{'};\quad c' = \text{`}x+y+z=2\text{'}.$$

$$\begin{array}{c c} & \overbrace{\begin{array}{ccc} 011 & 101 & 110 \end{array}}^{xyz} \\ wxy \left\{ \begin{array}{c} 001 \\ 010 \\ 100 \end{array} \right. & \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

Then $\mathcal{P}$ is domain consistent. But $\mathcal{P}^D$ is not, because the matrix for the relation between c and c', shown at the right, has an all-zero row (and an all-zero column).

References: The dual of a CSP was defined by R. Dechter and J. Pearl [*Artificial Intelligence* **38** (1989), 353–366], who observed that many of the constraints between hidden variables are often redundant because they're consequences of others. When

dual Horn clauses
"compound" Boolean variable
hyperbindings
support clauses
hidden variable
bipartite
hypergraph
Heawood graph
Fano hypergraph
binary constraint
Boolean matrix
supports
Dechter
historical remarks–
Pearl

the unnecessary constraints are removed, we get what database theorists call a "join graph." Domain consistency of $\mathcal{P}^D$ was called *pairwise consistency* by P. Janssen, P. Jégou, B. Nouguier, and M.C. Vilarem [*IEEE International Workshop on Tools for Artificial Intelligence* **1** (1989), 420–427]. F. Bacchus, X. Chen, P. van Beek, and T. Walsh [*Artificial Intelligence* **140** (2002), 1–37] made a thorough study of local consistencies in $\mathcal{P}^*$ and $\mathcal{P}^D$.

272. The total size of all domains in (22), before reduction, is $4 \cdot 26 = 104$, compared to $4 \cdot 1 + 6 \cdot 5 + 3 \cdot 8 + 5 \cdot 5 = 83$ in exercise 43. When reducing (22) to (91), 46 domain values are immediately ruled out by having no initial supports. (For example, the Horn clause for $\overline{\text{be=<}}$ has an empty left-hand side.) Then the algorithm of exercise 269 makes 67 deductions (such as $\overline{\text{qs=<}} \Rightarrow \overline{\text{fq=+}}$) before finishing.

Filtering in the dual model goes much faster, in part because all constraints are binary. After 48 domain values are immediately ruled out, only three deductions need to be made by the Horn-core method of exercise 268. (For example, one of them is $\overline{\text{e=---}} \Rightarrow \overline{\text{b=+-+}}$.) The final domains are of size 1 for the interior junctions $\{\mathrm{d, g, h, i, j, k, l, n, p}\}$. In fact, *forward consistency* by itself gives excellent reduction.

Exterior junctions $\{\mathrm{a, q, r}\}$ of type V are left with domains of size 3; the others, $\{\mathrm{b, c, e, f, m, o, s}\}$, are left with 2-element domains. The actual line labels are represented only implicitly by the domains of this model.

273. Suppose the branch variable at the root is ab. One of the four branches is 'ab = +'. Since variable ab appears in the binary relation for junction a, FC reduces the domain of ac to {>}; hence we'll assign ac = > next. Oops: The ternary relation at junction c (namely the relation on $\{\mathrm{ac, cd, cm}\}$ should now tell us that we're in trouble; but FC won't be aware of any difficulty until either cd or cm has been assigned a value.

Another branch is 'ab = <'. That one reduces the domain of ac to {+, >}. It should get us into trouble at junction b; but no trouble will be sensed there until there's an assignment to either be or bd. (And other branches near the root fare no better.)

274. In this case the branch 'ab = +' changes D_a to {+>} and D_b to {+-+}, by FC. Hence we'll soon take the branch 'a = +>', which forces ac = >, which reduces D_c to $\emptyset$.

The branch 'ab = <' sets $D_b \leftarrow \emptyset$. The branch 'ab = -' soon forces ac = <, be = -, ..., and the complete solution at the right of (23), all via FC. Finally, the branch 'ab = >' and FC give the other three solutions, with minimal branching.

275. If we place a queen in a corner, say in cell $(1, 1)$, both of the free cells in row 3 are domain inconsistent with respect to column 3. If we place a queen near the corner, say in cell $(2, 2)$, the free cell in row 1 is domain inconsistent with respect to column 1.

[Singleton domain consistency was introduced by R. Debruyne and C. Bessière in *IJCAI* **15** (1997), 412–417; see also the implementation hints by C. Bessiere, S. Cardon, R. Debruyne, and C. Lecoutre in *Constraints* **16** (2011), 25–53. It can be very useful as a preprocessing step for difficult problems; but the cost of maintaining it during search is usually too high. With Boolean domains this idea is called "falsifying failed literals"; see also the SLUR algorithm of exercise 7.2.2.2–444.]

276. We have $jj' \in R_{ii'}$ if and only if $(i', j') = (i, j) + (1, \pm 2)k$ for some k (modulo 5). For example, $R_{24} = \{12, 15, 21, 23, 32, 34, 43, 45, 51, 54\}$. (These are the positions of pairs of queens in complete solutions to the problem. Every complete solution is equivalent to one of the two toroidal solutions; see exercise 7.2.2–12.)

[With 6 queens, path consistency is achieved after only one round of removals. With 7 or more queens, the initial constraints are path consistent.]

database theorists
join graph
pairwise consistency
Janssen
Jégou
Nouguier
Vilarem
Bacchus
Chen
Beek
Walsh
forward consistency
Debruyne
Bessière
Cardon
Lecoutre
preprocessing step
failed literals
SLUR algorithm
toroidal

277. (a) Let $q = 1 - p$. In Pass 1, r_{ij} is examined if and only if $r_{ik} = 0$ for $0 \le k < j$, hence with probability q^j. So the expected total cost is $\sum_{i=0}^{d-1}\sum_{j=0}^{d'-1} q^j = (1-q^{d'})d/p$.

Pass 2 examines r_{ij} if and only if we have (i) $r_{kj} = 0$ for $0 \le k < i$; (ii) $r_{ik} = 1$ for some $k < j$; and (iii) either $r_{k0} \ldots r_{k(j-1)} \ne 0 \ldots 0$ or $r_{k0} \ldots r_{kj} = 0 \ldots 00$, for $i < k < d$. So the probability is $q^i(1-q^j)(1-pq^j)^{d-1-i}$.

Summing this geometric series over i, we find that the total expected cost of Pass 2 is $(1-q^d)d'/p - S$, where $S = \sum_{j=0}^{d'-1}(1-pq^j)^d/p$ is the expected number of unnecessary probes made by the naïve algorithm. [This analysis was first carried out by M. R. C. van Dongen, A. B. Dieker, and A. Sapozhnikov, who also derived a complicated formula for the variance. See *Constraint Programming Letters* **2** (2008), 55–77.]

(b) Do the inner loop only for values of j with $s'_j = 0$. Then, if that loop ends with $s_i = 0$, do another loop on j, but only for values of j with $m_{ij} = 0$. (This algorithm is due to M. R. C. van Dongen; see Fig. 7.3 in his Ph.D. thesis (Cork: National Univ. of Ireland, 2002). The expected number of probes in Pass 1 remains the same; but the expected number of column supports found on that pass is increased. No simple formula is known for the expected number of probes in the subsequent Pass 2.)

[When $d' = 2$, the expected cost of this improved algorithm can be shown to equal $(1+q)d + q^{d-1} - q^{2d-1}$. And the expected cost when $d' = 3$ turns out to be $(1+q+q^2)d + q^{d-2} + 2q^{d-1} - q^{2d-3}(1+q+q^2) + q^{3d-3}(1-q^2)$.]

(c)	$p = .01$	$p = .02$	$p = .03$	$p = .04$	$p = .05$	$p = .10$	$p = .50$	$p = .90$
mean cost (naïve)	12700	8700	6300	4900	4000	2000	400	220
mean cost (a)	8100	6000	4600	3700	3100	1700	390	210
mean cost (b)	7500	5200	3700	2800	2200	1100	200	110
dev (b)	310	300	280	260	200	130	18	3

278. Marc van Dongen observes that an optimum algorithm queries r_{ij} only when either (i) both s_i and s'_j are unknown, or (ii) one of them is known but not the other. Every optimum algorithm can be assumed to make all of its type (i) queries first, because (ii) followed by (i) is never better than (i) followed by (ii).

The algorithm of exercise 277(b) treats rows and columns in dramatically different ways. Yet in October 2024, van Dongen announced that a delicate analysis proves that is indeed optimum. [To appear.]

279. (a) Every x_i has the value of some source, by induction on i. But x_n doesn't.

(b) Let R_i be primary and x_i be secondary for $1 \le i \le n$. Also let $x_{i,j}$ be secondary for $1 \le i \le m$ and $j \in \{0,1,2\}$, together with $3m$ options '$R_i\ x_i{:}j\ x_{i,j}$'. Add another primary item #, with three options '$\#\ x_n{:}j\ x_{1,j}\ \ldots\ x_{m,j}$' for $j \in \{0,1,2\}$; that takes care of the binary constraints. Finally, introduce $15(n-m)$ options '$R_i\ x_i{:}a\ x_{j(i)}{:}b\ x_{k(i)}{:}c$' for $m < i \le n$ and for all $a,b,c \in \{0,1,2\}$ with ($a = b$ or $a = c$).

(c) Define $j(i)$ and $k(i)$ in $\binom{m}{2}\binom{m+1}{2}\ldots\binom{n-1}{2} = 2^{m-n}(n-1)^{\underline{n-m}}(n-2)^{\underline{n-m}}$ ways.

(d) These problems are tough for Algorithm 7.2.2.1C; for instance, the first random example tried for $m = 24$ and $n = 64$ took 1.4 teramems. But it became much more tractable, only 31 *giga*mems, when each item R_i for $m < i \le n$ was renamed $\#R_i$, and the sharp preference heuristic of exercise 7.2.2.1–10 was used. (That trick also polished off nine other random instances, with a median run time of 1.5 *mega*mems.)

(e) To support $x_i = a$ in a binary constraint, set the other variable to $(a+1) \bmod 3$. To support $x_i = a$ in a ternary constraint, set the other two variables to a.

But after $x_n \leftarrow a$, answer 268 will remove a from the domains of x_1, x_2, ...

geometric series
van Dongen
Dieker
Sapozhnikov
variance
van Dongen
van Dongen
sharp preference heuristic

[This family of problems was introduced by J. Hwang and D. G. Mitchell, *LNCS* **3709** (2005), 343–357, who showed that with suitable choices of $j(i)$ and $k(i)$ it can be solved via backtracking only with an exponentially large search tree, if every node of that tree is a d-way branch on the value of some variable (or on the options that can cover an item), assuming that the algorithm prunes domains (or removes options) only via forward consistency. They devised a Prover–Delayer game, as in Theorem 7.2.2.2R.

On the other hand, a polynomial-size search tree *can* be constructed with *binary* branching, where every search tree node chooses either to include an option or not: For each value a tentatively assigned to x_n, try to include an option for R_i that specifies either $x_{j(i)}{:}a$ or $x_{k(i)}{:}a$, where i is as small as possible. That option leads to an immediate contradiction. So we can remove it, and continue until $x_n = a$ is contradicted.

We can obviously generalize the chain CSP by allowing *arbitrary* ternary constraints R_i for $m < i \le n$, perhaps different for each i. Many such generalizations are likely to be instructive.]

Hwang
Mitchell
backtracking
exponentially large
d-way branch
forward consistency
Prover–Delayer game
Delayer
binary branching
Janson

280. (a) Let $X_i = [x_i \text{ is a sink}]$. Then $\mathrm{E}\,X_i = \Pr(X_i = 1) = q_{\max(i,m)+1}\ldots q_n$, where $q_l = \Pr(i \notin \{j(l), k(l)\}) = \binom{l-2}{2}/\binom{l-1}{2}$. Hence $\mathrm{E}\,X_i = \binom{\max(i,m)-1}{2}/\binom{n-1}{2}$; and $S_{m,n} = \sum_{i=1}^{n} \mathrm{E}\,X_i = \frac{n}{3}(1 + 2m^3/n^3)$.

(b) Set $d \leftarrow n - m - 1$, $a_{0,0,m} \leftarrow 1$. Then for $1 \le i \le d$ and $0 \le j \le i$, set

$$a_{i,j,m} \leftarrow \binom{m+i-j-2}{2}[j \ne i]\, a_{i-1,j,m} \;+\; \left(\binom{m+i-1}{2} - \binom{m+i-j-1}{2}\right)[j \ne 0]\, a_{i-1,j-1,m}.$$

Then there are $a_{i,j,m}$ cases in which x_1 is connected to exactly j of the variables $\{x_{m+1}, \ldots, x_{m+i}\}$. Consequently the number of cases in which x_1 is *not* connected to x_n is $b_{m,n} \leftarrow \sum_{j=0}^{d} \binom{n-j-2}{2} a_{d,j,m}$; and the probability that a particular source is connected to x_n is $p_{m,n} = 1 - b_{m,n}/q_{m,n}$, where $q_{m,n}$ is the answer to exercise 279(c). Finally, $C_{m,n} = mp_{m,n} + \sum_{i=m+1}^{n} p_{i,n}$. We have $C_{24,64} \approx 8.4023$ and $C_{24,500} \approx 41.08$.

(c) Let $f(s,t) = 0$ if $s < 0$ or $t < 0$ and $f(s,t) = [s=0]$ if $s + t = m$; also $f(s,t) = \binom{t+1}{2} f(s-2,t+1) + (s-1)tf(s-1,t) + \binom{s}{2} f(s,t-1)$ when $s + t > m$. Then $f(s,t)$ is the number of cases with $s+t$ variables, m of which are sources, and t sinks. Hence $c_{m,n} = f(n-1,1)/q_{m,n}$. We have $c_{24,64} \approx 1.7522 \times 10^{-25}$.

(Incidentally, $c_{m,n} = 0$ for $n < 2m-1$; and $c_{m,2m-1} = m!(m-1)!^2(m-2)!/ ((2m-2)!(2m-3)!) = 32 \cdot 16^{-m} m^2 \pi\,(1 + O(1/m))$.)

281. Observing that $C_{m,n} \le C_{2,n} \le C_{m,n} + m$, Svante Janson has proved that $C_{m,n} \sim \frac{3}{8}\sqrt{\pi^3 n}$; and he has also obtained formulas for the higher moments. [*Random Structures & Algorithms* **64** (2024), 768–803.] On the other hand, he conjectures that $c_{m,n}$ approaches $(\rho + o(1))^n$, for some constant $\rho < 1$ that has no simple form.

282. Let $y_k = (x_{(k+1) \bmod n} - x_k) \bmod d$ for $0 \le k < n$. Then $x_0 \ldots x_{n-1}$ is a solution if and only if $(y_0 + \cdots + y_{n-1}) \bmod d = 0$. Hence the number of solutions is $d\sum_k \binom{n}{dk}$.

283. (a) We can assume that $x_0 \leftarrow 0$ is assigned first; then $x_1 \leftarrow 0$ or 1; then $x_2 \leftarrow x_1$ or x_1+1; etc. The active domains after x_j has been assigned will be $D_{j+1} = \{x_j, x_{j+1}\}$; $D_k = \{0, \ldots, d-1\}$ for $j+1 < k < n-1$; $D_{n-1} = \{0, d-1\}$. So the search tree size will be $\Omega(2^n d)$. [In fact, Algorithm 7.2.2.1C looks at exactly $2^n d - d + 1$ nodes when $d > n$, $2^n d + 1$ nodes when $d = n$, and $2^n d + n^2 - 2n + 2$ nodes when $d = n-1$.]

(b) After $x_0 \leftarrow 0$ we'll have $D_k = \{0\}$ for $0 < k < n$, if $d > n$; $D_k = \{0, k\}$, if $d = n$; and $\{0, k-1, k\}$ for $1 < k < n-1$, if $d = n-1$. The latter case is the most interesting: After setting $x_j = j-1$, we'll have $D_k = \{k-1\}$ for $j < k < n$. So there will be $O(n^2 d)$ nodes altogether. [In fact, Algorithm S looks at exactly $(n+1)d + n$ nodes when $d > n$; $2n^2 + 2n$ nodes when $d = n$; and $(n^2 + 7n - 2)d/2$ nodes when $d = n-1$.]

284. There are exactly d ways to insert 'n' into such a permutation of $\{1,\ldots,n-1\}$, namely at the beginning or after one of $\{n-1,n-2,\ldots,n-d+1\}$. So the answer is $d!\,d^{n-d}$, by induction. [The number of permutations with exactly k instances of $p_{j+1} \ge p_j + d$ was investigated by J. Riordan in the final chapter of *An Introduction to Combinatorial Analysis* (1958), thereby generalizing the Eulerian numbers. See OEIS A120434 for the case $d = 2$.]

285. $p_j = n$ if and only if $j \ge 1$ is minimum with $j \notin S$. Remove $p_1 \ldots p_j$ and recurse.

[Richard Stanley, in *Enumerative Combinatorics* **1**, second edition (2012), exercise 1.114(b), discovered another interesting family of permutations with this uniqueness property, namely those $p_1 \ldots p_n$ such that $\{p_1,\ldots,p_k\}$ is an *interval*, for $1 \le k \le n$; a typical example for $n = 7$ is 4325617. Such permutations are readily generated via backtracking, but not so easy to set up as a CSP; the condition is that if $i < j < k$ we don't have $p_i < p_k < p_j$ or $p_j < p_k < p_i$, In other words, they're '$(132,312)$-avoiding'. Their left-to-right reversals, the $(213,231)$-avoiding permutations, are precisely those that produce degenerate binary search trees; see exercise 6.2.2–5(b).]

286. True. In fact, the set S corresponding to the inverse is $S^R = \{n-j \mid j \in S\}$.

287. Let p_j and q_j be primary items for $1 \le j \le n$; also introduce $(n-1)(n-2)$ secondary items $j.k$ for $1 \le j < n$ and $1 \le k < m = n-1$, which correspond to items y_k of the pairwise encoding trick that enforces $p_{j+1} \le p_j + 1$. For each $1 \le j,k \le n$ there's an option containing p_j, q_k, and perhaps other items: If $j > 1$ and $k > 2$, set $t \leftarrow k-2$; then while $t > 0$ include $(j-1).t$ and set $t \leftarrow t \mathbin{\&} (t-1)$. (This contributes α_{k-2}.) If $j < n$ and $k \le m$, set $t \leftarrow -k$; then while $t > -m$ include $j.(-t)$ and set $t \leftarrow t \mathbin{\&} (t-1)$. (This contributes β_{k-1}.) For example, the "diagonal" options when $n = 5$ are 'p_1 q_1 1.1 1.2', 'p_2 q_2 2.2', 'p_3 q_3 2.1 3.3', 'p_4 q_4 3.2', 'p_5 q_5 4.3 4.2'.

288. We can also require $q_{k+1} \le q_k + 1$: Introduce new secondary items $j{,}k$ for $1 \le j < n$ and $1 \le k < m$. Whenever answer 287 put $r.s$ in the option for p_j and q_k, also put $r{,}s$ in the option for p_k and q_j. (Thus one option for $n = 5$ is 'p_2 q_3 1.1 2.3 3,2'.)

289.

2443	2177	5144	2141	2143	2144
2433	2772	5544	2442	2443	2644
1422 ;	1372 ;	1552 ;	1412 ;	1413 ;	1662 .
3331	3377	3332	3331	3331	6662
(1 of 3)	(1 of 32)	(1 of 3)	(1 of 1)	(1 of 12)	(1 of 24)

[*Historical note:* Fillomino was invented by Waku Sakinaga; see *Puzzle Communication Nikoli* **47** (February 1994).]

290. If $t_n = t_n(1) + t_n(2) + \cdots$ is the desired number, where $t_n(m)$ is the number of patterns with m in the upper right corner, we have the recurrences $t_n(m) = a_n(m,m) + \sum_{m'=1}^{2n-m} b_n(m,m;m',m')$; $a_n(l,m) = \bigl(l < 2?\ 0\colon l > 2n?\ 0\colon l = 2n?\ 1\colon l = 2?\ t_{n-1} - 2t_{n-1}(m) + a_{n-1}(m,m)\colon a_{n-1}(l-2,m) + 2\sum_{m'=1}^{2n-l} b_{n-1}(l-2,m;m',m')\bigr)$; $b_n(l,m;l',m') = \bigl(m = m'?\ 0\colon l < 1?\ 0\colon l' < 1?\ 0\colon l+l' > 2n?\ 0\colon l+l' = 2n?\ 1\colon l = l'?\ t_{n-l} - t_{n-l}(m) - t_{n-l}(m') + b_{n-l}(m,m;m',m')\colon l < l'?\ a_{n-l}(l'-l,m') + \sum_{m''=1}^{2n-l-l'} b_{n-l}(m'',m'';l'-l,m')[m'' \ne m]\colon b_n(l',m';l,m)\bigr)$. Here $a_n(l,m)$ is the number of length n prefixes of $2\times\infty$ fillomino patterns that end with two m's at the right, where those m's are part of an l-omino; $b_n(l,m;l',m')$ is similar, but ending with $m \ne m'$ at the right, respectively parts of an l-omino and an l'-omino. Hence $(t_1,t_2,\ldots,t_{145}) = (1$, 5, 33, 138, 715, 3524, …, 51376 52565 68766 30928 69800 54061 86098 15559 89493 34784 20112 85272 12992 22603 93822 34860 83493 24519 70607 50508). (The ratio t_{n+1}/t_n converges rapidly to 4.91867 12250 37424 13083 06703 91572 28440 ….)

Riordan
Eulerian numbers
OEIS
Stanley
interval
backtracking
modeling as CSP
patterns in permutations
$(132,312)$-avoiding
reversals
$(213,231)$-avoiding
degenerate
binary search trees
Historical note
Sakinaga
recurrences

author
downloadable programs
Green
Saarinen
@kobouzu17
symmetry

291. Suppose the given shape S has N cells. There are N primary items ij, one for each cell. (If S is an $m \times n$ grid, for instance, we have $N = mn$ and $0 \le i < m$, $0 \le j < n$.) A *potential d-omino* is a set $P \subseteq S$ of rookwise connected cells for which every $ij \in P$ is either blank or labeled d, but not adjacent to any cell $\notin P$ that's labeled d. (All such P can be found by using an interesting variant of the algorithm in exercise 7.2.2–75; see, for example, the author's online program FILLOMINO-DLX.) There are secondary items e_d, one for each edge between two unlabeled cells and each possible d. And there's one option for each potential d-omino P, containing (i) all $ij \in P$ and (ii) all e_d for which e is an edge between a blank cell $\in P$ and a blank cell $\notin P$.

For example, the puzzle of exercise 289 has $m = n = 4$, $N = 16$, and exactly (8, 5, 11, 11) potential (1, 2, 3, 4)-ominoes, hence 35 options. Two of the potential tetromino options are '02 03 12 13 h_{224} v_{124}' and '11 21 22 32 h_{224} v_{124} v_{334}', where h_{ij} and v_{ij} denote the horizontal and vertical edges that connect $(i{-}1)j$ and $i(j{-}1)$ with ij.

How large can d be? Suppose c_d of the given cells are labeled d, for a total of $C = c_1 + c_2 + \cdots + c_s$ "clues," where s is the maximum label. Then every potential d-omino has $d \le \max(N - C, s)$. And there's a sharper bound $\max(N - C^+, s)$, where $C^+ = \sum_{d=1}^{s} c_d^+$ and c_d^+ is a lower bound on the number of d labels that are known to exist. For example, we may take $c_1^+ = c_1$; $c_2^+ = 2(c_2 -$ the number of pairs of adjacent 2s); and for $d \ge 3$, $c_d^+ = d\lceil c_d/d \rceil + [c_d \bmod d = 0$ and the ds aren't disjoint d-ominoes$]$.

292.

(a)
```
221221
1ee312
e1e332
eeeeee
233e1e
213ee1
122122
```
; (b)
```
335555666
314159266
444999236
555899932
535897932
338877773
888772643
338462643
344466644
```
; (c)
```
6244221244
6244888244
6688866688
2684446888
2684226882
6448446482
6248441448
6248862248
6688666688
6882268888
```
; (d)
```
1334141224
2344i44434
2143i22334
1223i33i14
21i3i3ii22
2iiiiii444
34444i3324
33221i3421
1444ii4433
3334224223
```
; (e)
```
2244446633
5153366313
5553616345
4444646345
5566643145
3565543345
3552542215
3442513334
2344544414
2332242244
```
.

[Puzzle (c), by Chris Green, was posted at `puzzleparade.blogspot.com` (19 July 2013), #60; puzzle (d), which totally defeats the construction in answer 291 because it requires a humongous number of options, was posted at `gmpuzzles.com` by Tapio Saarinen (7 October 2014); puzzle (e) was posted on Twitter by `@kobouzu17` (31 December 2022).

293. We save a factor of roughly 8 by removing symmetry. Each potential puzzle with k clues leads to k potential puzzles with one fewer clue, until we reach invalid cases that are matched by more than one of the given 59951. Potential puzzles without redundancies must still be screened to ensure that they can't be solved with labels greater than 5.

All told we obtain 938484 nonisomorphic minimal 4×4 puzzles whose clues don't exceed 5, of which (937236, 1240, 8, 0) have (1, 2, 4, 8)-fold symmetry. Exactly (1124, 56253, 374643, 377611, 104436, 20410, 3520, 430, 57) of them have (4, 5, ..., 12) clues.

`4␣␣␣`	`4␣␣␣`	`4␣␣␣`	`4␣␣␣`	`␣␣␣1`	`␣␣␣3`	`␣␣5␣`	`1␣33`	`␣1␣1`	`␣3␣2`	`␣␣2␣`	`␣␣41`
`␣␣␣4`	`␣␣␣4`	`␣␣␣4`	`␣␣␣4`	`␣15␣`	`␣␣3␣`	`␣5␣5`	`55␣␣`	`2␣4␣`	`4␣4␣`	`3␣␣1`	`2␣␣␣`
`1␣␣␣`	`1␣␣␣`	`2␣␣␣`	`2␣␣␣`	`␣51␣`	`␣3␣␣`	`5␣5␣`	`␣544`	`␣5␣4`	`␣2␣4`	`␣␣␣4`	`␣3␣␣`
`␣␣␣3`	`␣␣␣2`	`␣␣␣2`	`␣␣␣5`	`1␣␣␣`	`3␣3␣`	`5␣5␣`	`5544`	`5␣5␣`	`4␣4␣`	`␣2␣␣`	`3␣␣2`
(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)	(xi)	(xii)

Fig. A–16. A gallery of interesting 4×4 fillomino puzzles.

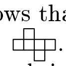
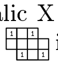
Puzzles (i)–(iv) in Fig. A–16, which have just 4 clues each, make a nice sequence by which we can introduce newbies to the wonders of fillomino. One of the cutest examples with 4-fold symmetry is puzzle (v). And (vi) and (vii) are among the 15 with "pure" clues (all the same). Puzzle (viii) is interesting not because it's hard to solve, but because all twelve of its clues are necessary. Similarly, none of the eight clues in (ix) and (x), which appear in the cells of odd parity like a checkerboard, are redundant. The most difficult 4×4 fillomino puzzles, rated by the search tree size (16) when full domain consistency is maintained, are probably those in (xi) and (xii). (See Appendix E.)

294. (Solution by N. Beluhov.) Let P be a maximal rookwise connected subset of the solution, having no labels $\le s$. Every element of P must have the same label $d = |P|$, because the solution is unique. Let T be a spanning tree of P. Every edge $u \mathrel{—} v$ of T partitions T, hence P, into two polyominoes $P_u \cup P_v$ when that edge is removed from T, where $u \in P_u$ and $v \in P_v$. By uniqueness, we cannot have $s < |P_u| < |P_v|$.

Case 1, T has one centroid, v. When T is rooted at v, let v's children be u_1, …, u_k, with corresponding subtree sizes $s_1 \ge \cdots \ge s_k$. Then $s_1 \le s_2 + \cdots + s_k$ by Eq. 2.3.4.4–(7). Hence we have $|P_v| = s_2 + \cdots + s_k + 1 > |P_{u_1}| = s_1$ in the decomposition $P = P_{u_1} \cup P_v$. It follows that $s_1 \le s$; and $d \le ks + 1 \le 4s + 1$.

Case 2, T has two centroids, $u \mathrel{—} v$. We may suppose that $u = i(j{-}1)$ and $v = ij$, coordinatewise. If T also contains both of the edges $u' = (i{-}1)(j{-}1) \mathrel{—} u$ and $v' = (i{-}1)j \mathrel{—} v$ (or, similarly, if T contains both $u' = (i{+}1)(j{-}1) \mathrel{—} u$ and $v' = (i{+}1)j \mathrel{—} v$), we get a decomposition $P = P_{u'} \cup P_{v'} \cup P_{uv}$ by deleting those edges, where $|P_{u'}| \le s$ and $|P_{v'}| \le s$. On the other hand, if in the original tree T we delete only edge $u \mathrel{—} v$ and replace it with edge $u' \mathrel{—} v'$, we get a new tree for which u' and v' are the two centroids, as well as a new polyomino $P_{u'v'} = P_{u'} \cup P_{v'}$. By symmetry between $u \mathrel{—} v$ and $u' \mathrel{—} v'$, therefore, $d = |P_{uv}| + |P_{u'v'}| \le 2|P_{u'v'}| \le 2|P_{u'}| + 2|P_{v'}| \le 4s$.

Finally, if T doesn't contain such u' and v', we can regard u and v as co-roots of T; and their subtrees (at most four total) must each have size $\le s$. Hence $d \le 4s+2$.

This proof shows that we can obtain $d = 4s + 2$ for $s = 1$ only when the P is the "italic X hexomino" . That's impossible if the overall shape is a rectangular grid; but  is a valid puzzle in a grid minus two corners, and $d = 5$ is possible in a 3×3 grid.

Here's $s = 2$:

␣␣␣1␣1␣␣
␣1␣2␣2␣␣
1␣␣␣␣␣␣1 ;
␣␣2␣2␣1␣
␣␣1␣1␣␣␣

and here's $s \ge 4$, shown (with many redundant clues, for clarity) for $s = 5$:

14␣␣␣1␣5␣␣34
43212␣␣54␣2␣
␣␣␣3␣␣␣5␣31␣
␣␣4␣␣␣␣5␣␣2␣
␣55555␣5␣␣␣1
␣␣␣␣␣␣␣␣␣␣␣␣ .
1␣␣␣5␣55555␣
␣2␣␣5␣␣␣␣4␣␣
␣13␣5␣␣␣3␣␣␣
␣2␣45␣␣21234
43␣␣5␣1␣␣␣41

for $s = 3$, see exercise 292(a);

295. (Solution by N. Beluhov and P. Mebane.) When $m = 1$ they are ␣ and

$$1^a\,{\sqcup}^{b_1}\,1\,{\sqcup}^{b_2}\,1 \ldots 1\,{\sqcup}^{b_r}\,1^c \quad \text{for } r \ge 1,\ 0 \le a, c \le 1,\ 2 \le b_j \le 4,\ (1{-}a)b_1 \le 2,\ (1{-}c)b_r \le 2.$$

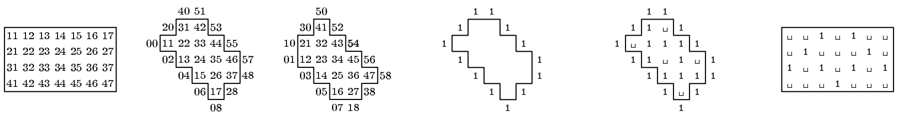
When $m = 2$ they're $\alpha\genfrac{}{}{0pt}{}{\sqcup}{\sqcup}\beta_1\genfrac{}{}{0pt}{}{\sqcup}{\sqcup}\ldots\genfrac{}{}{0pt}{}{\sqcup}{\sqcup}\beta_r\genfrac{}{}{0pt}{}{\sqcup}{\sqcup}\gamma$ for $r \ge 0$, $\alpha, \gamma \in \{\epsilon, \genfrac{}{}{0pt}{}{1}{\sqcup}, \genfrac{}{}{0pt}{}{\sqcup}{1}\}$, $\beta_j \in \{\genfrac{}{}{0pt}{}{1\sqcup}{\sqcup 1}, \genfrac{}{}{0pt}{}{\sqcup 1}{1\sqcup}\}$.

Now suppose $3 \le m \le n$. By answer 294 with $s = 1$, this problem is equivalent to tiling an $m \times n$ rectangle with monominoes and X pentominoes, together with tee tetrominoes at the edges and bent trominoes at the corners.

Notice that the 1s all occur in cells ij with the same parity $(i{+}j) \bmod 2$. Therefore we can transform the $m\times n$ grid into two "Aztec rectangles," rotating 45° and mapping $ij \mapsto (m - i + j + \delta, i + j - \delta)/2$ for $1 \le i \le m$, $1 \le j \le n$, and $\delta \in \{0, 1\}$. If we put a border around the Aztec rectangles by appending $m + n + 2$ images of ij for $i = 0$

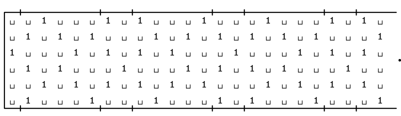
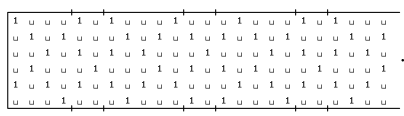
parity
checkerboard
domain consistency
Beluhov
unique
spanning tree
centroid of a free tree
hexomino
Beluhov
Mebane
monominoes
X pentominoes
pentominoes
tee tetrominoes
tetrominoes
bent trominoes
trominoes
parity
Aztec rectangles
rotating 45°

or $i = m+1$ or $j = 0$ or $j = n+1$, and if we fill those border cells with `1`s, we get an equivalent problem that's easy to visualize: (i) There must be no 2×2 array of `1`s; (ii) no two ␣s can be kingwise adjacent. For example, if $m = 4$, $n = 7$, and $\delta = 0$ we have



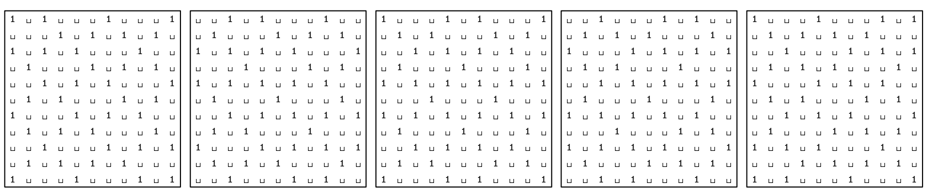
matrix 4×7 | even Aztec 4×7 | odd Aztec 4×7 | even Aztec 4×7 problem | even Aztec 4×7 solution | even matrix 4×7 solution

The two Aztec rectangles are isomorphic when mn is even. The solutions when m is even and $\delta = 0$ are the first n columns of A_m when $n \bmod (m+1) \in \{1, m\}$, and of B_m when $n \bmod (m+1) \in \{m-2, m\}$, where A_m and B_m are infinite matrices

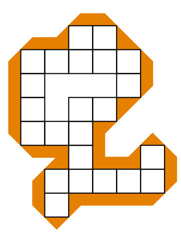
$A_6 =$  $\ldots,\ B_6 =$  $\ldots$

whose columns have period length $2m+2$, illustrated here for $m = 6$.

Solutions exist for odd m only when n is odd and $\delta = 0$. Two easy constructions always work: The even-numbered rows are `␣1␣1␣1␣1␣1...`, and the odd-numbered rows alternate between `1␣␣␣1␣␣␣1␣␣␣1...` and `␣␣1␣␣␣1␣␣␣1␣␣....` Besides those two, $(n-1)/2$ additional solutions also arise when $m = n$ is odd, such as the following:



296. Given a polyomino P with $d = d(P)$ cells, consider the path that goes through the centers of all cells that lie just outside of P's perimeter (its outer boundary). The region between this path and the perimeter is called P's "halo," and we shall call its area $h = h(P)$. The formula $h = (p - v)/2$ is not difficult to prove, where p is the perimeter's length and v is the number of maximal vertical line segments that it has. (A 23-omino and its halo are pictured.) Furthermore, N. Beluhov and S. Gerdjikov [*Középiskolai Matematikai és Fizikai Lapok [KöMaL]* (3) **70**, 7 (October 2020), 419, problem A. 783] have shown that $h \le t/2$ implies $d \le b_t = \lfloor (t^2+4)/8 \rfloor$. This bound is in fact sharp for all $t \ge 4$, with the maximum d attained by a $\lfloor t/2\rfloor \times \lceil t/2\rceil$ Aztec rectangle with $\delta = 0$ (see exercise 295).



$h = \frac{36-11}{2}$

Polyominoes without common edges have disjoint halos. Therefore if an $n \times n$ fillomino pattern Φ_n contains k d-ominoes, we must have $k(d + h_d) \le n^2 + O(n)$, where $h_d = \lceil \sqrt{8d-4}\,\rceil/2$ is the smallest possible halo area of a d-omino. Consequently $\#_d(\Phi_n)/n^2 \le d/(d+h_d) + O(1/n)$, and we have $\delta_d \le d/(d+h_d)$. These upper bounds are:

$d =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$\frac{d}{d+h_d} =$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{6}{11}$	$\frac{4}{7}$	$\frac{5}{8}$	$\frac{12}{19}$	$\frac{7}{11}$	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{20}{29}$	$\frac{11}{16}$	$\frac{12}{17}$	$\frac{13}{18}$	$\frac{28}{39}$	$\frac{30}{41}$	$\frac{8}{11}$	$\frac{17}{23}$	$\frac{3}{4}$	$\frac{38}{51}$	$\frac{40}{53}$	$\frac{42}{55}$	$\frac{22}{29}$	$\frac{23}{30}$	$\frac{24}{31}$	$\frac{25}{32}$

kingwise adjacent
halo
Beluhov
Gerdjikov
Aztec rectangle
Toroidal

Toroidal constructions (with appropriate offsets) give lower bounds for small d:

```
(d=1)  15     (d=2)  122    (d=3)   322    (d=4)  144    (d=9)  199       (d=11)    bbb1
       555 ;         244 ;          331 ;         4144 ;        9999922 ;         bbbbb333 ;
       151           244           322            4413          2299                bbb1
                                   331            1433
```

```
(d=14)  eeee1        (d=16)   ggg1          (d=19) jjjjj2        ;  (d=22)  mmmmm2       .
        eeeeee4444 ;         ggggg5                 jjjj2                    mmmm2
        eeee1                ggggg5555 ;            jjjjj1                   mmmmm7
                              ggg1                   jjjjj55555               mmmm7
                                                                              mmmm77777
```

(These constructions for $d \in \{14, 19, 22\}$ are due to Filip Stappers, and Erik Demaine contributed $d = 3$. See `https://erich-friedman.github.io/mathmagic/0316.html` for generalizations.) The efficient packing of Aztec rectangles and their halos also gives us the lower bounds $\delta_d \ge d/(b_{2k-1}+k)$ when $1 < b_{2k-2} < d \le b_{2k-1}$; $\delta_d \ge d/(b_{2k}+k)$ when $b_{2k-1} < d \le b_{2k}$; these bounds are in fact optimum when $d = b_{2k}$. So we have the following partial results:

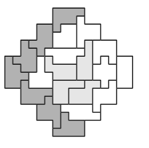
$d =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$\delta_d \ge$	$\frac{3}{8}$	$\frac{4}{9}$	$\frac{1}{2}$	$\frac{8}{15}$	$\frac{5}{8}$	$\frac{3}{5}$	$\frac{7}{12}$	$\frac{2}{3}$	$\frac{9}{14}$	$\frac{2}{3}$	$\frac{11}{16}$	$\frac{2}{3}$	$\frac{13}{18}$	$\frac{7}{10}$	$\frac{5}{7}$	$\frac{16}{23}$	$\frac{17}{24}$	$\frac{3}{4}$	$\frac{19}{27}$	$\frac{5}{7}$	$\frac{3}{4}$	$\frac{22}{31}$	$\frac{23}{32}$	$\frac{3}{4}$	$\frac{25}{32}$

Are these the true values of δ_d, or can some of them be increased?

```
␣1␣75␣␣␣␣
␣␣␣3␣␣␣41
␣␣␣␣␣15␣␣
57␣␣␣2␣␣␣
832␣9␣␣7␣
␣␣␣␣␣32␣␣
␣␣4␣2␣35␣
1␣␣465␣␣7
␣8␣␣␣␣␣␣␣
```

297. Thomas Snyder invented this challenging problem and posted his ingenious construction online at Grandmaster Puzzles (23 May 2014), remarking that he needed "quite a bit of exploration around a good seed." The second known example, shown here, was found by Germán González-Morris in 2024. (See Appendix E for solutions.)

298. (a) 9; 4; 7. (See exercise 7.2.2.1–386. Adding a domino to a 2×3 hexomino gives $(0, 1, 1, 1, 4)$ octominoes with respectively $(180°$, biaxial, axial, diagonal, no) symmetry; adding two nonadjacent monominoes gives $(2, 1, 3, 0, 7)$.)



(b) Using answer 7.2.2.1–266, piece is, for example, `0[1-4] 1[0-3]`. Break symmetry by restricting piece , say, to 1/8 of its options. Among the $(8 \cdot 16928)$ packings, connectedness is obtained uniquely(!) as shown. [This pleasant puzzle was introduced by R. Kurchan in 2022.]

299. (a) That suggestion would make the current stamp equal to (1, 2, 1, 2, 3, 2, 1, 2, 3, 2, 3, 2, 1, 0) when σ in Fig. 117 has the respective values (1, 2, ..., 14). Therefore entries such as '${}^{y}_{1}$' would improperly be omitted from the trail in lines 11 and 16. And the omission of '${}^{y}_{5}$' on line 29 would give the incorrect value $y = 4$ on line 30.

(b) If we associate a fresh value of σ to each node, the "correct" current stamps corresponding to (1, ..., 14) are then (1, 2, 1, 3, 4, 3, 1, 5, 6, 5, 7, 5, 1, 0). To obtain this behavior, place *both* x and x' on the trail when x changes, so that both x and `STAMP`(x) are restored when backtracking. Backtracking should also restore σ to its previous state. For example, the trail at line 06 would be '$|_0\, {}^{x}_{0,0}\, {}^{y}_{0,0}\, |_1\, {}^{y}_{1,1}$'. At line 29 it would be '$|_0\, {}^{x}_{0,0}\, {}^{y}_{0,0}\, |_5\, {}^{y}_{1,1}\, {}^{x}_{8,1}\, |_7\, {}^{y}_{5,5}$'. (That's $1+3+3+1+3+3+1+3 = 18$ entries instead of $1+2+2+2+2+1+2+2+2+1+2 = 19$; not a huge win in this case.)

300. This property is invariant because it is true initially and unchanged by deletion. [C. Lecoutre and R. Szymanek used it when iterating over all tuples of a relation that belong to the current domains; see *LNCS* **4204** (2006), 284–298.]

301. (a) `BITS`$(v) = 2^d - 1$. (b) `NEXT`$_v(a) = a+1$ and `IN`$_v$`[`a`]` $= 1$ and `PREV`$_v(a+1) = a$, for $0 \le a < d$; `NEXT`$_v(d)$ = `IN`$_v$`[`d`]` $= 0$; `PREV`$_v(0) = d$. (c) `DOM`$_v$`[`k`]` = `IDOM`$_v$`[`k`]` $= k$, for $0 \le k < d$; `SIZE`$(v) = d$.

Stappers
Demaine
Friedman
Aztec rectangles
Snyder
González-Morris
biaxial
axial
Break symmetry
unique
Kurchan
Lecoutre
Szymanek
tuples

302. (a) $\rho(\texttt{BITS}(v)+2^d)$. (b) $\texttt{NEXT}(d)$. (c) Initialize $\texttt{MIN}(v)$ to 0. If deleting $a = \texttt{MIN}(v)$ in (110), also do this: Set $t \leftarrow \texttt{MIN}(v)+1$, $\texttt{MIN}(v) \leftarrow d$. For $0 \le k < \texttt{SIZE}(v)$, if $\texttt{DOM}_v\texttt{[}k\texttt{]} < \texttt{MIN}(v)$, set $\texttt{MIN}(v) \leftarrow \texttt{DOM}_v\texttt{[}k\texttt{]}$, and break out of this loop if $\texttt{MIN}(v) = t$.

303. True. (See Eq. 4.1–(5).)

304. This algorithm uses an approach similar to Quicksort (Algorithm 5.2.2Q) to exchange elements of $\texttt{D}$ that are out of place. It *doesn't* change b_k when b_k should have become zero according to (111), because such words b_k will never be fetched. The operation "trail(x)" means "push the pair (address of x, value of x) onto the trail."

R1. [Initialize.] Set $i \leftarrow 0$, $s \leftarrow \texttt{S}$, $j \leftarrow s-1$.

R2. [Done?] (At this point all cases k for $k < i$ and $k > j$ are done.) If $i > j$, go to R7.

R3. [Try $k = \texttt{D[}i\texttt{]}$.] Set $k \leftarrow \texttt{D[}i\texttt{]}$ and $x \leftarrow b_k \mathbin{\&} b'_k$. If $x = 0$, go to R4. Otherwise, if $x \ne b_k$, trail(b_k) and set $b_k \leftarrow x$. Set $i \leftarrow i+1$ and return to R2.

R4. [Done?] (We know that $b_{\texttt{D[}i\texttt{]}}$ should be zero.) If $i = j$, go to R7.

R5. [Try $k = \texttt{D[}j\texttt{]}$.] Set $k \leftarrow \texttt{D[}j\texttt{]}$ and $x \leftarrow b_k \mathbin{\&} b'_k$. If $x = 0$, set $j \leftarrow j-1$ and return to R4. Otherwise, if $x \ne b_k$, trail(b_k) and set $b_k \leftarrow x$.

R6. [Swap.] Set $\texttt{D[}i\texttt{]} \leftrightarrow \texttt{D[}j\texttt{]}$, $i \leftarrow i+1$, $j \leftarrow j-1$, and return to R2.

R7. [Terminate.] If $\texttt{S} \ne i$, trail($\texttt{S}$) and set $\texttt{S} \leftarrow i$. ▮

305. Let the tuples of R be τ_i for $0 \le i < t$. And for $1 \le j \le k$, $0 \le a < d$, let $r[j,a]$ be the bitset whose ith bit is $[v_j = a \text{ in } \tau_i]$. (Thus, if R is the relation (78), we can let $(\tau_0, \ldots, \tau_6) = (000, 001, 010, 012, 020, 121, 211)$, with $t = 7$; then $r[1,0] = 1111100$, $r[1,1] = 0000010$, $r[1,2] = 0000001$, …, $r[3,1] = 0100011$, $r[3,2] = 0001000$.)

First we need to remove invalid tuples from the current R. For each j with $\texttt{OSIZE}_j \ne \texttt{SIZE}_j$, the values $\texttt{DOM}_j\texttt{[}k\texttt{]}$ for $\texttt{SIZE}_j \le k < \texttt{OSIZE}_j$ have been deleted from D_j since R's last propagation. So we intersect b with b', where b' is either (i) $\sim\mathrm{OR}\{r[j,\texttt{DOM}_j\texttt{[}k\texttt{]}] \mid \texttt{SIZE}_j \le k < \texttt{OSIZE}_j\}$ or (ii) $\mathrm{OR}\{r[j,\texttt{DOM}_j\texttt{[}k\texttt{]}] \mid 0 \le k < \texttt{SIZE}_j\}$; use (i) if $\texttt{OSIZE}_j - \texttt{SIZE}_j < \texttt{SIZE}_j$, otherwise use (ii). This bitwise OR is computed by looking only at the $\texttt{S}$ words of $r[j,a]$ whose indices appear in the first $\texttt{S}$ locations of $\texttt{D}$.

Then we need to filter the domains. For each j with $\texttt{SIZE}_j > 1$, and each $a = \texttt{DOM}_j\texttt{[}k\texttt{]}$ for $0 \le k < \texttt{SIZE}_j$, we remove a from D_j if (v_j, a) has no current support in R. That's easily tested by running through the nonzero words of b, using the first $\texttt{S}$ entries of $\texttt{D}$, and ANDing them with the words of $r[j,a]$, until we find a nonzero word (or not).

The filtering operation goes faster if we keep a table of residual supports $s[j,a]$, where $s[j,a]$ is the index of the word where a support was previously found. A loop through b is needed only if word $s[j,a]$ of $b \wedge r[j,a]$ is zero.

If filtering makes a domain empty, we backtrack (having found a contradiction).

[Compact tables can be made considerably *more* compact by extending these algorithms to allow such things as tuples with wildcards (e.g., 01∗∗2∗) or even allowing ZDD-like specifications. For a survey of these developments, see the Ph.D. thesis of Hélène Verhaeghe (University of Louvain, 2021), viii + 169 pages.]

306. (a)

$t =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
$l_t =$	0	1	2	1	0	1	2	3	2	1	2	3	4	3	4	5	4	3	2	3	2	1	2	3	2	3	4	3	2	1
$s_t =$	0	0	0	0	0	1	1	0	1	1	2	2	0	2	4	0	4	2	2	3	2	1	2	2	2	3	2	3	2	1

(b) Initially $p_0 \leftarrow 0$, $q_0 \leftarrow \infty$, $r \leftarrow 0$. For $t = 0, 1, \ldots,$ do this: Set $k \leftarrow r$ and, while $l_t < p_k$ or $l_t > q_k$, set $k \leftarrow k-1$; then $s_t \leftarrow p_k$. If $l_{t+1} < l_t$ (a backward step), update the intervals as follows: If $l_{t-1} < l_t$ (a "valley"), first set $r \leftarrow r+1$ and $q_r \leftarrow l_t$; then set $p_r \leftarrow l_t$; then if $p_{r-1} = p_r$, set $r \leftarrow r-1$ and, if $q_{r+1} > q_r$, also set $q_r \leftarrow q_{r+1}$.

ruler function
Quicksort
trail(x)
bitwise OR
bitwise AND
residual supports
wildcards
ZDD
Verhaeghe
valley

For example, after finding $s_7 = 0$ in (a), the current intervals are updated to $[0\,..\,\infty]$, $[1\,..\,2]$, $[3\,..\,3]$ because $t = 7$ is a valley at level 3. They're next updated to $[0\,..\,\infty]$, $[1\,..\,2]$, $[2\,..\,3]$. When eventually $t = 25$ they're $[0\,..\,\infty]$, $[1\,..\,2]$, $[2\,..\,5]$, $[3\,..\,3]$.

(c) The shortest such sequence goes from 0 down to 8, then up to 1, down to 6, up to 3, down to 5, up to 4, down to 15, up to 9, down to 12, up to 10, down to 14, up to 13 at time $8+7+5+3+\cdots+1 = 53 = 2\sum_{j=1}^{r}(q_j - p_j + 1) + p_r - 1$. (In general the shortest goes from 0 down to q_1, then up to $p_1 - 1$, …, down to q_r, up to $p_r - 1$.)

(d)

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
	$t =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
either	$l_t =$	0	1	2	3	4	5	4	5	6	5	6	7	6	7	8	7	8	7	8	7	8	7	8	7	6	5	4	3	2	1
	$s_t =$	0	0	0	0	0	0	0	5	0	5	6	0	6	7	0	7	8	7	8	7	8	7	8	7	6	5	0	0	0	0
or	$l_t =$	0	1	2	3	4	5	6	5	6	7	6	7	8	7	8	7	8	7	8	7	8	7	8	7	6	5	4	3	2	1
	$s_t =$	0	0	0	0	0	0	0	0	6	0	6	7	0	7	8	7	8	7	8	7	8	7	8	7	6	0	0	0	0	0

(e) $389533569/9694845 \approx 40.2$. [In general, if we consider $X = \sum_{t=0}^{2m-1} s_t$ over all $\binom{2m}{m}/(m+1)$ level sequences that have m forward steps and m backward steps, empirical results for small m suggest that $\max X \approx .54m^2$.]

(f) Empirically, the average of $l_t - s_t$ for $m = 10^6$ is only about 6.8, although the standard deviation turns out to be about 52; and the empirical average value of r at step t is, incidentally, 441 ± 200. By exercise 2.3.4.5–5, the average of l_t is exactly $\bigl((m+1)4^m - (2m+1)\binom{2m}{m}\bigr)/\bigl(2m\binom{2m}{m}\bigr) = \frac{1}{2}\sqrt{\pi m} + O(1)$; this is ≈ 885 when $m = 1000000$. So backmarking can be expected to save considerable recomputation. [Can the asymptotics of $\sum(l_t - s_t)$ and $\sum r_t$ as $m \to \infty$ be determined analytically?]

(g) Initially $M_{ja} = 0$ for all j and a. We let $a_j = 0$ before x_j is assigned.

G1. [Begin step t.] Set $j \leftarrow l_t + 1$ and $a \leftarrow a_j$. (We want to assign a new value to x_j.)

G2. [Advance a.] Set $a \leftarrow a + 1$. Go to G7 if $a > d_j$. Otherwise repeat this step if $M_{ja} < s_t$. (We've already seen that $x_j = a$ isn't consistent with the currently assigned values $\{x_i = a_i \mid 1 \le i \le M_{ja}\}$.)

G3. [Begin further tests.] Set $k \leftarrow M_{ja}$.

G4. [Check relations with x_k.] If $(a_1, \ldots, a_k, a)$ doesn't satisfy all constraints between $(x_1, \ldots, x_k)$ and x_j that involve x_k, set $M_{ja} \leftarrow k$ and return to G2. (If $k = 0$, we simply test the unary constraint on x_j.)

G5. [Loop on k.] If $k < l_t$, set $k \leftarrow k + 1$ and return to G4.

G6. [Finish forward step.] Set $M_{ja} \leftarrow j$, $a_j \leftarrow a$, $l_{t+1} \leftarrow l_t + 1$; step t is done.

G7. [Finish backward step.] Set $a_j \leftarrow 0$ and $l_{t+1} \leftarrow l_t - 1$; step t is done. ▌

(Here s_t is evaluated as in (b). Of course the values of l_t and s_t need not be stored in memory. This technique was introduced by J. Gaschnig for binary constraints [*IJCAI* **5** (1977), 457], and extended to $(k+1)$-ary constraints by R. Dechter in §5.2.2 of her book *Constraint Processing* (2003).)

307. It's the color of the item in `NODE[`x`]`, but irrelevant in a spacer node.

308. Node $x + 6$. (The former nodes 1 to 5 were doubly linked list heads.)

309. True, if x isn't a spacer node.

310. False; unused secondary items will still be active. (The author experimented with a version of Algorithm C that keeps primary and secondary items segregated within the `ITEM` array, but found that the extra complications were hardly ever helpful.)

311. (See further comments in the answer to exercise 7.2.2.1–8, which is similar.)

I1. [Read the first line.] Set $s \leftarrow -1$, $x \leftarrow i \leftarrow 0$. Then, for each item name α on the first line, set $i \leftarrow i+1$, `SIZE(`$4i$`)` $\leftarrow 0$, `NAME(`$4i$`)` $\leftarrow \alpha$. If α names the first secondary item, also set $s \leftarrow i$. (As in the text, `SIZE(`t`)` $\equiv$ `SET[`$t-1$`]`, `POS(`t`)` $\equiv$ `SET[`$t-2$`]`,

Catalan numbers
unary constraint
Gaschnig
IJCAI: Proc Int Joint Conf on AI
Dechter
color
list heads
secondary items
author
ITEM

and NAME(t) occupies SET$[t-4]$ and SET$[t-3]$. We're temporarily using only four slots of SET for each item.) At the end, set ACTIVE $\leftarrow i$, ITM$(0) \leftarrow 0$.

I2. [Read an option.] Go to I3 if no input remains. Otherwise let the next line of input contain the distinct item names $\alpha_1, \ldots, \alpha_k$, with respective colors $c_1, \ldots, c_k$ (where $c_j = 0$ if α_j has no color). Complain if a primary item is colored, or if all items are secondary. Do the following for $1 \le j \le k$: Find the index i_j for which $\texttt{NAME}(4i_j) = \alpha_j$, using an algorithm from Chapter 6. Set $t \leftarrow \texttt{SIZE}(4i_j)$, $\texttt{SIZE}(4i_j) \leftarrow t+1$, $\texttt{ITM}(x+j) \leftarrow i_j$, $\texttt{LOC}(x+j) \leftarrow t$, $\texttt{CLR}(x+j) \leftarrow c_j$. Finally, adjust the spacers by setting $\texttt{LOC}(x) \leftarrow k$, $x \leftarrow x+k+1$, $\texttt{ITM}(x) \leftarrow -k$. Repeat step I2.

I3. [Initialize ITEM.] Set $k \leftarrow 0$, $j \leftarrow 4$. While $k < \texttt{ACTIVE}$, set $\texttt{ITEM[}k\texttt{]} \leftarrow j$, $k \leftarrow k+1$, $j \leftarrow j+4+\texttt{SIZE}(4k)$. If $s < 0$, set $\texttt{SECOND} \leftarrow j$ and $s \leftarrow \texttt{ACTIVE}$.

I4. [Expand SET.] While $k > 0$, do the following: Set $j \leftarrow \texttt{ITEM[}k-1\texttt{]}$ and $\texttt{SIZE}(j) \leftarrow \texttt{SIZE}(4k)$, $\texttt{POS}(j) \leftarrow k-1$, $\texttt{NAME}(j) \leftarrow \texttt{NAME}(4k)$; if $\texttt{SIZE}(j) = 0$ and $k < s$, terminate (primary $\texttt{ITEM[}k-1\texttt{]}$ has no options); if $k = s$ set $\texttt{SECOND} \leftarrow j$. Set $k \leftarrow k-1$.

I5. [Adjust NODE.] For $x' = 1, 2, \ldots, x-1$, do the following if $\texttt{ITM}(x') \ge 0$: Set $i \leftarrow \texttt{ITEM[ITM}(x')-1\texttt{]}$, $j \leftarrow i+\texttt{LOC}(x')$, $\texttt{ITM}(x') \leftarrow i$, $\texttt{LOC}(x') \leftarrow j$, $\texttt{SET[}j\texttt{]} \leftarrow x'$. ▌

312. Set $\theta \leftarrow \infty$. For $0 \le k < \texttt{ACTIVE}$, do the following steps if $\texttt{ITEM[}k\texttt{]} < \texttt{SECOND}$: Set $\lambda \leftarrow \texttt{SIZE(ITEM[}k\texttt{])}$; if $\lambda = \theta$ and $\texttt{ITEM[}k\texttt{]} < i$, set $i \leftarrow \texttt{ITEM[}k\texttt{]}$; if $\lambda < \theta$, set $i \leftarrow \texttt{ITEM[}k\texttt{]}$, $\theta \leftarrow \lambda$, and terminate the loop if $\lambda = 1$. (Early termination violates the statement of the exercise, but we do it anyway because tiebreaking isn't important when $\theta = 1$; the remaining option for i is forced.) Afterwards, go to C9 if $\theta = \infty$. (Notice that θ will never be zero, although it could be zero in exercise 7.2.2.1–9.)

313. $i = 23$ (meaning x) and $c = \texttt{B}$ will leave q with no options.

314. Item 11 (q) is selected in step C2, deactivated in step C3, and hidden with OACTIVE $=$ ACTIVE $= 4$ in step C4. (Thus the options containing q, represented by nodes $2 = \texttt{SET[11]}$ and $14 = \texttt{SET[12]}$, leave the option lists for items $23 = \texttt{ITM(3)}$, $31 = \texttt{ITM(4)}$, $4 = \texttt{ITM(1)}$, $23 = \texttt{ITM(15)}$.) Step C5 sets $\texttt{TRAIL[0]} \leftarrow (4,2)$, $\texttt{TRAIL[1]} \leftarrow (31,2)$, $\texttt{TRAIL[2]} \leftarrow (17,2)$, $\texttt{TRAIL[3]} \leftarrow (23,2)$, and $y_1 \leftarrow t \leftarrow 4$. Step C6 tries option $x_0 \leftarrow 2$, deactivating items 23, 31, 4. Then C7, with OACTIVE $= 4$ and ACTIVE $= 1$, hides $(23,0)$ and jumps to C11 while trying to hide $(31,\texttt{A})$. After sizes are restored, C6 tries option $x_0 \leftarrow 14$, which is successfully hidden by C7. Here's the state when we reach C8:

	i	SET$[i]$		i	SET$[i]$
LNAME	0	p	•	17	17
RNAME	1			18	7
POS	2	1	LNAME	19	x
SIZE	3	1	RNAME	20	
•	4	6	POS	21	3
	5	11	SIZE	22	2
	6	1	•	23	12
LNAME	7	q		24	8
RNAME	8			25	15
POS	9	4		26	3
SIZE	10	2	LNAME	27	y
•	11	2	RNAME	28	
	12	14	POS	29	2
LNAME	13	r	SIZE	30	2
RNAME	14		•	31	18
POS	15	0		32	9
SIZE	16	2		33	4

k:	0	1	2	3	4		
ITEM$[k]$:	17	4	31	23	11	ACTIVE $= 3$	
x:	0	1	2	3	4	5	6
ITM(x):	0	4	11	23	31	-4	4
LOC(x):	4	6	11	26	33	4	4
CLR(x):	—	0	0	0	A	—	0
x:	7	8	9	10	11	12	13
ITM(x):	17	23	31	-4	4	23	-2
LOC(x):	18	24	32	2	5	23	2
CLR(x):	0	A	0	—	0	B	—
x:	14	15	16	17	18	19	
ITM(x):	11	23	-2	17	31	-2	
LOC(x):	12	25	2	17	31	—	
CLR(x):	0	A	—	0	B	—	

(Hmm; why are $\texttt{SET[22]} = 2$ and $\texttt{SET[23]} = 12$? Answer: When x was purified by hide$(23,\texttt{A})$ in C7, option 'p x:B' was deleted from p's list but not x's. The other two options involving x had already been deleted from x's list by hide$(11,0)$ in step C4.)

315. Suppose there's a primary item $i' \neq i$ whose options all involve i. Then hide$(i, 0)$ will remove all options of i'. If it sees $\mathtt{FLAG} = 0$ when $\mathtt{SIZE}(i')$ becomes zero, it will abort its normal operations prematurely.

Notice that this case can arise only when all options of i also include i', because step C2 minimizes $\mathtt{SIZE}(i)$. Suppose the MRV heuristic had not been used; then it would be possible to have active items without options, and step C2 would have to go to C10 after choosing an item with $\mathtt{SIZE}(i) = 0$.

316. Before testing λ versus θ in answer 312, go to C10 if $\lambda = 0$ (see exercise 315). Otherwise add a large constant to λ if $\lambda > 1$ and $\mathtt{NAME(ITEM[}k\mathtt{])}$ doesn't begin with '#'.

317. The secondary item i' is inactive, so its option list has already been purified.

318. Let FORCE be an array whose size is at least the number of primary items. After setting s' in (118), say "If $s' = 1$ and $\mathtt{FLAG} = 0$ and $i' < \mathtt{SECOND}$ and $\mathtt{POS}(i') < \mathtt{ACTIVE}$, set $\mathtt{FORCE}[f] \leftarrow i'$, $f \leftarrow f + 1$." Also, before setting FLAG in (118), set $f \leftarrow 0$.

Steps C1$^+$ through C11$^+$ are identical to steps C1 through C11, except that C1$^+$ also sets $f \leftarrow 0$, and that C2$^+$ and C8$^+$ are revised (including new intermediate steps):

C2$^+$. [Choose i.] Set $\theta \leftarrow \infty$. For $0 \le k < \mathtt{ACTIVE}$, do the following steps if $\mathtt{ITEM}[k] < \mathtt{SECOND}$: Set $\lambda \leftarrow \mathtt{SIZE(ITEM[}k\mathtt{])}$; then if $\lambda = 1$, set $\mathtt{FORCE}[f] \leftarrow \mathtt{ITEM}[k]$, $f \leftarrow f + 1$; otherwise if $\lambda = \theta$ and $\mathtt{ITEM}[k] < i$, set $i \leftarrow \mathtt{ITEM}[k]$; otherwise if $\lambda < \theta$, set $i \leftarrow \mathtt{ITEM}[k]$, $\theta \leftarrow \lambda$.

C2.1$^+$. [Forced?] If $f > 0$, set $f \leftarrow f - 1$, $i \leftarrow \mathtt{FORCE}[f]$, and go to C8.2$^+$. Otherwise if $\theta = \infty$, go to C9$^+$.

C8$^+$. [Advance to the next level.] Set $l \leftarrow l + 1$.

C8.1$^+$. [Not forced?] If $f = 0$, go to C2$^+$. Otherwise set $f \leftarrow f - 1$, $i \leftarrow \mathtt{FORCE}[f]$, and repeat step C8.1$^+$ if $\mathtt{POS}(i) \ge \mathtt{ACTIVE}$.

C8.2$^+$. [Force a move.] Perform steps C3$^+$ and C4$^+$. Then set $y_{l+1} \leftarrow t$ and go to C6$^+$. ▌

319. Indeed, it's tempting to save five mems by doing those assignments only when $i' \neq i$; the runtimes for dancing cells in (120) would then look almost 10% better! But on the author's computer, the *true* running time for, say, Problem Q increases from 64.3 user seconds to 72.2 user seconds, even though 43.9 Gμ becomes 37.6 Gμ. The reason is that conditional branches can slow down a modern computer's pipeline. (Similar remarks apply when $x'' = x'$ in (118), or $a' = a$ in general sparse-set deletion, (110).)

320. The remaining problem is the same as the former, but with option o_1 removed. Removing o_1 decreases $\mathtt{SIZE}(i')$ by $[i' \in o_1]$, for each item i'. Hence every active primary item i' of the remaining problem will have $\mathtt{SIZE}(i') \ge d - 1 = \mathtt{SIZE}(i)$. Furthermore, if $\mathtt{SIZE}(i') = \mathtt{SIZE}(i)$, we'll have $\mathtt{POS}(i') \ge \mathtt{POS}(i)$.

321. In fact there are two such nodes. One of them is labeled '$v_{31} : a_{31}$'. The other is the right child of the node labeled '$v_{21} : a_{21}$'.

322. The following algorithm operates under the same ground rules as Algorithm C$^+$, except that it uses two additional arrays, $d_0 d_1 \ldots d_T$ and $\mathtt{LS}[s]$ for $0 \le s \le T_0$, where T_0 is the number previously called T (the largest possible stage). The new size T of the x and d arrays must be larger than before, because it must now accommodate the largest possible *level* under binary branching. We use the simple subroutine

$$\text{opt}(x, x') = \text{'}x \leftarrow x'\text{; while } \mathtt{ITM}(x-1) > 0 \text{ set } x \leftarrow x - 1\text{'}$$

to set x to the leftmost item of the option that contains x'.

MRV heuristic
forced move
mems
dancing cells
author
conditional branches
pipeline
sparse-set deletion
deletion
T_0
stage
level
opt(x,x')

B1. [Initialize.] Set the problem up in memory as in exercise 311. Also insert additional entries into the SET array, if they're needed for branching heuristics. Set ACTIVE to the number of items, SECOND to the internal number of the smallest secondary item (or ∞ if there are none), and $l \leftarrow s \leftarrow t \leftarrow f \leftarrow 0$.

B2. [Not forced?] If $f = 0$, go to B3. Otherwise set $f \leftarrow f - 1$ and $i \leftarrow \texttt{FORCE}[f]$. Repeat step B2 if $\texttt{POS}(i) \geq \texttt{ACTIVE}$; otherwise set $y_s \leftarrow t$ and go to B6.

B3. [Choose i.] Set $\theta \leftarrow \infty$. For $0 \leq k < \texttt{ACTIVE}$, do the following steps if $\texttt{ITEM}[k] < \texttt{SECOND}$: If $\texttt{SIZE}(\texttt{ITEM}[k]) = 1$, set $\texttt{FORCE}[f] \leftarrow \texttt{ITEM}[k]$, $f \leftarrow f + 1$; otherwise set $\lambda \leftarrow h\bigl(\texttt{ITEM}[k]\bigr)$, where h is the given heuristic function; if $\lambda = \theta$ and $\texttt{ITEM}[k] < i$, set $i \leftarrow \texttt{ITEM}[k]$; if $\lambda < \theta$, set $i \leftarrow \texttt{ITEM}[k]$, $\theta \leftarrow \lambda$.

B4. [Forced?] If $f > 0$, set $f \leftarrow f - 1$, $i \leftarrow \texttt{FORCE}[f]$, $y_s \leftarrow t$, and go to B6. Otherwise if $\theta = \infty$, set $y_s \leftarrow t$ and go to B14.

B5. [Trail the sizes.] Terminate with trail overflow if $t + \texttt{ACTIVE}$ exceeds the maximum available TRAIL size. Otherwise set $\texttt{TRAIL}[t+k] \leftarrow \bigl(\texttt{ITEM}[k], \texttt{SIZE}(\texttt{ITEM}[k])\bigr)$ for $0 \leq k < \texttt{ACTIVE}$; then set $y_s \leftarrow t$ and $t \leftarrow t + \texttt{ACTIVE}$.

B6. [Try i's first option.] Set $d_l \leftarrow \texttt{SIZE}(i)$, $x_l \leftarrow \texttt{SET}[i]$, and do $\mathrm{opt}(x, x_l)$. (We'll try to extend the current partial solution by including the option that starts at x.)

B7. [Deactivate $\texttt{ITM}(x)$.] Set $i \leftarrow \texttt{ITM}(x)$, $i' \leftarrow \texttt{LOC}(x)$, $k \leftarrow \texttt{POS}(i)$. If $k \geq \texttt{ACTIVE}$, go to B8. Otherwise do step B11; after finishing B11, set $\texttt{ACTIVE} \leftarrow \texttt{ACTIVE} - 1$, $i'' \leftarrow \texttt{ITEM[ACTIVE]}$, $\texttt{ITEM[ACTIVE]} \leftarrow i$, $\texttt{ITEM}[k] \leftarrow i''$, $\texttt{POS}(i) \leftarrow \texttt{ACTIVE}$, $\texttt{POS}(i'') \leftarrow k$.

B8. [Advance x.] Set $x \leftarrow x + 1$. Return to B7 if $\texttt{ITM}(x) > 0$.

B9. [Enter new stage.] Set $s \leftarrow s + 1$.

B10. [Enter new level.] Set $l \leftarrow l + 1$ and $\texttt{LS}[s] \leftarrow l$. Terminate with level overflow if $l > T$ (there's no room to store x_l); otherwise return to B2.

B11. [Hide incompatible options.] For $j \leftarrow i + \texttt{SIZE}(i) - 1$ down to i, do the following if $j \neq i'$: Set $x' \leftarrow \texttt{SET}[j]$, and do step B12 if $\texttt{CLR}(x) = 0$ or $\texttt{CLR}(x') \neq \texttt{CLR}(x)$.

B12. [Visit siblings of x'.] Do $\mathrm{opt}(x'', x')$. Then while $\texttt{ITM}(x'') > 0$, do step B13 unless $x'' = x'$, and set $x'' \leftarrow x'' + 1$.

B13. [Hide option x''.] Set $i'' \leftarrow \texttt{ITM}(x'')$ and $j'' \leftarrow \texttt{LOC}(x'')$. If $j'' \geq \texttt{SECOND}$ and $\texttt{POS}(i'') \geq \texttt{ACTIVE}$, do nothing (item i'' has already been purified). Otherwise set $s' \leftarrow \texttt{SIZE}(i'') - 1$. If $s' = 0$ and $j'' < \texttt{SECOND}$, set $f \leftarrow 0$ and go to B16 (the active primary item i'' has no option beside x''). Otherwise if $s' = 1$ and $j'' < \texttt{SECOND}$, set $\texttt{FORCE}[f] \leftarrow i''$ and $f \leftarrow f + 1$. If $s' > 0$, set $x''' \leftarrow \texttt{SET}[i'' + s']$, $\texttt{SIZE}(i'') \leftarrow s'$, $\texttt{SET}[i'' + s'] \leftarrow x''$, $\texttt{SET}[j''] \leftarrow x'''$, $\texttt{LOC}(x'') \leftarrow i'' + s'$, $\texttt{LOC}(x''') \leftarrow j''$.

B14. [Visit a solution.] Visit the solution that's specified by nodes $x_{\texttt{LS}[j]}$ for $0 \leq j < s$.

B15. [Back up.] Terminate if $s = 0$. Otherwise set $t \leftarrow y_s$, $s \leftarrow s - 1$, $l \leftarrow \texttt{LS}[s]$.

B16. [Purge x_l.] If $d_l = 1$, go to B15. Otherwise, for $y_s \leq k < t$, set $\texttt{SIZE}(i') \leftarrow s'$ if $\texttt{TRAIL}[k] = (i', s')$. Then set $\texttt{ACTIVE} \leftarrow t - y_s$, $t \leftarrow y_s$, $x'' \leftarrow x_l$, and do step B13. Also set $x' \leftarrow x_l$, and do step B12. (Step B13 won't find $s' = 0$, because every active primary item has at least two active options when $d_l > 1$.) Return to B10. ▮

(Step B11 is a subroutine, called by step B7. Similarly, B12 and B13 are subroutines. Subroutine B13 might jump to B16 directly instead of returning to its caller, B12.)

This algorithm maintains the entire history $x_0 \ldots x_l$ of all branches leading to the current level, so that an interested user can monitor the current progress. But only one node per stage (namely $x_{\texttt{LS[0]}}, x_{\texttt{LS[1]}}, \ldots$) is actually needed.

secondary item
heuristic function
siblings
subroutine
level
stage

323. Step C4$^+$ takes advantage of d-way branching to hide all of i's options once, instead of d times. Binary branching can't do that.

(Incidentally, Problem C doesn't really need the MRV heuristic, because the ordering of its primary items causes Algorithm B to choose the same items i even with the trivial heuristic $h(i) = 0$. However, that heuristic takes 1665.8 Gμ with Problem H, compared to 445.7 Gμ with $h(i) = \mathtt{SIZE}(i)$ and 407.4 Gμ with Algorithm C$^+$.)

324. In step B1, provide space for $\mathtt{WT}(i)$ (initially set to 1.0) in the `SET` array, when i is primary. (It's best to store it as a single-precision floating point number, because it will be used in division. Furthermore, there won't be any problem of overflow, because the assignment "$\mathtt{WT}(i) \leftarrow \mathtt{WT}(i) + 1.0$" will do nothing when $\mathtt{WT}(i) = 2^{24}$.)

In step B13, set $\mathtt{WT}(i'') \leftarrow \mathtt{WT}(i'') + 1.0$ before going to B16.

(The author's implementation also provides optional diagnostic features that can display an item's weight at crucial times.)

325. For n queens and p knights on an $n \times n$ board, we can start with a setup like 7.2.2.1–(23) for the queens, with primary items $\{r_i \mid 1 \le i \le n\}$, $\{c_j \mid 1 \le j \le n\}$ listed in organ-pipe order, together with secondary items $\{a_s \mid 1 < s \le 2n\}$, $\{b_d \mid -n < d < n\}$. Let's add primary items $\{N_k \mid 0 \le k < p\}$ for the knights, together with secondary items $\{R_k \mid 0 \le k < p\}$, $\{C_k \mid 0 \le k < p\}$, whose colors will be row and column indices. Finally, n^2 secondary items $\{ij \mid 1 \le i,j \le n\}$ will keep the queens and knights separate. The queen options are 'r_i c_j a_{i+j} b_{i-j} ij' for $1 \le i,j \le n$; the knight options are 'N_k ij $R_k{:}i$ $C_k{:}j$ $R_{k'}{:}i'$ $C_{k'}{:}j'$', for all $1 \le i,j,i',j' \le n$ and $0 \le k < p$ with $(i-i')^2 + (j-j')^2 = 5$ and $k' = (k-1) \bmod p$. (When $n = 8$ and $p = 5$ there are $(16{+}5) + (64{+}30{+}10)$ items and $64{+}1680$ options.)

326. Using answers 324 and 325, Algorithm B will begin by choosing options 'r_5 c_1 a_6 b_4 51', 'c_4 a_5 b_{-3} 14 r_1', 'r_3 c_8 a_{11} b_{-5} 38', 'c_3 a_9 b_3 63 r_6', 'c_5 a_{12} b_2 75 r_7', 'c_7 a_{15} b_1 87 r_8', 'c_2 a_4 b_0 22 r_2', 'c_6 a_{10} b_{-2} 46 r_4' in stages 0 through 7, with $d_0 \ldots d_7 = 85321111$. Then stage 8 chooses '$N_0$ 11 $R_0{:}1$ $C_0{:}1$ $R_4{:}2$ $C_4{:}3$' as one of N_0's 295 remaining options; and stage 9 chooses 'N_1 32 $R_1{:}3$ $C_1{:}2$ $R_0{:}1$ $C_0{:}1$' as one of two for N_1. (Our formulation has not precluded 'N_1 23 $R_1{:}2$ $C_1{:}3$ $R_0{:}1$ $C_0{:}1$', since we use forward consistency only.) We first run into trouble in stage 11, when N_4's weight becomes 2 and we reach step B16 for the first time. (The clock shows only 91 kilomems so far, since the initialization.)

A complex calculation that rules out all knight placements, including all 295 options for N_1, eventually takes us back to stage 7. At that point, about 125 Mμ have elapsed; and $(N_0, N_1, N_2, N_3, N_4)$ have acquired weights (3448, 4019, 4839, 4859, 2504). Since $d_4d_5d_6d_7 = 1111$, we backtrack to stage 3, where 'c_3 a_7 b_1 43 r_4' is now forced.

Stage 4 now finds that r_1, r_5, c_4, c_6 have only two active options, while each knight item N_k has 316. But N_3 is chosen for branching, by (122), since $316/4859 < 2/1$. Another complex calculation, never branching on a queen, eventually leads back to stage 2.

And so on. After each of the options in stages 3, 2, 1, 0 has been purged, a complex exploration of knight moves consumes 150 to 200 Mμ and increases the knight-item weights. At the end (759 Mμ) those weights are (21261, 23721, 27138, 27795, 28194).

(It is *not* true that the queen items retain weight 1. Knights can be placed in such a way that forced queen moves lead to a contradiction. In fact, r_5 acquires weight 47!)

327. (a) 76 Mμ : 819 Mμ; (b) 13.3 Gμ : 118.8 Gμ; (c) 11.6 Gμ : 13.6 Tμ.

328. (Solution by P. Weigel.) Let there be primary items p_i for $0 \le i \le n$, representing pigeons, and secondary items h_j for $0 \le j < n$, representing holes. Also primary items f and F together with secondary items $*$, x, y, which cleverly fool the WTD heuristic as follows: The options for pigeons are 'p_i h_j $*{:}0$ $y{:}(i{+}j) \bmod 2$', for all i and j, except

floating point number
overflow
author
organ-pipe order
forward consistency
Weigel
pigeons

that $*$ is omitted when $j = i \bmod n$. The options for f are 'f $*$:1' and $n-2$ identical copies of 'f x:0'; the options for F are 'F $*$:1' and $n-2$ identical copies of 'F x:1'.

(First we branch on f or F, causing $*$ to get color 1. A contradiction soon arises, giving weight 2 to either p_0 or p_n. After that, branching never occurs again on f or F, because they have at least $n-2$ options and their weight remains 1. If all active p_i have weight 1, they all have at most $n/2$ remaining options, because of the parity item y.)

However, the d-way heuristic WTD† is *not* fooled, because it continues to branch on f or F until all $n-1$ options have been tried. To defeat it, we can simply add a new primary item #, with two identical options '#'; the second # shuns f and F.

For example, the running time for WTD when $n = 20$ is 6.8 gigamems, using #, and 2.6 gigamems for WTD†, compared to 473 *kilo*mems for MRV.

This XCC problem also turns out to be exponentially bad for FRB and FRB†.

329. In step B1, provide space for single-precision floating point numbers `TRY(`i`)` (initially 1.0) and `FR(`i`)` (initially 0.5) in the `SET` array, when i is primary. In step B6, also set $i_0 \leftarrow i$ and `TRY`$(i_0) \leftarrow$ `TRY`$(i_0) + 1$. Then at the end of step B8, set `FR`$(i_0) \leftarrow$ `FR`$(i_0) -$ `FR`(i_0)/`TRY`(i_0). In step B13, set `FR`$(i_0) \leftarrow$ `FR`$(i_0) + (1.0 -$ `FR`$(i_0))$/`TRY`(i_0) before going to B16. (See answer 324.)

330. (a) 1.1 Gμ : 819 Mμ; (b) 207.9 Gμ : 118.8 Gμ; (c) 17.1 Tμ : 13.6 Tμ.

331. *Any* heuristic function h that can be used in step B3 can also be used in step C2$^+$, provided that we replace 'if $\lambda = 1$' in that step by 'if $\lambda = 0$, set $f \leftarrow 0$ and go to C10$^+$; otherwise if $\lambda = 1$'. (The case $\lambda = 0$ could not previously arise, because the 'hide' routine (118) normally prevents the size of any primary item from becoming zero. Suppose, however, that i' is a primary item for which (i) every option that contains i' also contains the primary item i; and (ii) some option o contains i but not i'. Then i has more options than i'; and a non-MRV heuristic method might choose to branch on i. If so, 'hide' will be called in step C4$^+$ with `FLAG` $\leftarrow -1$, and (118) will set `SIZE`$(i') \leftarrow 0$. This size will be trailed in step C5$^+$, and we'll find $\lambda = 0$ after trying option o for item i.)

To implement the WTD heuristic (see exercise 324), increase `WT`(i') before setting `FLAG` $\leftarrow 1$ in (118), and increase `WT(ITEM[`i`])` before going to C10$^+$ from step C2$^+$. Similarly, to implement FRB (see exercise 329), update the failure rate of $i =$ `ITEM[`k`]` by setting `TRY`$(i) \leftarrow$ `TRY`$(i) + 1$ in step C4$^+$; set `FR`$(i) \leftarrow$ `FR`$(i) + (1.0 -$ `FR`$(i))$/`TRY`(i) in step C7$^+$ before going to C11$^+$, and `FR`$(i) \leftarrow$ `FR`$(i) -$ `FR`(i)/`TRY`(i) in step C8$^+$. (Sample implementations are in the online programs SSXCC-WTD0 and SSXCC-FRB0.)

332. Yes! Call them WTD† and FRB† as in (131). Then WTD† improves Problem K, achieving 2.5 Gμ; FRB† improves Problems O*, U, Y*, achieving (5844.4, 117.5, 2.5) Gμ.

333. S[10, `r`] can be 16 or 19. S[13, `r`] can be 05, 16, or 19. S[19, `p`] can be 00 or 10. S[19, `q`] can be 00 or 13.

334. Allocate also a new integer field `MATCH(`i`)`, for every *secondary* item i. Let `STAMP` be a 32-bit integer, initially 0; all the `MARK` fields are also initially 0. To get ready for testing option o, do this, assuming that `NODE[`o`]` is the spacer preceding option o: Set `STAMP` $\leftarrow$ (`STAMP` $+ 1) \bmod 2^{32}$. If `STAMP` $= 0$, set `STAMP` $\leftarrow 1$ and `MARK`$(i) \leftarrow 0$ for all i. For $x = o+1$, $o+2$, ..., set $i \leftarrow$ `ITM`(x), and exit the loop if $i \le 0$; otherwise set `MARK`$(i) \leftarrow$ `STAMP`, and if $i \ge$ `SECOND` also set `MATCH`$(i) \leftarrow ($`CLR`$(x) = 0?\ -1$: `CLR`$(x))$.

Now, given an item $i \notin o$ (equivalently, `MARK`$(i) \ne$ `STAMP`), do this: For $j \leftarrow i$, $i+1$, ..., exit the loop unsuccessfully if $j = i +$ `SIZE`(i); otherwise set $o' \leftarrow$ `SET[`j`]` and exit the loop successfully if o' does not fail the following compatibility test: "Set $x' \leftarrow o'$ and $x \leftarrow x' + 1$. While $x \ne x'$, set $i' \leftarrow$ `ITM`(x); if $i' < 0$, set $o' \leftarrow x \leftarrow x + i' - 1$;

parity
d-way
WTD†
FRB
floating point numbers
hide
MRV
WTD heuristic
FRB
online programs
WTD†
FRB†
secondary
spacer

otherwise if $\texttt{MARK}(i') = \texttt{STAMP}$ and ($i' < \texttt{SECOND}$ or $\texttt{CLR}(x) \neq \texttt{MATCH}(i')$), fail; set $x \leftarrow x + 1$." (Notice that o' is set to the spacer preceding a successful option.)

335. False. If $i \in o \in O_s$ we have $i \in I_s$ if and only if i has not been "purified"; that is, i is not a secondary item whose nonzero color was fixed by an option in $\{c_1, \ldots, c_s\}$.

336. $O_{-1} = \{00, 05, 10, 13, 16, 19\}$; $O_0^{\text{init}} = \{00, 05, 13, 19\}$; $O_0 = \{00, 19\}$; $\texttt{AGE}(10) = \texttt{AGE}(16) = -1$ (purged); $\texttt{AGE}(13) = 0$ (removed); $\texttt{AGE}(05) = 0$ (purged). (Soon afterwards, a solution will be found at stage 2, with $O_1^{\text{init}} = O_1 = \{19\}$, $O_2^{\text{init}} = \emptyset$, $\texttt{AGE}(00) = 1$, $\texttt{AGE}(19) = 3$ (both chosen), or with the roles of 00 and 19 reversed.)

(To help understand the concept of age, we can associate implicit "age labels" to the edges of tree (121). Age labels on the horizontal edges, marked '$\neq$', are always even numbers; for example, they're $(0, 0, 0)$ in the first row and $(2, 2, 2)$ in the second. Age labels on the vertical edges, marked '$=$', are always odd numbers, such as (1) in the first column and $(1, 3)$ in the second.)

337. There's an economy of scale when we can use exercise 334 to make many compatibility tests with respect to the same option.

338. Yes. For example, suppose o = '`p q`', o' = '`p r`', o'' = '`q s`', $S[o, \texttt{r}] = S[o'', \texttt{r}] = o'$, $S[o, \texttt{s}] = S[o', \texttt{s}] = o''$. If we choose o, blocking o' will trigger $(o'', \texttt{r})$, thereby enqueuing o''. Similarly, blocking o'' will enqueue o'. (No harm is done.)

339. We allow o to be within an option. Global variable `A` is the current age. Global variables `ACTIVE` and `OACTIVE` represent the number of currently active items, in a slightly tricky way: If option o is being blocked by a new choice c_{s+1}, then $\texttt{ACTIVE} = |I_{s+1}|$ and $\texttt{OACTIVE} = |I_s|$; but $\texttt{ACTIVE} = \texttt{OACTIVE} = |I_s|$ if o is being removed or purged.

O1. [Move to left spacer.] While $\texttt{ITM}(o) > 0$, set $o \leftarrow o - 1$. Then set $x \leftarrow o + 1$.

O2. [Hide o from $\texttt{ITM}(x)$.] Set $i \leftarrow \texttt{ITM}(x)$, $p \leftarrow \texttt{LOC}(x)$. If $p \geq \texttt{SECOND}$ and $\texttt{POS}(i) \geq \texttt{OACTIVE}$, go to O3 (item i has been purified); otherwise set $s' \leftarrow \texttt{SIZE}(i) - 1$. If $s' = 0$ and $p < \texttt{SECOND}$, go to O11 (i is wiped out); otherwise set $x' \leftarrow \texttt{SET}[i + s']$, $\texttt{SIZE}(i) \leftarrow s'$, $\texttt{SET}[i + s'] \leftarrow x$, $\texttt{SET}[p] \leftarrow x'$, $\texttt{LOC}(x) \leftarrow i + s'$, $\texttt{LOC}(x') \leftarrow p$.

O3. [Loop on x.] Set $x \leftarrow x + 1$. Return to O2 if $\texttt{ITM}(x) > 0$.

O4. [Begin trigger loop.] Set $\texttt{AGE}(o) \leftarrow \texttt{A}$, $p \leftarrow \texttt{TRIG}(o)$, $\texttt{HEAD} \leftarrow 0$.

O5. [Loop done?] If $p = 0$, go to O10. Otherwise set $o' \leftarrow \texttt{INFO}(p)$, $q \leftarrow \texttt{LINK}(p)$, $i' \leftarrow \texttt{INFO}(q)$, $p' \leftarrow \texttt{LINK}(q)$.

O6. [Is o' active?] Set $a \leftarrow \texttt{AGE}(o')$. If $a > \texttt{A}$, go to O7 (o' is active); otherwise set $i \leftarrow \texttt{ITM}(o' + 1)$. If $\texttt{LOC}(o' + 1) \geq i + \texttt{SIZE}(i)$, go to O8 ($o'$ is inactive).

O7. [Is i' active?] Go to O9 if $\texttt{POS}(i') < \texttt{ACTIVE}$ (i' is active).

O8. [Keep trigger.] Set $\texttt{LINK}(q) \leftarrow \texttt{HEAD}$, $\texttt{HEAD} \leftarrow p$, $p \leftarrow p'$, and return to O5.

O9. [Trigger becomes fixit.] Set $\texttt{INFO}(p) \leftarrow o$, $\texttt{LINK}(q) \leftarrow \texttt{FIX}(o')$. If $\texttt{FIX}(o') = 0$, put $o' \Rightarrow Q$ and set $\texttt{AGE}(o') \leftarrow \infty$. Then set $\texttt{FIX}(o') \leftarrow p$, $p \leftarrow p'$; return to O5.

O10. [Success.] Set $\texttt{TRIG}(o) \leftarrow \texttt{HEAD}$ and terminate successfully.

O11. [Clear the queue.] If $\texttt{QF} = \texttt{QR}$, go to O12. Otherwise $Q \Rightarrow o$, unfix(o), and repeat step O11. Here the routine 'unfix(o)' changes all of o's fixits back to triggers:

$$\text{unfix}(o) = \begin{cases} \text{Set } p \leftarrow \texttt{FIX}(o) \text{ and } \texttt{FIX}(o) \leftarrow 0. \\ \text{While } p > 0, \text{ set } o' \leftarrow \texttt{INFO}(p),\ q \leftarrow \texttt{LINK}(p),\ \texttt{INFO}(p) \leftarrow o, \\ \quad p' \leftarrow \texttt{LINK}(q),\ \texttt{LINK}(q) \leftarrow \texttt{TRIG}(o'),\ \texttt{TRIG}(o') \leftarrow p,\ p \leftarrow p'. \end{cases}$$

O12. [Failure.] Terminate unsuccessfully (because item i has lost its last option). ▮

purified
A
age
ACTIVE
OACTIVE
purified

Steps O2–O3 make option o inactive (that is, not present in the sets of its active items). Steps O4–O9 remove o from $S[o', i']$ when both o' and i' are active, by creating "holes" to be fixed; but (o', i') remains on o's trigger stack if o' or i' are inactive. Step O6 relies on the fact that the first item of o' is primary. Notice that our data structures make it easy to convert triggers to fixits and vice versa.

inactive option
first item of o'
AGE(o')
author

Step O9 makes AGE(o') infinite when o' enters Q; we'll use this in step E2 below.

340. We follow the conventions of Algorithm O, as in exercise 339.

E1. [Done?] If $\mathtt{QF} = \mathtt{QR}$, terminate successfully. Otherwise $Q \Rightarrow o$.

E2. [Is o active?] If $\mathtt{AGE}(o) = \infty$, go to E3 (o is active). Otherwise unfix(o), as in step O11, and return to E1.

E3. [Mark o's items.] Set STAMP, MARK, and MATCH as in the first paragraph of answer 334.

E4. [Begin fixit loop.] Set $p \leftarrow \mathtt{FIX}(o)$.

E5. [Loop done?] If $p = 0$, set $\mathtt{FIX}(o) \leftarrow 0$ and return to E1. Otherwise set $q \leftarrow \mathtt{LINK}(p)$, $i \leftarrow \mathtt{INFO}(q)$, $p' \leftarrow \mathtt{LINK}(q)$. (We needn't look at $\mathtt{INFO}(p)$ just now.)

E6. [Find a support.] (Now i is a primary item, and $i \notin o$.) Use the second paragraph of answer 334 to find an option o' such that $i \in o' \parallel o$. If unsuccessful, go to E8.

E7. [Record the support.] Set $\mathtt{INFO}(p) \leftarrow o$, $\mathtt{LINK}(q) \leftarrow \mathtt{TRIG}(o')$, $\mathtt{TRIG}(o') \leftarrow p$, $p \leftarrow p'$, and return to E5.

E8. [Prepare to purge.] (There's no active support for (o, i).) Set $\mathtt{FIX}(o) \leftarrow p$ and unfix(o) as in step O11.

E9. [Purge o.] Set $\mathtt{OACTIVE} \leftarrow \mathtt{ACTIVE}$ and call opt_out(o). Terminate unsuccessfully if that fails; otherwise return to E1. ▌

341. (These steps have much in common with Algorithm E above.)

A1. [Begin option loop.] Set $\mathtt{QR} \Leftarrow \mathtt{AVAIL}$, $\mathtt{QF} \leftarrow \mathtt{QR}$, $\mathtt{A} \leftarrow -1$, $\mathtt{OACTIVE} \leftarrow \mathtt{ACTIVE}$, $o \leftarrow 0$.

A2. [Mark o's items.] Set STAMP, MARK, and MATCH as in the first paragraph of answer 334.

A3. [Begin item loop.] Set $k \leftarrow 0$ and $i \leftarrow \mathtt{ITEM}[k]$.

A4. [Find a support.] If $\mathtt{MARK(i)} = \mathtt{STAMP}$, go to A6. Otherwise use the second paragraph of answer 334 to find an option o' for which $i \in o' \parallel o$. If that succeeds, go to A5. Otherwise call opt_out(o), and go to A7 if that succeeds. Otherwise terminate unsuccessfully.

A5. [Record the support.] Set $p \Leftarrow \mathtt{AVAIL}$, $q \Leftarrow \mathtt{AVAIL}$, $\mathtt{INFO}(p) \leftarrow o$, $\mathtt{LINK}(p) \leftarrow q$, $\mathtt{INFO}(q) \leftarrow i$, $\mathtt{LINK}(q) \leftarrow \mathtt{TRIG}(o')$, $\mathtt{TRIG}(o') \leftarrow p$.

A6. [Item loop done?] Set $k \leftarrow k + 1$, $i \leftarrow \mathtt{ITEM}[k]$. Return to A4 if $i < \mathtt{SECOND}$.

A7. [Option loop done?] Set $o \leftarrow o + \mathtt{LOC}(o) + 1$. Return to A2 if $o < \mathtt{LAST}$.

A8. [Empty the queue.] Call empty_q(), terminating unsuccessfully if it fails. ▌

342. Do the actions in the following paragraph for all options o with $\mathtt{AGE}(o) \geq 0$:

Set $p \leftarrow \mathtt{TRIG}(o)$ and $q' \leftarrow -1$. While $p \neq 0$, do this: "Set $q \leftarrow \mathtt{LINK}(p)$ and $p' \leftarrow \mathtt{LINK}(q)$. If $\mathtt{AGE(INFO}(p)) \geq 0$, simply set $q' \leftarrow q$; otherwise put $p \Rightarrow \mathtt{AVAIL}$, $q \Rightarrow \mathtt{AVAIL}$, and set $\mathtt{TRIG}(o) \leftarrow p'$ if $q' < 0$, $\mathtt{LINK}(q') \leftarrow p'$ if $q' \geq 0$. Then set $p \leftarrow p'$."

343. The author's experiments have found neither (a) nor (b) to be an improvement.

344. (a) An entry (o', i') might go into $\mathtt{TRIG}(o)$ long before o' becomes inactive. For example, we might have chosen $o = S[o', i']$ already in step S1 (Algorithm A).

(b) We shall place a "hint" $(-c, v)$ into every newly reconstructed trigger stack, when we wish to claim that all entries (o', i') below the hint have $\mathtt{AGE}(o') < c$. Here v is a validation code: Options change their status and their age as the search tree evolves; but this hint will remain valid as long as v is equal to $\mathtt{SS[}c \gg 1\mathtt{]}$, the "stage stamp" that was recorded for stage $\lfloor c/2 \rfloor$ in step S2.

Each step of Algorithm O$^+$ is the same as the corresponding step of Algorithm O, except as noted below. Algorithm O's variable HEAD is replaced by arrays $\mathtt{H[}a\mathtt{]}$ and $\mathtt{T[}a\mathtt{]}$ of temporary list heads and tails, for $0 \le a < 2T_0$. Initially $\mathtt{H[}a\mathtt{]} = 0$ for all a.

O4$^+$. [Begin trigger loop.] Set $\mathtt{AGE}(o) \leftarrow \mathtt{A}$, $p \leftarrow \mathtt{TRIG}(o)$, $a_{\min} \leftarrow \infty$, $p' \leftarrow 0$.

O5.1$^+$. [Hint?] If $o' \ge 0$, proceed to step O6$^+$ (this entry is not a hint).

O5.2$^+$. [Valid hint?] If $-o' \le \mathtt{A}$ and $i' = \mathtt{SS[}(-o') \gg 1\mathtt{]}$, set $p' \leftarrow p$ and go to O10$^+$.

O5.3$^+$. [Discard a useless entry.] $p \Rightarrow \mathtt{AVAIL}$, $q \Rightarrow \mathtt{AVAIL}$, $p \leftarrow p'$, and return to O5$^+$.

O7$^+$. [Is i' active?] Go to O9$^+$ if $\mathtt{POS}(i') < \mathtt{ACTIVE}$ (i' is active). Otherwise set $a \leftarrow A$.

O8$^+$. [Keep trigger.] If $a < 0$, go to O5.3$^+$. If $a < a_{\min}$, set $a_{\min} \leftarrow a$. If $\mathtt{H[}a\mathtt{]} = 0$, set $\mathtt{T[}a\mathtt{]} \leftarrow q$. Then set $\mathtt{LINK}(q) \leftarrow \mathtt{H[}a\mathtt{]}$, $\mathtt{H[}a\mathtt{]} \leftarrow p$, $p \leftarrow p'$, and return to O5$^+$.

O10$^+$. [Bucket sort.] If $p' \ne 0$ and $a_{\min} = -o' - 1$, set $p' \leftarrow \mathtt{LINK}(q)$, $p \Rightarrow \mathtt{AVAIL}$, $q \Rightarrow \mathtt{AVAIL}$ (avoid a double hint). For $a = a_{\min}, a_{\min}+1, \ldots, \mathtt{A}-1$, do this: "If $\mathtt{H[}a\mathtt{]} \ne 0$, set $\mathtt{LINK(T[}a\mathtt{])} \leftarrow p'$, $p \Leftarrow \mathtt{AVAIL}$, $q \Leftarrow \mathtt{AVAIL}$, $\mathtt{LINK}(p) \leftarrow q$, $\mathtt{INFO}(p) \leftarrow -a-1$, $\mathtt{INFO}(q) \leftarrow \mathtt{SS[}(a+1) \gg 1\mathtt{]}$, $\mathtt{LINK}(q) \leftarrow \mathtt{H[}a\mathtt{]}$, $\mathtt{H[}a\mathtt{]} \leftarrow 0$, $p' \leftarrow p$." Then if $\mathtt{H[A]} \ne 0$, set $\mathtt{LINK(T[A])} \leftarrow p'$, $p' \leftarrow \mathtt{H[A]}$, $\mathtt{H[A]} \leftarrow 0$. Finally set $\mathtt{TRIG}(o) \leftarrow p'$ and terminate successfully. (See Algorithm 5.2.5R.) ▌

345. With the hints, 6.3 Gμ (compared to 1.3 Gμ for Algorithm C); without them, 124.1 Gμ. (The ratio is even more extreme, about 1 to 150, when $n = 15$. In that case Algorithm S runs in 2.1 Tμ, compared to 432 Gμ for Algorithm C. One might guess that options need never be purged, when the "extreme" problem is being solved, because every possible option is present. But that's definitely false, even when $n = 2$! After the option '1' is removed, option '1 2' has no support with respect to item 2.)

346. If SSTAMP becomes 0, remove all hints from all trigger stacks. Then set $\mathtt{SS[}k\mathtt{]} \leftarrow k$ for $0 \le k < s$ and $\mathtt{SSTAMP} \leftarrow s$. (The values in SS are distinct, and less than SSTAMP; so they can safely be used in future hints.)

347. Let left(o, x) mean "$o \leftarrow x - 1$; while $\mathtt{ITM}(o) > 0$ set $o \leftarrow o - 1$."

J1. [Begin loop.] Do left(o, x_l), and set $p \leftarrow \mathtt{OACTIVE} \leftarrow \mathtt{ACTIVE}$, $x \leftarrow o+1$, $i \leftarrow \mathtt{ITM}(x)$.

J2. [Is i inactive?] Set $p' \leftarrow \mathtt{POS}(i)$. If $p' \ge p$, go to J4 (i has been purified).

J3. [Deactivate i.] Set $p \leftarrow p - 1$, $i' \leftarrow \mathtt{ITEM[}p\mathtt{]}$, $\mathtt{ITEM[}p\mathtt{]} \leftarrow i$, $\mathtt{ITEM[}p'\mathtt{]} \leftarrow i'$, $\mathtt{POS}(i) \leftarrow p$, $\mathtt{POS}(i') \leftarrow p'$. If $i \ge \mathtt{SECOND}$, set $\mathtt{MATCH}(i) \leftarrow \mathtt{CLR}(x)$ (see answer 334).

J4. [Loop done?] Set $x \leftarrow x + 1$ and $i \leftarrow \mathtt{ITM}(x)$. Return to J2 if $i > 0$.

J5. [Begin another loop.] Set $\mathtt{ACTIVE} \leftarrow p$. (We'll block the options $\ne o$ from the lists of all the newly inactive items, $\{\mathtt{ITEM[}k\mathtt{]} \mid \mathtt{ACTIVE} \le k < \mathtt{OACTIVE}\}$.)

J6. [Block $\mathtt{ITEM[}p\mathtt{]}$'s options.] Set $i \leftarrow \mathtt{ITEM[}p\mathtt{]}$ and $j \leftarrow i + \mathtt{SIZE}(i) - 1$. If $i \ge \mathtt{SECOND}$ and $\mathtt{MATCH}(i) \ne 0$, purify i as follows: "While $j \ge i$, set $o' \leftarrow \mathtt{SET[}j\mathtt{]}$, $j \leftarrow j-1$, and call opt_out(o') if $\mathtt{CLR}(o') \ne \mathtt{MATCH}(i)$." Otherwise block i's options $\ne o$ as follows: "While $j \ge i$, do left$(o', \mathtt{SET[}j\mathtt{]})$, set $j \leftarrow j - 1$, and call opt_out(o') if $o' \ne o$."

J7. [Loop done?] Set $p \leftarrow p + 1$. Return to J6 if $p < \mathtt{OACTIVE}$.

hint
validation code
stage stamp
stamping
T_0
Bucket sort
sort
radix list sort
hints
purified
purify

J8. [Deactivate o.] Set `SIZE(ITEM[`p`])` $\leftarrow 0$ for all p with `ACTIVE` $\le p <$ `OACTIVE` and `ITEM[`p`]` $<$ `SECOND`. Also set `AGE(`o`)` $\leftarrow$ `A`. ▌

It's important for j to be decreasing, not increasing, in step J6, because the options being blocked move right as they leave the sets. The calls on opt_out(o') will not fail, because o is supported. No change is needed to any trigger stack in step J8, because the active option o being chosen contains no active primary items. Sizes are zeroed in that step because step O6 should henceforth consider option o to be inactive.

348. There's no closed tour, hence no solution, when mn is odd. We shall write simply 'ij' for cell (i,j). Let $ij \overset{Q}{\text{---}} i'j'$ be the adjacency relation for the $m \times n$ queen graph, namely "$ij \ne i'j'$ and ($i=i'$ or $j=j'$ or $i+j=i'+j'$ or $i-j=i'-j'$)"; similarly, let $ij \overset{N}{\text{---}} i'j'$ denote adjacency in the corresponding knight graph, "$(i-i')^2+(j-j')^2=5$." Let A be the set $\{i'j' \mid i'j' \overset{Q}{\text{---}} ij\}$ of cells attacked by the queen. Say that cell $i'j'$ is *red* if it has the same parity as the queen's cell, that is, if $(i'+j'+i+j) \bmod 2 = 0$; otherwise, $i'j'$ is *white*. We will assume that all red cells have odd labels; the other case is similar.

Suppose A has a_1 red cells and a_0 white cells; usually $a_1 > a_0$. Also suppose P has p_1 odd numbers, p_0 even numbers. We must have $p_1 \le a_1$ and $p_0 \le a_0$, because $P \subseteq A$.

Let there be mn primary items $i'j'$ and secondary items $x_{i'j'}$, for $0 \le i' < m$, $0 \le j' < n$; also mn primary items $\#k'$ and secondary items $y_{k'}$, for $1 \le k' \le mn$. The "color" of $x_{i'j'}$ will be a label, and the "color" of $y_{k'}$ will be a cell. The options are '$i'j'$ $\#k'$ $x_{i'j'}{:}k'$ $y_{k'}{:}i'j'$, $x_{i''j''}{:}k''$ $y_{k''}{:}i''j''$', for all i', j', k', i'', j'', k'' such that $i'j' \overset{N}{\text{---}} i''j''$ and $k''=1+(k' \bmod mn)$ and $OK(i',j',k')$ and $OK(i'',j'',k'')$ are true, where $OK(i',j',k')$ means "$0 \le i' < m$, $0 \le j' < n$, $(i'+j'+k'+i+j) \bmod 2 = 1$, and $k' \in P \Rightarrow i'j' \in A$." (The option says, "Step k' of the tour goes from cell $i'j'$ to cell $i''j''$.")

We can make this construction much more efficient when $a_1 = p_1$, by simply omitting all of the options in which $i'j'$ is red, $i'j' \in A$, and $k' \notin P$; also those in which $i''j''$ is red, $i''j'' \in A$, and $k'' \notin P$. Moreover, if $a_1 - p_1 = t > 0$, we can retain those options but append $*'$ or $*''$ to each of them, where $*'$ and $*''$ are new primary items of multiplicity t. (This modification makes it an MCC problem, not XCC, if $t > 1$.)

349. Use the primary item $i'j'$ only when $i'j'$ is white, and $\#k'$ only when k' is odd; use the secondary item $x_{i'j'}$ only when $i'j'$ is red, and $y_{k'}$ only when k' is odd. Also introduce new primary items $i'j'{\rightarrow}$ and ${\rightarrow}i'j'$ for every red cell $i'j'$. The options are now '$\#k'$ $i'j'{\rightarrow}$ $x_{i'j'}{:}k'$ $y_{k'}{:}i'j'$ $i''j''$ $x_{i'''j'''}{:}k'''$ $y_{k'''}{:}i'''j'''$ ${\rightarrow}i'''j'''$' for all i', j', k', i'', j'', k'', i''', j''', k''' such that $i'j' \overset{N}{\text{---}} i''j''$, $i''j''$ is white, $i''j'' \overset{N}{\text{---}} i'''j'''$, $i'j' \ne i'''j'''$, k' is odd, $k''=k'+1$, $k'''=1+(k'' \bmod mn)$, $OK(i',j',k')$, $OK(i'',j'',k'')$, and $OK(i''',j''',k''')$.

350. By fixing the labels of those eight cells, the construction produces 10591 options on $200+100$ items. Its 43 solutions are found by Algorithm S with heuristics (MRV, WTD, FRB) in respectively (343, 231, 1602) Gμ. (And the FC method FRB$^{\dagger *}$ takes 1475 Gμ.) The solutions shown here minimize and maximize the sum of attacked labels.

73	34	75	04	**71**	36	69	66	**17**	96
76	**05**	72	35	**02**	09	18	**97**	68	65
33	74	**03**	08	**37**	70	**67**	16	95	98
06	77	84	01	10	**19**	38	99	64	15
83	32	**07**	24	85	00	**11**	14	**61**	94
78	25	82	**31**	20	**23**	60	39	12	63
81	30	**79**	22	**59**	86	**13**	62	93	40
26	**53**	28	87	56	21	90	**43**	48	45
29	80	51	54	**89**	58	49	46	**41**	92
52	27	88	57	50	55	42	91	44	**47**

31	34	63	08	**61**	36	91	14	**17**	58
64	**07**	32	35	90	09	18	**59**	92	15
33	30	**89**	62	**37**	60	**13**	16	57	98
06	65	86	01	10	**19**	38	99	12	93
29	**02**	**05**	88	85	00	**11**	94	**97**	56
66	87	28	**03**	20	**23**	84	39	48	95
27	04	**67**	22	**83**	78	**47**	96	55	40
68	**73**	70	77	24	21	80	**43**	52	49
71	26	75	82	**79**	46	51	54	**41**	44
74	69	72	25	76	81	42	45	50	**53**

351. (a, b) Almost all parameter combinations (m,n,i,j) are unsatisfiable, either because $p_1 > a_1$ or because a small search tree proves impossibility. (The cases $(5,6,2,2)$ and $(6,6,2,2)$ are MCC problems.) The only surviving combinations with $m \le n$ are $(6,7,2,3)$: $2 \cdot 52$ solutions, 3.6 Gμ; $(7,8,2,2)$: 16 solutions, 9.6 Gμ; $(7,8,2,3)$: 1206 solutions, 597.1 Gμ; $(7,8,3,3)$: $2 \cdot 989$ solutions, 1450.7 Gμ; $(7,10,2,4)$: 491 solutions, 1338.4 Gμ; $(8,8,3,3)$: $2 \cdot 688$ solutions, 2154.8 Gμ; $(8,9,3,4)$: $2 \cdot 6010$ solutions, 33.9 Tμ; and two really hard cases $(9,10,4,4)$ and $(10,10,4,4)$. [All runtimes are from

sparse-set representation
queen graph
knight graph
parity
multiplicity
MCC problem

DC-WTD, without symmetry reduction. This problem was originally posed by Peter Weigel, who was able to show after massive calculations that the 9×10 case has exactly $2\cdot 1658756$ solutions. He has also found many thousands of solutions to the 10×10 problem — for which he estimates that, with methods that are currently known, about 10 years of computation will be needed to obtain a complete count.]

symmetry reduction
Weigel
opt(x, x')
deactivate(i)
multiplicities
forced

18	09	06	39	34	**37**
05	28	**17**	36	**07**	40
10	**19**	**08**	33	38	35
29	**04**	27	**16**	**41**	**32**
20	**11**	30	**01**	24	15
03	26	**13**	22	**31**	42
12	21	**02**	25	14	**23**

05	12	**03**	56	**31**	28	51	54
44	**01**	06	**13**	52	55	30	27
11	**04**	45	**02**	**29**	**32**	**53**	50
20	**43**	**16**	**07**	14	39	26	33
17	10	**19**	46	**23**	36	49	38
42	21	**08**	15	40	**47**	34	25
09	18	**41**	22	35	24	**37**	48

69	18	**01**	38	**67**	22	**53**	12	65	10
36	39	68	**17**	**02**	**13**	66	09	52	55
19	70	**37**	**04**	21	**08**	**23**	54	**11**	**64**
40	35	20	**31**	**16**	**03**	14	45	56	51
27	30	**05**	42	**07**	24	**59**	48	63	46
34	**41**	28	25	**32**	15	44	**61**	50	57
29	26	33	06	**43**	60	49	58	**47**	62

53	46	21	**02**	55	48	**23**	14
20	**01**	54	**47**	22	**13**	56	49
45	52	**19**	12	**03**	50	15	24
18	**11**	**64**	51	**16**	**05**	**32**	57
63	44	**17**	**04**	**31**	58	25	06
10	**41**	36	**61**	38	**07**	28	33
43	62	39	**08**	35	30	**59**	26
40	09	42	**37**	60	27	34	**29**

10	63	36	**19**	72	65	34	**43**
37	20	09	**64**	35	44	**71**	66
62	**11**	18	**01**	06	**67**	42	33
21	38	**05**	**08**	**17**	70	45	68
12	**61**	**02**	39	**04**	**07**	**32**	**41**
55	22	**59**	**16**	**31**	40	69	46
60	**13**	56	**03**	52	49	30	27
23	54	15	58	25	28	**47**	50
14	57	24	**53**	48	51	26	**29**

23	20	25	78	**59**	10	57	76	**61**	14
26	**79**	22	09	18	77	60	**13**	56	75
21	24	**19**	80	**11**	58	**17**	74	15	62
36	27	82	**03**	**08**	**05**	12	63	52	55
83	**02**	**37**	06	81	66	**53**	**16**	**73**	**64**
28	35	84	**01**	**04**	**07**	40	65	54	51
85	88	**31**	38	**41**	48	**67**	70	45	72
34	**29**	90	87	**32**	39	46	**43**	50	69
89	86	33	30	**47**	42	49	68	**71**	44

352. Besides opt(x,x'), the following adaptation of Algorithm B uses the subroutine

$$\text{deactivate}(i) = \begin{cases}\texttt{ACTIVE} \leftarrow \texttt{ACTIVE}-1,\ i''' \leftarrow \texttt{ITEM[ACTIVE]},\ k \leftarrow \texttt{POS}(i);\\ \texttt{ITEM[ACTIVE]} \leftarrow i,\ \texttt{ITEM}[k] \leftarrow i''',\ \texttt{POS}(i) \leftarrow \texttt{ACTIVE},\ \texttt{POS}(i''') \leftarrow k.\end{cases}$$

The elements of its `TRAIL` array are triplets, not pairs.

F1. [Initialize.] Set the problem up in memory as in step B1 of answer 322. Also insert additional entries $\texttt{BOUND}(i)$ and $\texttt{SLACK}(i)$ into the `SET` array for each primary item i, initialized to v_i and $v_i - u_i$ when i's given multiplicities are $[u_i\,..\,v_i]$. Terminate if $\texttt{SIZE}(i) < u_i$ for any i. Deactivate any items for which $\texttt{SIZE}(i) = 0$.

F2. [Not forced?] If $f = 0$, go to F3. Otherwise set $f \leftarrow f-1$ and $i \leftarrow \texttt{FORCE}[f]$. Repeat step F2 if $\texttt{POS}(i) \ge \texttt{ACTIVE}$; otherwise set $y_s \leftarrow t$, $\theta \leftarrow 1$, and go to F6.

F3. [Choose i.] Set $\theta \leftarrow \infty$. For $0 \le k < \texttt{ACTIVE}$, do the following steps if $\texttt{ITEM}[k] < \texttt{SECOND}$: Set $i' \leftarrow \texttt{ITEM}[k]$, $s \leftarrow \min\bigl(\texttt{SLACK}(i'), \texttt{BOUND}(i')\bigr)$, $\lambda \leftarrow \texttt{SIZE}(i') + 1 + s - \texttt{BOUND}(i')$. If $\lambda = 1$, set $\texttt{FORCE}[f] \leftarrow i'$ and $f \leftarrow f+1$, for $1 \le j \le \texttt{SIZE}(i')$. (In that case, every remaining option of i' is forced.) Otherwise, if $\lambda \le \theta$ and $(\lambda < \theta$ or $(s \le \sigma$ and $(s < \sigma$ or $(\texttt{SIZE}(i') \ge \sigma'$ and $(\texttt{SIZE}(i') > \sigma'$ or $i' < i)))))$, set $\theta \leftarrow \lambda$, $i \leftarrow i'$, $\sigma \leftarrow s$, and $\sigma' \leftarrow \texttt{SIZE}(i')$. (See exercise 7.2.2.1–166.)

F4. [Forced?] If $f > 0$, set $f \leftarrow f-1$, $i \leftarrow \texttt{FORCE}[f]$, $\theta \leftarrow 1$, $y_s \leftarrow t$, and go to F6. Otherwise if $\theta = \infty$, set $y_s \leftarrow t$ and go to F16.

F5. [Trail the sizes.] Terminate with trail overflow if $t + \texttt{ACTIVE}$ exceeds the maximum available `TRAIL` size. Otherwise set $\texttt{TRAIL}[t+k] \leftarrow \bigl(\texttt{ITEM}[k], \texttt{SIZE(ITEM}[k]\texttt{)}, \texttt{BOUND(ITEM}[k]\texttt{)}\bigr)$ for $0 \le k < \texttt{ACTIVE}$, omitting `BOUND` if $\texttt{ITEM}[k] \ge \texttt{SECOND}$. Then set $y_s \leftarrow t$ and $t \leftarrow t + \texttt{ACTIVE}$.

F6. [Try i's first option.] Set $d_l \leftarrow \theta$, $x_l \leftarrow \texttt{SET}[i]$, and do opt$(x, x_l)$. (We'll try to extend the current partial solution by including the option that starts at x.)

F7. [Commit $\texttt{ITM}(x)$.] Set $i \leftarrow \texttt{ITM}(x)$, $i' \leftarrow \texttt{LOC}(x)$, $k \leftarrow \texttt{POS}(i)$. If $k \ge \texttt{ACTIVE}$, go to F10 (the secondary item i has been purified). Otherwise, if $i < \texttt{SECOND}$, set $\texttt{BOUND}(i) \leftarrow \texttt{BOUND}(i) - 1$, and go to F9 if $\texttt{BOUND}(i) > 0$.

F8. [Cover i.] (Now $i \ge \texttt{SECOND}$ or $\texttt{BOUND}(i) = 0$.) Do step F13. Then deactivate(i) and go to F10.

F9. [Hide option x.] Set $x'' \leftarrow x$ and do step F15.

F10. [Advance x.] Set $x \leftarrow x+1$. Return to F7 if $\texttt{ITM}(x) > 0$.

F11. [Enter new stage.] Set $s \leftarrow s+1$.

F12. [Enter new level.] Set $l \leftarrow l+1$ and $\texttt{LS[}s\texttt{]} \leftarrow l$. Terminate with level overflow if $l > T$ (there's no room to store x_l); otherwise return to F2.

F13. [Hide incompatible options.] For $j \leftarrow i + \texttt{SIZE(}i\texttt{)} - 1$ down to i, do the following if $j \neq i'$: Set $x' \leftarrow \texttt{SET[}j\texttt{]}$, and do step F14 if $\texttt{CLR(}x\texttt{)} = 0$ or $\texttt{CLR(}x'\texttt{)} \neq \texttt{CLR(}x\texttt{)}$.

F14. [Visit siblings of x'.] Do opt(x'', x'). Then while $\texttt{ITM(}x''\texttt{)} > 0$, do step F15, and set $x'' \leftarrow x'' + 1$.

F15. [Hide option x''.] Set $i'' \leftarrow \texttt{ITM(}x''\texttt{)}$ and $j'' \leftarrow \texttt{LOC(}x''\texttt{)}$. If $j'' \geq \texttt{SECOND}$ and $\texttt{POS(}i''\texttt{)} \geq \texttt{ACTIVE}$, do nothing (item i'' has already been purified). Otherwise set $s' \leftarrow \texttt{SIZE(}i''\texttt{)} - 1$. If $j'' < \texttt{SECOND}$ and $s' < \texttt{BOUND(}i''\texttt{)} - \texttt{SLACK(}i''\texttt{)}$, set $f \leftarrow 0$ and go to F18 (the active primary item i'' needs x'' to attain its lower bound). Otherwise, if $j'' < \texttt{SECOND}$ and $s' = 0$, deactivate(i'') (which has lost its last option). Otherwise, if $s' > 0$, set $x''' \leftarrow \texttt{SET[}i''+s'\texttt{]}$, $\texttt{SIZE(}i''\texttt{)} \leftarrow s'$, $\texttt{SET[}i''+s'\texttt{]} \leftarrow x''$, $\texttt{SET[}j''\texttt{]} \leftarrow x'''$, $\texttt{LOC(}x''\texttt{)} \leftarrow i''+s'$, $\texttt{LOC(}x'''\texttt{)} \leftarrow j''$.

F16. [Visit a solution.] Visit the solution that's specified by nodes $x_{\texttt{LS[}j\texttt{]}}$ for $0 \leq j < s$.

F17. [Back up.] Terminate if $s = 0$. Otherwise set $t \leftarrow y_s$, $s \leftarrow s-1$, $l \leftarrow \texttt{LS[}s\texttt{]}$.

F18. [Purge x_l.] If $d_l = 1$, go to F17. Otherwise, for $y_s \leq k < t$, set $\texttt{SIZE(}i'\texttt{)} \leftarrow s'$ if $\texttt{TRAIL[}k\texttt{]} = (i', s', b')$; also set $\texttt{BOUND(}i'\texttt{)} \leftarrow b'$ if $i' < \texttt{SECOND}$. Then set $\texttt{ACTIVE} \leftarrow t - y_s$, $t \leftarrow y_s$, $x' \leftarrow x_l$, $d_l \leftarrow 1$, and do step F14. Return to F12. ▮

(As before, steps F13, F14, and F15 are subroutines; and subroutine F15 might jump directly to F18 instead of returning to its caller.)

353. Begin step F3 with $\theta \leftarrow \infty$ and $\theta' \leftarrow \infty$, where θ is an integer and θ' is a floating point number. Later in that step, if $\lambda > 1$, set $\lambda' \leftarrow \lambda/w$, where $w = \texttt{WT(}i'\texttt{)}$ for WTD, $w = \texttt{FR(}i'\texttt{)}$ for FRB. Then if $\lambda' \leq \theta'$ and $(\lambda' < \theta'$ or $(s \leq \sigma$ and $(s < \sigma$ or $(\texttt{SIZE(}i'\texttt{)} \geq \sigma'$ and $(\texttt{SIZE(}i'\texttt{)} > \sigma'$ or $i' < i)))))$, set $\theta' \leftarrow \lambda'$, $\theta \leftarrow \lambda$, $i \leftarrow i'$, $\sigma \leftarrow s$, and $\sigma' \leftarrow \texttt{SIZE(}i'\texttt{)}$.

354. Assume that $m > 1$ and $n > 1$. Let # be a primary item of multiplicity k; also let xy be a primary item of multiplicity $[1\,..\,k]$, for $0 \leq x < m$ and $0 \leq y < n$. If d is odd, there are mn options, for $0 \leq x_0 < m$ and $0 \leq y_0 < n$, consisting of # together with $\{xy \mid 0 \leq x < m,\ 0 \leq y < n,\ (x-x_0)^2 + (y-y_0)^2 < d^2/4\}$. If d is even, there are $(m-1)(n-1)$ options, for $1 \leq x_0 < m$ and $1 \leq y_0 < n$, consisting of # together with $\{xy \mid 0 \leq x < m,\ 0 \leq y < n,\ (x-x_0)(x-x_0+1) + (y-y_0)(y-y_0+1) < d^2/4\}$.

355. The solutions are the ways to pile such polyominoes into that shape, using at least u_p and at most v_p copies of piece p, so that at least u_{xy} and at most v_{xy} of those pieces occupy cell (x, y).

Problem $\mathcal{E}$ in the text (see (134)) is the special case where the pieces are simply the twelve pentominoes, with $u_p = v_p = 1$, and the shape is simply a 7×7 square, with $u_{xy} = 1$ and $v_{xy} = 1 + [x=0$ or $y=0$ or $x=6$ or $y=6]$ for $0 \leq x, y < 7$. (Symmetry was broken by restricting piece P to one of its eight orientations.) Two of the 10,343,858 solutions are shown here: The most interesting one doubles up only on cells that are corners or adjacent to corners. (It's unique, except for reflecting the RW bipair.) The other is one of only seven that have X in the center, and double up only above the diagonal.

```
 V VX  V  Y  Y PY PY     P PY YZ YZ VY VW VW
VX  X  X  W  Y  P OP     P  P  Z  Y  W  W OV
 V  X  W  W  R  P  O     P  Z  Z  X  W  S OV
 T  W  W  R  R  R  O     Q  Q  X  X  X  S OS
 T  T  T  R  Z  Z  O     Q  U  U  X  R  T OS
QT  S  S  S  U  Z OU     Q  U  R  R  R  T OS
QS QS  Q  Q  U UZ UZ     Q  U  U  R  T  T  T
```

(A. O. Muñiz's post for 2 May 2023 in `https://puzzlezapper.com/blog/` proposes the name "polyomino piling" for cases where all $u_{xy} > 0$ and some $v_{xy} > 1$.)

siblings
subroutines
Symmetry was broken
bipair
Muñiz
polyomino piling
piling

356. Assume that there's a secondary item for each variable, and that '$x = a$' is represented by options that contain 'x:a'. Then the example can be handled by introducing a new primary item #, with multiplicity $[0\,..\,2]$; we simply include # in the options that force $x = a$, $y = b$, and $z = c$.

This construction fails, however, if some option contains more than one of the colorings x:a, y:b, z:c, because it requires us to include # more than once in that option. We can omit any option where all three appear. And we can add '+:1' to options where two of them appear, where + is a new secondary item. Then the options '! # +:1' and '! +:0', where ! is a new primary item, finish the job.

In general, a similar scheme will encode '$x_1 \neq a_1$ or $\cdots$ or $x_k \neq a_k$', using a new primary item # whose multiplicity is $[0\,..\,k-1]$.

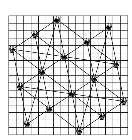
357. (This is essentially an application of the idea in the previous exercise.) If $0 < p < q$ and $p \perp q$, the relevant lines '$y = (p/q)x + \text{constant}$' of slope p/q can be enumerated by considering the lower left points at which they intersect an $n \times n$ grid, namely $\{(x, y) \mid 0 \le x < n - 2q, 0 \le y < n - 2p\} \setminus \{(x, y) \mid x \ge q, y \ge p\}$. Also, the lines of slope s are in one-to-one correspondence with lines of slopes $1/s$, $-s$, and $-1/s$. Thus we can build a table of $N(n)$ triples $(\alpha_i, \beta_i, \gamma_i)$, where line i is characterized by '$\alpha_i x + \beta_i y = \gamma_i$'. (We have $(N(4), \ldots, N(12)) = (0, 12, 32, 76, 136, 252, 356, 572, 836)$.)

Start with the items r_i, c_j, a_s, b_d and the n^2 options of the n queens problem (see 7.2.2.1–(23)). Add $N(n)$ new primary items $\#_k$, for $0 \le k < N(n)$, each with multiplicity $[0\,..\,2]$. Then append to option '$r_i\ c_j\ a_{i+j},\ b_{i-j}$' every item $\#_t$ for which $\alpha_t i + \beta_t j$ equals γ_t in the table of relevant lines.

Historical notes: Answer 7.1.4–241(a) pointed out Sam Loyd's observation in 1896 that this problem is solvable when $n = 8$. Solutions for larger n were counted by F. Pahl, who posted the results to `math.stackexchange` in answer to question 4642059 (February 2023). The asymptotic behavior is currently unknown — not even whether solutions exist for infinitely many n. See OEIS sequence A365437.

358. Yes, that would be quick. It's well known that points $\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$ are collinear if and only if $x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) = 0$. This test already rules out more than 3.5 million cases after examining at most the first five rows. [The fastest way to visit all solutions is probably a customized backtrack program. But the MCC technology would be helpful if additional constraints were imposed.]

359. This beautiful "orchard pattern" ($m = 17$) is unique, except for rotation and reflection. It was discovered when studying the $N(16) = 2668$ relevant lines of answer 357. (Incidentally, the line $x + 2y = 4$ is part of 1172 solutions that contain all three of its points; at the other extreme, the line $x + 2y = 14$ is part of 226825 solutions that contain ≥ 3 of its eight points.)



360. (Of course this is quite different from the queens-and-knights problem of exercise 325.) Let Q and S be primary items with multiplicities q and s, respectively. Also let r_i, c_j, a_d, b_d, and s_{ij} be secondary items, for $0 \le i < m$, $0 \le j < n$, and $0 \le d < m+n-1$. There are two options for every cell ij of the board, one for queen placement and one for knight placement, namely

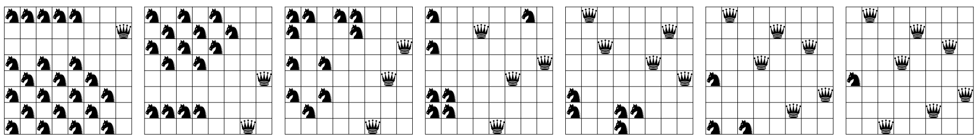
$$\text{'}Q\ r_i\ c_j\ a_{i+j}\ b_{i+n-1-j}\ \ \bigcup\{s_{kl}{:}0 \mid ij \mathrel{—\!—} kl \text{ in the } m \times n \text{ knight graph}\}\text{'},$$

$$\text{'}S\ s_{ij}\ r_i{:}0\ c_j{:}0\ a_{i+j}{:}0\ b_{i+n-1-j}{:}0\ \ \bigcup\{s_{kl}{:}0 \mid ij \mathrel{—\!—} kl \text{ in the } m \times n \text{ knight graph}\}\text{'}.$$

The trick here, due to Peter Weigel, is to give color 0 to some secondary items while giving a blank (unmatchable) color to others, while noting that queens can be a knight's move apart and knights can be a queen's move apart. (See exercise 7.2.2.1–169.)

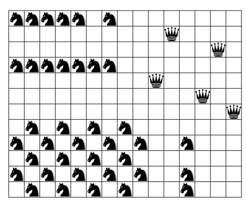
Historical notes
Loyd
Pahl
OEIS
collinear
backtrack
orchard pattern
unique
queens-and-knights problem
knight graph
Weigel
blank (unmatchable) color
unmatchable

361. A bevy of $q = 1, \ldots, 7$ queens can coexist respectively with (22, 15, 11, 7, 5, 3, 1) knights in respectively (12, 40, 56, 328, 16, 40, 104) ways, ignoring symmetry; and no additional knights can be added. For example, here's a maximum solution for each case:



The dancing links technology of Algorithm 7.2.2.1M handles each of these 14 problems in fewer than 301 Mμ. Algorithm F is faster yet — at most 183 Mμ, when $(q, s) = (2, 15)$. But the WTD heuristic makes it worse, when q is small: More than 1 Gμ when $q = 1$. And the FRB heuristic is champion in all cases with $q < 5$; for instance, it needs only 33 Mμ when $(q, s) = (3, 12)$, compared to 186 Mμ by Algorithm 7.2.2.1M.

Indeed, the FRB heuristic turns out to be dramatically superior on larger instances of this problem. For example, it needs only 991 gigamems to find all solutions when $(m, n, q, s) = (12, 15, 5, 38)$; the other three methods spend more than 49 Tμ on this problem before even completing the first of 143 branches at the root of the search tree! (Incidentally, all 8 solutions are obtained from the one shown by reflecting the board and-or by using the "tilted square" trick of exercise 7.2.2–11. The maximum number of knights for $(m, n, q) = (12, n, 5)$ and $n \geq 12$ turns out to be $25 + \lfloor \frac{9}{2}(n - 12) \rfloor$.)



Historical notes: This problem was introduced by ITA Software in spring 2002 as a pre-interview test question, in the case $m = n = 8$ and $q = s$, and popularized in 2004 by Roger Hui (see `http://code.jsoftware.com/wiki/Essays/Queens_and_Knights`).

362. Construct an MCC problem with primary items R_{ik} and C_{jk} of multiplicity k, for $0 \leq i, j < 10$ and $1 \leq k \leq 4$; also secondary items ijk, which are essentially Boolean variables whose color is [cell ij is colored k]; also primary items ij. There are 400 options, '$ij\ R_{ik}\ C_{jk}\ ij1{:}[k\,{=}\,1]\ ij2{:}[k\,{=}\,2]\ ij3{:}[k\,{=}\,3]\ ij4{:}[k\,{=}\,4]$'; they enforce (i) and (ii).

Also introduce primary items #ij, which signify that cell ij belongs to a polyomino whose size matches its color. Every potential placement of a c-omino has a corresponding option that includes c of these sharp items. For example, the solution shown makes use of the options '#06 051:0 061:1 071:0 161:0'; '#03 #13 022:0 032:1 042:0 122:0 132:1 142:0 232:0'; '#02 #11 #12 013:0 023:1 033:0 103:0 113:1 123:1 133:0 213:0 223:0'; '#00 #01 #10 #20 004:1 014:1 024:0 104:1 114:0 204:1 214:0 304:0'.

```
4432441332
4332414432
4243433421
1243343424
2344142334
2331442344
4423324413
3424324143
3414234243
3144233244
```

It turns out that Tullis's tapestry is *unique*: There's only one solution, except for rotation and reflection(!).

363. (a) There exists $y = y_1y_2y_3$ such that $S(x, y)$ and $S(y, x)$ both hold.

(b) There exists $z = z_1z_2z_3$ such that $S(x, z)$ and $S(z, y)$ both hold.

(c) Given any variable x, we can introduce new auxiliary variables x^L and x^R, analogous to z in (b), by stipulating $S(x^L, x)$ and $S(x, x^R)$. Repeating this process leads to auxiliary variables such as x^{RRRLL}, which is the k-tuple $x_1^R x_1 \ldots x_{k-3} x_k^{RRRL} x_k^{RRRLL}$ when $k \geq 3$. Each new auxiliary $x^{\alpha L}$ introduces an independent value $x_k^{\alpha L}$ at the right; each new auxiliary $x^{\alpha R}$ introduces an independent value $x_1^{\alpha R}$ at the left.

Let $x^{(i)} = x^{L^{i-1}R^{k-1}} = x_1^{L^{i-1}R^{k-1}} \ldots x_1^{L^{i-1}R} x_i$. Also let $y^{[j]} = y^{R^{k-j}L}$, a k-tuple that ends with $\ldots y_j y_k^{R^{k-j}L}$. Therefore $S(x^{(i)}, y^{[j]})$ enforces $x_i = y_j$, and no other constraints between x and y.

WTD heuristic
FRB heuristic
tilted square
Historical notes
ITA Software
Hui
MCC problem
unique
auxiliary variables

(d) Enforce $x_1 = x_4$ using the construction in (c) with $x = y$. Then enforce $x_2 = x_3$, using the same construction (but with a fresh set of auxiliary variables). Finally, enforce $x_4 = x_5$, with yet another fresh set.

364. Suppose $\mathcal{P}$ has m relations R_j, where R_j is k_j-ary and has s_j tuples. Concatenate all the relations together into a single k-ary relation R with $s_1 \ldots s_m$ tuples, where $k = k_1 + \cdots + k_m$. Let D be the union of the domains of $\mathcal{P}$; each variable of $\mathcal{P}^*$ has domain D^k, the set of k-tuples of values in D. One special variable, v, represents the concatenation of the scopes of all the relations in $\mathcal{P}$. All other variables of $\mathcal{P}^*$ are auxiliaries, used to enforce equalities between components of v using the binary relation S as in exercise 363(d). For example, when $\mathcal{P}$ is given by (1) and (2), we have $k = 3+2+3 = 8$; R consists of the $3 \times 3 \times 3$ tuples $\{\texttt{BANBECOD}, \texttt{BANBECON}, \ldots, \texttt{SINSOLED}\}$; the domain of $\mathcal{P}^*$ is D^8, where $D = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}, \texttt{I}, \texttt{L}, \texttt{N}, \texttt{O}, \texttt{S}, \texttt{U}\}$; and v is a variable that we might call '$x_1x_3x_5x_1x_4x_2x_4x_5$'. Variable v must satisfy the unary constraint $R(v)$, together with appropriate applications of S to enforce the conditions $v_1 = v_4$, $v_3 = v_8$, and $v_5 = v_7$.

[This exercise illustrates a contrast between mathematics and computer science: A mathematician likes elegant constructions, but a computer scientist finds it painful to even *think* about such wild inefficiency (despite needing "only" polynomial time)!]

365. (a) True; we can't have $R_\alpha(x,y) \wedge R_\alpha(x',y')$ when $x < x'$ and $y > y'$. (Hence any CSP in which all relations have the form R_α, for one or more α, is efficiently decidable.)

(b) In other words, we want to show that the CSP with $3t+1$ variables and all domains $[0\,..\,3t]$, which asks for a mapping $u \mapsto u'$ such that $R_{it}(u,v)$ implies $R_{jt}(u',v')$, is unsatisfiable. The path $3i \longrightarrow 3i+1 \longrightarrow 3i+2 \longrightarrow 3i+3$ in R_{it} must map into the path $3j \longrightarrow 3j+1 \longrightarrow 3j+2 \longrightarrow 3j+3$ in R_{jt}; so we have $(3i)' = 3j$, $(3i+1)' = 3j+1$, $(3i+2)' = 3j+2$, $(3i+3)' = 3j+3$.

Suppose $i > j$, and let $x \le 3i$ be maximum such that $x' = x - 3(j-i)$. Then we have $y' = x'+1$ when $y = x-1$, because $y' \ne x'-1$. If $x \bmod 3 = 1$, we have $R_{it}(y,x) \wedge \neg R_{jt}(y',x')$. If $x \bmod 3 = 2$, we have $R_{it}(z,y) \wedge \neg R_{jt}(z',y')$, where $z = y-1$. If $x \bmod 3 = 0$, we have $z' = x'$, hence $\neg R_{jt}((z-1)',z')$. The case $i > j$ is similar.

(c) Now there are $3t+3$ variables with domain $[0\,..\,3t]$. Enforcing domain consistency leads to the unique solution $u' = u - 2[u \ge 3i+3]$.

[Exercises 363–365 were inspired by T. Feder and M. Y. Vardi, *SICOMP* **28** (1998), 57–104. Page 71 claims that a digraph G homomorphic to both R_{1t} and R_{2t} is homomorphic also to R_{0t}; but $G = R_{110111}$ is a counterexample.]

366. Let $\mathcal{P}$ be the given CSP. Remove from the domain of w every value what doesn't have support from v; and let C' be the CSP that results by also removing variable v. All solutions to $\mathcal{P}$ are obtainable from the solutions to C', by assigning all values of v that are compatible with w. (Now C' may, in turn, be reducible to C'', and so on. If the constraint graph of $\mathcal{P}$ is a tree, everything will therefore reduce to a trivial CSP with a single variable.) [See D. Sabin and E. Freuder, *LNCS* **1330** (1997), 167–181.]

367. The first part is easy. (Hence Theorem I implies, in particular, that 2SAT problems are solvable in linear time.) Answers to the second part are '$vw \notin \{ab\}$' (six cases); '$vw \notin \{0b, 1b'\}$ where $b \ne b'$' (six cases); '$vw \notin \{a0, a'1, a''2\}$ where $\{a, a', a''\} = \{0,1\}$' (six cases).

368. True. If R and R' are represented by 0–1 matrices, the intersection $R'' = R \cap R'$ is represented by the pointwise product matrix $R''_{ij} = R_{ij}R'_{ij}$. And if this product has at least two nonzero rows and at least two nonzero columns, it's not hard to see that complete $\cap$ complete $=$ complete; complete $\cap$ correspondence $=$ correspondence;

scopes
auxiliaries
math versus CS
CS versus math
inefficiency
polynomial time
domain consistency
Feder
Vardi
support
Sabin
Freuder
2SAT
0–1 matrices
pointwise product matrix

complete ∩ two-fan = two-fan; correspondence ∩ two-fan = correspondence; two-fan ∩ two-fan = complete or correspondence or two-fan.

369. $D_w = 01234 \cap 0234 \cap 0124$ (where '01234' is shorthand for '$\{0,1,2,3,4\}$'); $D_x = 01234 \cap 0234 \cap 013$; $D_y = 012345 \cap 145 \cap 1234$; $D_z = 012345 \cap 145 \cap 01345$.

370. (a) We assign to a variable the first time we choose a sink that contains it. If $v_j \leftarrow a'_j$ was previously assigned, (140) yields a path in G from (v_j, a'_j) to (v_i, a'_i) for some $a'_i \neq a_i$. So we must already have deleted the component containing (v_i, a'_i).

(b) Suppose $v \leftarrow a$ is assigned before $w \leftarrow b$. If the constraint is violated, there's an arc $(v,a) \longrightarrow (w,b')$ in G for some $b' \neq b$. The component containing (w,b') must therefore have been chosen before we assigned $v \leftarrow a$.

(c) The vertices of a solution all appear in good components of $\widehat{G}$. Consequently we'll assign something to every vertex. Now apply (b).

371. Notice that this representation scheme has an elegant recursive description: If p is a pointer, it represents the set $S(p) = \emptyset$ if $p = \perp$; otherwise it represents $S(p) = \{\texttt{INFO}(p)\} \cup S(\texttt{LINK}(p))$, where $\texttt{INFO}(p) < \texttt{INFO}(\texttt{LINK}(p))$.

We use a subroutine 'align(p,q)', which finds the smallest element where $S(p)$ and $S(q)$ agree, or ∞ if they have no common element: While $\texttt{INFO}(p) \neq \texttt{INFO}(q)$, set $p \leftarrow \texttt{LINK}(p)$ if $\texttt{INFO}(p) < \texttt{INFO}(q)$, otherwise set $q \leftarrow \texttt{LINK}(q)$.

To compute the intersection we use $m+1$ pointer variables, initializing them by setting $p \leftarrow Y$ and $p_j \leftarrow \texttt{LINK}(X_j)$ for $1 \le j \le m$. Cyclically align$(p_{1+((j-1) \bmod m)}, p_{1+(j \bmod m)})$ for $j = 1, 2, \ldots,$ until m consecutive alignments cause no further change. (At this point $\texttt{INFO}(p_j)$ is constant for $1 \le j \le m$.) If $\texttt{INFO}(p_1) < \infty$, set $\texttt{LINK}(p) \Leftarrow \texttt{AVAIL}$, $p \leftarrow \texttt{LINK}(p)$, $\texttt{INFO}(p) \leftarrow \texttt{INFO}(p_1)$, $p_j \leftarrow \texttt{LINK}(p_j)$ for $1 \le j \le m$, and resume the cyclic alignment loop. Otherwise set $\texttt{LINK}(p) \leftarrow \perp$ and terminate.

372. The culprit is third from the right and third from the bottom.

373. (a) No; consider $(x,y,z) = (3,1,2)$ and $(3,2,1)$. (b) Yes, by (141).

374. $F(x_1,\ldots,x_t) = [f(x_1,\ldots,x_t) \ge \alpha]$, $G(x) = [g(x) > \beta]$ are monotone functions.

375. Theorem 7.1.1H says that conjunctions of Horn clauses are equivalent to *min*-closed relations on $\{0,1\}$ if $0 =$ false and $1 =$ true. So they're equivalent to *max*-closed relations on $\{0,1\}$ if we stand on our heads and let $0 =$ true and $1 =$ false.

[P. G. Jeavons and M. C. Cooper have shown [*Artif. Int.* **79** (1995), 327–339] that, over *any* domain, a constraint is max-closed if and only if it is a conjunction of constraints '$(x_1 \ge a_1)$ or $\cdots$ or $(x_t \ge a_t)$ or $(x \le b)$', a special case of (142).]

376. If constraint c involves the variables $v_1, \ldots, v_k$, we know that it contains tuples $a_{11}a_{12}\ldots a_{1k}, \ldots, a_{k1}a_{k2}\ldots a_{kk}$ such that $a_{ij} \in D_{v_j}$ and $a_{jj} = \max D_{v_j}$, for all i and j. Therefore, by the max-closed property, it also contains the tuple $a_{11}a_{22}\ldots a_{kk}$.

377. (a) Reflexive: $x \preceq x$. Antisymmetric: If $x \preceq y$ and $y \preceq x$ then $y = x \vee y = y \vee x = x$. Transitive: If $x \preceq y \preceq z$ then $x \vee z = x \vee (y \vee z) = (x \vee y) \vee z = y \vee z = z$; that is, $x \preceq z$.

(b) If $x \preceq z$ and $y \preceq z$ then $(x \vee y) \vee z = x \vee (y \vee z) = x \vee z = z$. So $x \vee y \preceq z$.

(c) Yes. It's obviously idempotent and commutative. Let $t = (x \vee y) \vee z$ and $u = x \vee (y \vee z)$. We have $x \preceq t$, $y \preceq t$, $z \preceq t$; hence $y \vee z \preceq t$, and $u \preceq t$. A similar proof shows that $t \preceq u$. Therefore, by antisymmetry, we have $t = u$ (associativity).

378. When domain consistency has been achieved, every $a \in D_v$ has support in every constraint that involves v. (Notice incidentally that every unary relation is max-closed, but only certain unary relations are $\vee$-closed. All of the unary relations of $\mathcal{P}$ are $\vee$-closed, and so are the unary projections of $\mathcal{P}$'s k-ary relations.) It follows that $\bigvee\{a \mid a \in D_v\} \in D_v$, and we can argue as in exercise 376.

two-fan
recursive
AVAIL
min-closed
Jeavons
Cooper
unary relations

[P. Jeavons, D. Cohen, and M. Gyssens, *LNCS* **976** (1995), 276–291, Theorem 16.]

379. Let $\vee$ be componentwise max on tuples. For example, suppose v and c are variables of $\mathcal{P}^*$, where c is a constraint of $\mathcal{P}$ between u, v, and w. The domain of variable c in $\mathcal{P}^*$ is the set of triples $a_1a_2a_3$ that are in $\mathcal{P}$'s relation c. The binary relation between v and c in $\mathcal{P}^*$ is satisfied when $v = a_2$ and $c = a_1a_2a_3$. This binary relation is $\vee$-closed; for if $(a_2, a_1a_2a_3)$ and $(a'_2, a'_1a'_2a'_3)$ both satisfy it, so does $(a_2 \vee a'_2,\ a_1a_2a_3 \vee a'_1a'_2a'_3) = (\max(a_2, a'_2),\ \max(a_1, a'_1)\max(a_2, a'_2)\max(a_3, a'_3))$.

380. For each a_1 with $0 \le a_1 < d_1$, extend $a_1 \ldots a_k$ to $a_1 \ldots a_k a_{k+1}$ for each a_{k+1} with $\max\{l_1, \ldots, l_k\} \le a_{k+1} \le \min\{r_1, \ldots, r_k\}$.

381. If, say, $a_1 = a_2$, so that $\langle a_1a_2a_3\rangle = a_1$, the condition says that any entry between two 1s in the same row of a matrix must be 1. Similarly, $b_1 = b_2$ tells us that columns must be convex. Finally, if the a's and b's are distinct, the median-closed condition can be seen to be equivalent to forcing the 1s to be kingwise connected. We cannot, for example, have $01 \in R$, $12 \in R$, and $20 \in R$ without also having $11 \in R$. [See P. Jeavons, D. Cohen, and M. C. Cooper, *Artif. Intelligence* **101** (1998), 262–263, Example 4.7.]

382. Path consistency is achieved by repeatedly operating on sets of three variables, say $\{u, v, w\}$. We first form the join $R_{uvw} = R_{uw} \bowtie R_{vw}$; then we project R_{uvw} onto u and v, and intersect that with R_{uv}. A polymorphism for R_{uv}, R_{uw}, and R_{vw} is also a polymorphism for R_{uvw} and its subsequent projection and intersection. Exercise 381 shows that $\langle abc\rangle$ is a polymorphism. Therefore all relations are CRC *after* achieving path consistency if they all were initially CRC.

383. If at every branch we choose, for example, the median of an active variable's current domain, and if there's no solution with that median value, then all solutions must lie on the same side of that incorrect value.

384. $A^+_{mn} = \sum_{j,k}\binom{m}{j}\binom{n}{k}A_{jk}$; $A_{mn} = \sum_{j,k}\binom{m}{j}\binom{n}{k}(-1)^{m+n-j-k}A^+_{jk}$, by Eq. 1.2.6–(33). Notice a significant technicality: We always have $A_{0n} = 0$ when $n > 0$, because a $0 \times n$ matrix has n all-zero columns. On the other hand, A_{00} might be either 1 or 0, depending on whether or not $\mathcal{A}$ contains the "null" 0×0 matrix. The set $\mathcal{A}^+$ contains all-zero matrices if and only if $A_{00} = 1$.

(The bivariate exponential generating functions $A(w,z) = \sum_{m,n\ge 0} A_{mn}\frac{w^m}{m!}\frac{z^n}{n!}$, $A^+(w,z) = \sum_{m,n\ge 0} A^+_{mn}\frac{w^m}{m!}\frac{z^n}{n!}$ are related by the identity $A^+(w,z) = e^{w+z}A(w,z)$.)

385. The matrix of a reduced implicational relation is either a permutation matrix, or an all-1s matrix, or has n 1s in some row i and m 1s in some column j, zeros elsewhere. Thus we have $P_{mn} = n![m=n] + 1 + mn$ when $m \ge 2$ and $n \ge 2$; $P_{m1} = P_{1n} = 1$ when $mn > 0$; and $P_{m0} = [m=0]$, $P_{0n} = [n=0]$. Exercise 384 now gives

P_{mn}	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$m=1$	1	1	1	1	1	1
$m=2$	1	7	7	9	11	13
$m=3$	1	7	16	13	16	19
$m=4$	1	9	13	41	21	25
$m=5$	1	11	16	21	146	31
$m=6$	1	13	19	25	31	757

P^+_{mn}	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$m=1$	2	4	8	16	32	64
$m=2$	4	16	46	114	264	592
$m=3$	8	46	155	418	1013	2326
$m=4$	16	114	418	1202	3046	7174
$m=5$	32	264	1013	3046	8107	19924
$m=6$	64	592	2326	7174	19924	51856

The generating function $P(w,z)$ is $1 + e^{wz} + (e^{w+z} - e^w - e^z)(1 + wz)$. Therefore the coefficients of $P^+(w,z) = e^{w+z}P(w,z)$ can be evaluated by the methods of Section 4.7.

386. Sequences A272644 and A099594 of the OEIS reveal that Q^+_{mn} is a so-called *poly-Bernoulli number*, introduced by M. Kaneko in *Journal de théorie des nombres de Bordeaux* **9** (1997), 221–228, and that the same numbers arise in many different contexts. For example, I. Kaplansky and J. Riordan [*Duke Math. J.* **13** (1946), 259–268]

Jeavons
Cohen
Gyssens
kingwise connected
Jeavons
Cohen
Cooper
Path consistency
join
projection of a relation
intersection of relations
polymorphism
median
$0 \times n$ matrix
null matrix
all-zero matrices
bivariate exponential generating functions
permutation matrix
generating function
OEIS
poly-Bernoulli number
Kaneko
Kaplansky
Riordan

showed that Q_{mn} is the number of permutations $p_1 \ldots p_{m+n}$ of $\{1,\ldots,m+n\}$ such that $j-m < p_j < j+n$ for all j. K. Vesztergombi [*Studia Scientiarum Mathematicarum Hungarica* **9** (1974), 181–185] showed that Q^+_{mn} enumerates the permutations with $j-m \le p_j \le j+n$, and that the generating function $Q(m,n)$ is $1/(e^w+e^z-e^{w+z})$. The number of ways to assign directions to the complete bipartite graph $K_{m,n}$ so that no oriented cycles arise is Q^+_{mn}; and so is the number of "lonesum matrices," namely the $m\times n$ matrices of 0s and 1s that are uniquely determined by their row sums and column sums. The same numbers enumerate "parades" of m boys and n girls, as well as a number of other types of combinatorial arrangements(!); see the author's online note "Parades and poly-Bernoulli bijections" (2024), 29 pages. We have the explicit formulas

bounded permutations
Vesztergombi
complete bipartite graph
directed acyclic graphs
acyclic orientations
lonesum matrices
row sums
column sums
parades
author

$$Q_{mn} = \sum_{k\ge0} k!^2 \left\{ {m \atop k} \right\} \left\{ {n \atop k} \right\}; \qquad Q^+_{mn} = \sum_{k\ge0} k!^2 \left\{ {m+1 \atop k+1} \right\} \left\{ {n+1 \atop k+1} \right\}.$$

Q_{mn}	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$m=1$	1	1	1	1	1	1
$m=2$	1	5	13	29	61	125
$m=3$	1	13	73	301	1081	3613
$m=4$	1	29	301	2069	11581	57749
$m=5$	1	61	1081	11581	95401	673261
$m=6$	1	125	3613	57749	673261	6487445

Q^+_{mn}	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$m=1$	2	4	8	16	32	64
$m=2$	4	14	46	146	454	1394
$m=3$	8	46	230	1066	4718	20266
$m=4$	16	146	1066	6902	41506	237686
$m=5$	32	454	4718	41506	329462	2441314
$m=6$	64	1394	20266	237686	2441314	22934774

387. Any $m\times n$ reduced connected row convex matrix can be annotated as shown below, by writing the number of topmost zeros at the top of each column and the number of bottommost zeros at the bottom. The top numbers decrease monotonically to 0, after which they increase monotonically; we also place a bar over the first 0 and all subsequent entries. A similar pattern occurs at the bottom, but with bars *underneath*. We can regard the example matrix as a path from ${4 \atop 6}$ to ${2 \atop 6}$ to ${1 \atop 6}$ to ${1 \atop 5}$ to ${\overline{0} \atop 5}$ to ${\overline{0} \atop 5}$ to ${\overline{3} \atop 5}$ to $\cdots$ to ${\overline{8} \atop \underline{1}}$ to ${\overline{8} \atop \underline{1}}$ to ${\overline{8} \atop \underline{2}}$.

$$\begin{array}{c}
\scriptstyle 4\,2\,1\,1\,\overline{0}\,\overline{0}\,\overline{3}\,\overline{4}\,\overline{6}\,\overline{6}\,\overline{8}\,\overline{8}\,\overline{8}\,\overline{8} \\
00001100000000 \\
00111100000000 \\
01111100000000 \\
01111110000000 \\
11111111000000 \\
00011111000000 \\
00000000110000 \\
00000000010000 \\
00000000001111 \\
00000000001110 \\
00000000001000 \\
\scriptstyle 6\,6\,6\,5\,5\,5\,5\,5\,4\,3\,\underline{0}\,\underline{1}\,\underline{1}\,\underline{2}
\end{array}$$

Given the total number of rows, m, we shall set up simultaneous recurrence relations between variables of four types: X^i_j, $\overline{X}^i_j$, $\underline{X}^i_j$, $\underline{\overline{X}}^i_j$. In all cases we have $i\ge0$, $j\ge0$, and $i+j<m$. Furthermore $i>0$ in types X and $\underline{X}$; $j>0$ in types X and $\overline{X}$. (Hence there are $\binom{m-1}{2}$ variables of type X^i_j, $\binom{m}{2}$ variables of type $\overline{X}^i_j$ and type $\underline{X}^i_j$, $\binom{m+1}{2}$ variables of type $\underline{\overline{X}}^i_j$.) Each variable has a parameter n; for example, $\overline{X}^3_5(n)$ is the number of $m\times n$ matrices whose final column has the annotations $\overline{3}$ and 5.

The initial conditions are $X^i_j(1)=1$; $\overline{X}^i_j(1)=[i=0]$; $\underline{X}^i_j(1)=[j=0]$; $\underline{\overline{X}}^i_j(1)=[i=j=0]$. The transition relations are

$$X^i_j(n{+}1) = \sum\{X^{i'}_{j'}(n) \mid i'\ge i, j'\ge j\};$$
$$\overline{X}^i_j(n{+}1) = [i=0]\sum\{X^{i'}_{j'}(n) \mid i'\ge 1, j'\ge j\} + \sum\{\overline{X}^{i'}_{j'}(n) \mid i'\le i, j\le j'\le m-i\};$$
$$\underline{X}^i_j(n{+}1) = [j=0]\sum\{X^{i'}_{j'}(n) \mid i'\ge i, j'\ge 1\} + \sum\{\underline{X}^{i'}_{j'}(n) \mid i\le i'\le m-j, j'\le j\};$$
$$\underline{\overline{X}}^i_j(n{+}1) = [i=j=0]\sum\{X^{i'}_{j'}(n) \mid i'\ge 1, j'\ge 1\} + \sum\{\underline{\overline{X}}^{i'}_{j'}(n) \mid i'\le i, j'\le j\} +$$
$$[i=0]\sum\{\underline{X}^{i'}_{j'}(n) \mid 1\le i'\le m-j, j'\le j\} + [j=0]\sum\{\overline{X}^{i'}_{j'}(n) \mid i'\le i, 1\le j'\le m-i\};$$

and $R_{mn} = \sum_{i,j}\underline{\overline{X}}^i_j(n)$. Variables are, of course, omitted from these formulas when they don't exist. For example, some of the relations when $m=3$ are $X^1_1(n{+}1)=X^1_1(n)$; $\underline{X}^1_0(n{+}1) = X^1_1(n) + \underline{X}^1_0(n) + \underline{X}^2_0(n)$; $\underline{\overline{X}}^2_0(n{+}1) = \overline{X}^0_1(n) + \overline{X}^1_1(n) + \underline{\overline{X}}^0_0(n) + \underline{\overline{X}}^1_0(n) +$

$\underline{X}^2_0(n)$; $\underline{\overline{X}}^0_0(n{+}1) = X^1_1(n) + \overline{X}^0_2(n) + \overline{X}^0_1(n) + \underline{X}^2_0(n) + \underline{X}^1_0(n) + \underline{\overline{X}}^0_0(n)$.

R_{mn}	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$m=1$	1	1	1	1	1	1
$m=2$	1	7	17	31	49	71
$m=3$	1	17	90	284	687	1411
$m=4$	1	31	284	1398	4861	13555
$m=5$	1	49	687	4861	23020	83858
$m=6$	1	71	1411	13555	83858	386774

R^+_{mn}	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$m=1$	2	4	8	16	32	64
$m=2$	4	16	56	176	512	1408
$m=3$	8	56	289	1231	4623	15887
$m=4$	16	176	1231	6655	30553	125197
$m=5$	32	512	4623	30553	166186	790250
$m=6$	64	1408	15887	125197	790250	4283086

generating function
polyomino
convex polyominoes
parallominoes
staircase polygons
Bousquet-Mélou
OEIS
Narayana numbers
OEIS

[Alternatively, the ordinary generating function $r(w,z) = \sum_{m,n} R_{mn} w^m z^n$ can be determined by using known results about polyomino enumeration. Let $c(w,z)$ be the generating function for convex polyominoes, and $\underline{c}(w,z)$ for the special case where the convex polyomino touches the lower left corner of its bounding box; also let $\overline{\underline{c}}(w,z)$ be the generating function for the even more special case of parallominoes or "staircase polygons" (exercise 7.2.2.1–303). Then $r(w,z) = c(w,z) + 2\underline{c}(w,z)^2/(1-\overline{\underline{c}}(w,z))$; and

$$\overline{\underline{c}}(w,z) = (1-w-z-\sqrt{\Delta})/2, \quad \text{where } \Delta = (w-z)^2 - 2w - 2z + 1;$$
$$\underline{c}(w,z) = wz/\sqrt{\Delta}\,;$$
$$c(w,z) = 1 + wz((1-w)^3 + (1-z)^3 - 1 - wz((w-z)^2 + w + z - 5))/\Delta^2 - 4w^2z^2/\Delta^{3/2}.$$

See Mireille Bousquet-Mélou, *Journal of Physics* **A25** (1992), 1925–1934, 1935–1944; also OEIS A001263, A008409, A324009. Incidentally, $\overline{\underline{c}}(w,z) = (\overline{\underline{c}}(w,z)+w)(\overline{\underline{c}}(w,z)+z)$ generates the Narayana numbers of exercise 2.3.4.6–3. For fixed m, R_{mn} is a polynomial in n, asymptotic to $(2^{2m-2}m/(2m-2)! - 2/((m-1)!(m-2)!))n^{2m-2}$.]

388. These relations are the special case of exercise 387 where column 1 is annotated with $\bar{\scriptstyle 0}$ and column n is annotated with $\underline{\scriptstyle 0}$. So $X^i_j(n) = \underline{X}^i_j(n) = 0$, and we need only the variables $\overline{X}^i_j(n)$ and $\underline{\overline{X}}^i_0(m)$; $S_{mn} = \sum_i \underline{\overline{X}}^i_0(n)$.

S_{mn}	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$m=1$	1	1	1	1	1	1
$m=2$	1	4	8	13	19	26
$m=3$	1	8	29	73	151	276
$m=4$	1	13	73	266	749	1781
$m=5$	1	19	151	749	2762	8321
$m=6$	1	26	276	1781	8321	31004

S^+_{mn}	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$m=1$	2	4	8	16	32	64
$m=2$	4	13	38	104	272	688
$m=3$	8	38	147	506	1612	4856
$m=4$	16	104	506	2103	7887	27477
$m=5$	32	272	1612	7887	34088	134825
$m=6$	64	688	4856	27477	134825	597539

[The generating function $\sum_{m,n} S_{mn} w^m z^n$ is $1/(1-\overline{\underline{c}}(w,z))$, OEIS A100754.]

389. (a) Implicational relations are closed under reflection and/or rotation. So we need only show one example in each equivalence class, and note the size of that class:

$$\begin{matrix}100\\010\\001\end{matrix}\,(2),\quad \begin{matrix}100\\001\\010\end{matrix}\,(4);\quad \begin{matrix}111\\100\\100\end{matrix}\,(4),\quad \begin{matrix}111\\010\\010\end{matrix}\,(4),\quad \begin{matrix}010\\111\\010\end{matrix}\,(1);\quad \begin{matrix}111\\111\\111\end{matrix}\,(1).$$

(b) Now the only symmetry is transposition (interchanging rows and columns):

$$\begin{matrix}100\\010\\001\end{matrix}\,(1);\ \begin{matrix}110\\010\\001\end{matrix}\,(2),\ \begin{matrix}110\\001\\001\end{matrix}\,(2),\ \begin{matrix}100\\001\\011\end{matrix}\,(1),\ \begin{matrix}010\\110\\001\end{matrix}\,(1),\ \begin{matrix}100\\011\\001\end{matrix}\,(2);\ \begin{matrix}001\\001\\111\end{matrix}\,(1),\ \begin{matrix}001\\101\\011\end{matrix}\,(2),\ \begin{matrix}001\\111\\001\end{matrix}\,(2),\ \begin{matrix}010\\110\\011\end{matrix}\,(2),\ \begin{matrix}100\\001\\111\end{matrix}\,(2),$$

$$\begin{matrix}100\\010\\111\end{matrix}\,(2),\ \begin{matrix}100\\011\\011\end{matrix}\,(1),\ \begin{matrix}100\\100\\111\end{matrix}\,(2),\ \begin{matrix}100\\101\\011\end{matrix}\,(2),\ \begin{matrix}100\\110\\011\end{matrix}\,(2),\ \begin{matrix}100\\111\\001\end{matrix}\,(2),\ \begin{matrix}110\\110\\001\end{matrix}\,(1);\ \begin{matrix}001\\011\\111\end{matrix}\,(1),\ \begin{matrix}001\\101\\111\end{matrix}\,(2),\ \begin{matrix}001\\111\\011\end{matrix}\,(2),\ \begin{matrix}010\\110\\111\end{matrix}\,(2),$$

$$\begin{matrix}010\\111\\011\end{matrix}\,(1),\ \begin{matrix}100\\011\\111\end{matrix}\,(2),\ \begin{matrix}100\\101\\111\end{matrix}\,(2),\ \begin{matrix}100\\110\\111\end{matrix}\,(2),\ \begin{matrix}100\\111\\011\end{matrix}\,(2),\ \begin{matrix}101\\001\\111\end{matrix}\,(1),\ \begin{matrix}101\\101\\011\end{matrix}\,(2),\ \begin{matrix}101\\111\\001\end{matrix}\,(2),\ \begin{matrix}110\\110\\011\end{matrix}\,(2);\ \begin{matrix}001\\111\\111\end{matrix}\,(2),\ \begin{matrix}010\\111\\111\end{matrix}\,(2),$$

$$\begin{matrix}100\\111\\111\end{matrix}\,(2),\ \begin{matrix}101\\011\\111\end{matrix}\,(1),\ \begin{matrix}101\\101\\111\end{matrix}\,(2),\ \begin{matrix}101\\111\\011\end{matrix}\,(2),\ \begin{matrix}110\\111\\011\end{matrix}\,(1),\ \begin{matrix}111\\111\\001\end{matrix}\,(2);\ \begin{matrix}011\\111\\111\end{matrix}\,(1),\ \begin{matrix}101\\111\\111\end{matrix}\,(2),\ \begin{matrix}110\\111\\111\end{matrix}\,(2);\ \begin{matrix}111\\111\\111\end{matrix}\,(1).$$

(c) Here, as in (a), we can exploit the eightfold symmetries of the square:

$$\begin{smallmatrix}100\\010\\001\end{smallmatrix}(2);\ \begin{smallmatrix}110\\010\\001\end{smallmatrix}(8),\ \begin{smallmatrix}110\\001\\001\end{smallmatrix}(4),\ \begin{smallmatrix}100\\011\\010\end{smallmatrix}(4);\ \begin{smallmatrix}111\\010\\010\end{smallmatrix}(4),\ \begin{smallmatrix}111\\001\\001\end{smallmatrix}(4),\ \begin{smallmatrix}110\\110\\001\end{smallmatrix}(4),\ \begin{smallmatrix}110\\011\\010\end{smallmatrix}(8),\ \begin{smallmatrix}110\\011\\001\end{smallmatrix}(4),\ \begin{smallmatrix}110\\010\\011\end{smallmatrix}(4),$$

$$\begin{smallmatrix}010\\111\\010\end{smallmatrix}(1);\ \begin{smallmatrix}111\\110\\100\end{smallmatrix}(4),\ \begin{smallmatrix}111\\110\\010\end{smallmatrix}(8),\ \begin{smallmatrix}110\\111\\010\end{smallmatrix}(4),\ \begin{smallmatrix}100\\111\\011\end{smallmatrix}(8);\ \begin{smallmatrix}111\\111\\100\end{smallmatrix}(8),\ \begin{smallmatrix}111\\111\\010\end{smallmatrix}(4),\ \begin{smallmatrix}011\\111\\110\end{smallmatrix}(2);\ \begin{smallmatrix}111\\111\\110\end{smallmatrix}(4);\ \begin{smallmatrix}111\\111\\111\end{smallmatrix}(1).$$

(d) And this case allows central symmetry, and reflection about either diagonal:

$$\begin{smallmatrix}100\\010\\001\end{smallmatrix}(1);\ \begin{smallmatrix}110\\010\\001\end{smallmatrix}(4),\ \begin{smallmatrix}110\\001\\001\end{smallmatrix}(2);\ \begin{smallmatrix}111\\001\\001\end{smallmatrix}(2),\ \begin{smallmatrix}110\\110\\001\end{smallmatrix}(2),\ \begin{smallmatrix}110\\011\\001\end{smallmatrix}(2),\ \begin{smallmatrix}110\\010\\011\end{smallmatrix}(2);\ \begin{smallmatrix}100\\110\\111\end{smallmatrix}(2),\ \begin{smallmatrix}100\\111\\011\end{smallmatrix}(4);\ \begin{smallmatrix}100\\111\\111\end{smallmatrix}(4),\ \begin{smallmatrix}110\\111\\011\end{smallmatrix}(1);\ \begin{smallmatrix}110\\111\\111\end{smallmatrix}(2);\ \begin{smallmatrix}111\\111\\111\end{smallmatrix}(1).$$

390. BDD methods can be used to find the respective values (122, 898, 6086, 13094, 165534, 468732, 13432658, 798255356, 114446643198) for lmn = (222, 223, 224, 233, 234, 333, 334, 344, 444). (But those methods quickly run out of space. No feasible way is currently known to compute, say, Q^+_{999}. Filip Stappers has discovered that the generating functions $G_{lm}(z) = \sum_{n\ge 0} Q^+_{lmn} z^n$ have a comparatively simple form when l and m are small; for example, $G_{22}(z) = (1-z)(1-8z^2)/((1-6z)(1-4z)(1-3z)(1-2z))$.)

391. An implicational constraint that's complete is closed under any "conservative" operation (that is, under any operation f that takes $f(x,y,z)$ into one of $\{x,y,z\}$).

A two-fan '$v = a$ or $w = b$' is closed under Δ only if we have, say, $\Delta(a,a',a) = a$ or $\Delta(b',b,b'') = b$, for all a, a', b, b', b'' in D. Consequently Δ must be a majority function, satisfying $\Delta(a,a,x) = \Delta(a,x,a) = \Delta(x,a,a) = a$ for all a and x. Conversely, every majority function is a polymorphism of all two-fans.

Finally, a correspondence relation on a domain D with $|D| \ge 3$ is invariant under Δ if and only if there's an index j such that $\Delta(x_1,x_2,x_3) = x_j$ whenever $\{x_1,x_2,x_3\}$ are distinct (that is, whenever there is no majority value).

So there are three solutions [all of which, incidentally, are represented by `001122` in Fig. A–17]. When $j = 3$, the function $\Delta(x,y,z)$ is $(x \ne y?\ z\colon x)$, which is called the "dual discriminator" because of its similarity to the discriminator in exercise 426.

392. μ_d preserves every such relation. Conversely, if $R(x_1,\ldots,x_m)$ is preserved by μ_d, so is $\widehat{R}(x_1,\ldots,x_m) = R((x_1-a_1) \bmod d, \ldots, (x_m-a_m) \bmod d)$, for any constants a_1, $\ldots$, a_m. Therefore if R is nonempty, we can assume that $\widehat{R}(0,\ldots,0)$ is true. Then $\widehat{R}(x_1,\ldots,x_m)$ and $\widehat{R}(y_1,\ldots,y_m)$ imply $\widehat{R}((x_1\pm y_1) \bmod d, \ldots, (x_m\pm y_m) \bmod d)$. So we can find linearly independent basis vectors $\alpha_j = (\alpha_{j1},\ldots,\alpha_{jm})$ such that $\widehat{R}(x_1,\ldots,x_m) \iff (x_1,\ldots,x_m) \equiv t_1\alpha_1 + \cdots + t_k\alpha_k$ (modulo d) for integers t_1, $\ldots$, t_k. We can renumber the coordinates and convert the basis vectors to Smith normal form, so that α_{ii} divides α_{jj} and $\alpha_{ij} = \alpha_{ji} = 0$ when $1 \le i < j \le k$; α_{kk} divides d; and α_{ii} divides α_{ij} when $1 \le i \le k < j \le m$. (See exercise 7–138.) Thus $k < m$; and $\widehat{R}$ (hence R) is a conjunction of k unary relations and $m-k$ $(k+1)$-ary relations that have the specified form.

For example, we might have $d = 36$, $m = 4$, $(a_1,a_2,a_3,a_4) = (3,1,4,1) \in R$, and $\tau_0 = (12,20,18,14) \in \widehat{R}$. Dividing by 7 mod 36 (i.e., multiplying by 31) tells us that $\tau_1 = 31\tau_0 = (12,8,18,2) \in \widehat{R}$. Maybe also $\tau_2 = (12,12,30,6) \in \widehat{R}$; hence $\tau_3 = \tau_2 - 3\tau_1 = (12,24,12,0) \in \widehat{R}$, and $\tau_4 = \tau_1 - \tau_3 = (0,20,6,2) \in \widehat{R}$.

Suppose now that all elements of $\widehat{R}$ are integer combinations, $t_1\tau_3 + t_2\tau_4$, mod 36. (The permuted coordinates $\alpha_1 = (2,0,20,6)$ and $\alpha_2 = (0,12,24,12)$ are Smithian.) Then $\widehat{R}(x_1,x_2,x_3,x_4)$ is '$18x_4 \equiv 0$ and $3x_1 \equiv 0$ and $x_2 \equiv 24t_1 + 20t_2 = 2x_1 + 10x_4$ and $x_3 \equiv 12t_1 + 6t_2 = x_1 + 3x_4$ (modulo 36)'. And the original relation $R(x_1,x_2,x_3,x_4)$ is $\widehat{R}((x_1+3) \bmod 36, (x_2+1) \bmod 36, (x_3+4) \bmod 36, (x_4+1) \bmod 36)$.

symmetries
central symmetry
BDD
Stappers
generating functions
conservative
two-fan
majority function
distinct
dual discriminator
discriminator
linearly independent
basis vectors
Smith normal form
echelon form

393. (a) There's a clause $([a_1\ldots a_k]\vee[b_1\ldots b_k])$ whenever $a_1\vee b_1=\cdots=a_k\vee b_k=1$, $a_1\ldots a_k\neq 1\ldots 1\neq b_1\ldots b_k$, and $a_1\ldots a_k<b_1\ldots b_k$. So there are $(3^k-2\cdot 2^k+1)/2=\lceil 3^k/2\rceil-2^k$ of them. [We can use BDDs to count the number of polymorphisms, when $k\le 7$; but that method becomes infeasible for larger k. See OEIS A051185.]

(b) When 0 and 1 are interchanged, a function $f(x_1,\ldots,x_k)$ changes to its dual, $\overline{f(\bar x_1,\ldots,\bar x_k)}$. For example, (146) becomes $(\overline{[110]}\vee\overline{[001]})\wedge\cdots\wedge(\overline{[010]}\vee\overline{[001]})$.

(c) When a Boolean polymorphism represents a family of sets, '$[x_1\ldots x_k]=1$' means that the set $S_x=\{j\mid x_j=1\}$ is in the family. If $[x_1\ldots x_k]=1$ in the polymorphism for NAND, we have $[y_1\ldots y_k]=0$ for all sets $S_y=\{j\mid y_j=1\}$ that are disjoint from S_x, because of the clause $(\overline{[x_1\ldots x_k]}\vee\overline{[y_1\ldots y_k]})$. Thus there's one polymorphism of NAND for each family of mutually intersecting sets.

394. Let R have $f_k(R)$ k-ary polymorphisms. (This is R's "spectrum"; see exercise 419.) When R is binary, it has the same polymorphisms as its *converse*, R^T (obtained by interchanging coordinates); hence $f_k(R)=f_k(R^T)$. When R is Boolean, the polymorphisms of its *dual*, R^D (obtained by interchanging 0 and 1) are the duals of its own polymorphisms; hence $f_k(R)=f_k(R^D)$.

In this problem R is a subset of $\{00,01,10,11\}$, and it also corresponds to a binary Boolean operation (see Table 7.1.1–1). Every Boolean function is a polymorphism of R when R is empty ('$\bot$'), or the full set ('$\top$'), or $\{00,11\}$ ('$\equiv$'); so $f_k(R)=2^{2^k}$ in those cases. Every Boolean function with $[1\ldots 1]=1$ works when $R=\{11\}$ (AND), or when $R=\{10,11\}$ ('$\llcorner$') or $\{01,11\}$ ('R'); so $f_k(R)=2^{2^k-1}$ in those cases (and dually also when $R=\{00\}$ (NOR), or $\{00,01\}$ ('$\bar\llcorner$'), or $\{00,10\}$ ('$\bar{\mathsf{R}}$')). When $R=\{01\}$ ('$<$') or $\{10\}$ ('$>$'), its polymorphisms are the operations with both $[0\ldots 0]=0$ and $[1\ldots 1]=1$; thus $f_k(R)=2^{2^k-2}$. The polymorphisms of $R=\{01,10\}$ (XOR) are the self-dual Boolean functions; hence $f_k(R)=2^{2^{k-1}}$. For $R=\{01,10,11\}$ and $\{00,01,10\}$, see exercise 393. And the remaining cases $R=\{00,01,11\}$ ('$\le$') and $\{00,10,11\}$ ('$\ge$') are the monotone Boolean functions, with $(f_1(R),\ldots,f_5(R))=(3,\,6,\,20,\,168,\,7581)$.

(This problem can be solved quickly by applying an XCC solver to the relevant indicator problem, using exercise 398. But it's best to solve small cases by hand, before using a computer to check one's reasoning.)

395. Formula (146) becomes a set of clauses like $([001]\vee[010]\vee[100])\wedge\cdots$, only the first of which isn't implied by the original binary version of (146). Thus the culprits must have $[001]=[010]=[100]=0$, $[011]=[101]=[110]=1$, and $[000]$ arbitrary.

396. True. (It's an immediate consequence of the definitions.)

397. Relation R_1 generates 27 constraints for $\mathcal{I}_3$, of which the first $\bigl(\text{from }\left(\begin{smallmatrix}0&0&1\\0&0&1\\0&0&1\end{smallmatrix}\right)\bigr)$ is '$[000][111]\in\{01,11\}$' (hence $[000]\neq 2$ and $[111]=1$). The second of those constraints $\bigl(\text{from }\left(\begin{smallmatrix}0&0&1\\0&0&1\\0&2&0\end{smallmatrix}\right)\bigr)$ is '$[000][002][110]\in\{001,020,111\}$' (hence $[110]\neq 2$); and so on.

Relations R_2 and R_3 each generate 27 further constraints. (We could, however, omit those from R_2, because R_2 is the projection of R_3 onto its second and third components.) The unary relation D_1 generates 8 constraints, because it has only 2 tuples ('0' and '1'). Those 8 state that $[000]\neq 2$, …, $[111]\neq 2$; so they add nothing new.

Problem $\mathcal{I}_3$ has four solutions, so there are four ternary polymorphisms. Three of them are, of course, the projections (151). The other is

$$f(x,y,z)\;=\;(x=y?\;x\colon\;x=z?\;x\colon\;y=z?\;y\colon\;0).$$

(Incidentally, f corresponds to `000000` in Fig. A–17.) Its existence implies that *any* CSP whose constraints all lie in the set $\Gamma=\{R_1,R_2,R_3,D_1\}$ is tractable.

BDDs
OEIS
dual
family of sets
spectrum
converse
dual
$\bot$
$\top$
AND
$\llcorner$
R
NOR
$\bar\llcorner$
$\bar{\mathsf{R}}$
$<$
$>$
XOR
self-dual
$\le$
$\ge$
monotone Boolean functions
indicator problem

(Relations R_1 and R_3 are decomposable into their binary projections. For example, $R_1 = \{00{*}, 02{*}, 11{*}\} \cap \{0{*}1, 0{*}0, 1{*}1\} \cap \{{*}01, {*}20, {*}11\}$. If we remap $(0,1,2) \mapsto (1,2,0)$, $f(x,y,z)$ becomes the median $\langle xyz\rangle$, and all four relations become CRC.)

398. Let there be d^k secondary items $x_1 \ldots x_k$ for all k-tuples with $x_j \in D$; their "colors" will be the values of the variables $[x_1 \ldots x_k]$. The primary items, for each $R \in \Gamma$, are $Ri_1 \ldots i_k$, when R has t m-tuples $\tau_1 \ldots \tau_t$ and $1 \le i_1, \ldots, i_k \le t$; they represent the matrices A in the definition of $\mathcal{I}_k(\Gamma)$.

The options for $Ri_1 \ldots i_k$ (see (150) and the online program INDICATOR-DLX) are as follows: Let $a_{l1} \ldots a_{lm} = \tau_{i_l}$ for $1 \le l \le k$, and $\alpha_j = a_{1j} \ldots a_{kj}$ for $1 \le j \le m$. (They are the rows and columns of A.) For each tuple $b_1 \ldots b_m \in R$, include no option if $\alpha_i = \alpha_j$ and $b_i \ne b_j$, for some $1 \le i < j \le m$. Otherwise include the option '$Ri_1 \ldots i_k \cup \bigcup\{\alpha_j{:}b_j \mid 1 \le j \le m \text{ and } \alpha_j \notin \{\alpha_1, \ldots, \alpha_{j-1}\}\}$'.

Note: Some of the secondary items might not occur in any options. Then the corresponding variable $[x_1 \ldots x_k]$ is unconstrained. With t unconstrained variables, the number of polymorphisms is d^t times the number of solutions to this XCC problem.

[R. Gault and P. Jeavons, in *Constraints* **9** (2004), 139–160, showed how to take advantage of symmetry to speed up the calculation. They identified several relations whose tractability status was unknown at the time, including (187) and (189).]

399. A polymorphism of R must be a monotone Boolean function. But it cannot have $([\alpha01], [\alpha10], [\alpha11]) = (0,0,1)$, for any binary string α. Thus $f(x_1, \ldots, x_k)$ is either the constant 1, or $\bigvee\{x_j \mid j \in S\}$, for some $S \subseteq \{1, \ldots, k\}$. (See Theorem 7.1.1Q.)

400. Since $[0^k] + [0^k] + [1^k] = 1$, we have $[0^k] = 0$ and $[1^k] = 1$. Moreover, $[\alpha] + [\bar\alpha] + [0^k] = 1$; hence f is self-dual. If $[\alpha] = [\beta] = 0$ and $\alpha \wedge \beta = 0^k$, then $[\alpha] + [\beta] + [\bar\alpha \wedge \bar\beta] = 1$; hence $[\alpha \vee \beta] = 0$. Consequently there's at least one unit vector with $[e_j] = 1$.

Suppose $[e_k] = 1$. Then $[\alpha 0] + [\bar\alpha 0] + [e_k] = 1$; hence $[\alpha 0] = 0$ for all α. So $[\bar\alpha 1] = 1$ for all α, and $f = \pi_k$. Similarly, $[e_j] = 1$ implies $f = \pi_j$.

Historical notes: I. Rosenberg, *Rocky Mountain Journal of Mathematics* **3** (1973), 631–639, introduced the term "strongly rigid" for what we are calling potency. He proved that exactly 14 Boolean ternary relations are potent for $\{0,1\}$, namely the proper subsets of $S_1 \cup S_2$ that contain either S_1 or S_2. He also proved that the "shortcut" graph of (159) is the smallest potent binary relation.

401. Letting $[xyz]$ denote $f(x,y,z)$, assume first that f is idempotent, namely that $[xxx] = x$ for all x. Then f is also conservative; that is, $[xyz] \in \{x,y,z\}$ for all x, y, and z (see answer 391). For example, $[001] = 2$ and $[222] = 2$ would imply $2 \ne 2$.

Suppose $[001] = 0$. Then $[110] \ne 0$, so $[110] = 1$. Also $[220] \ne 0$, so $[220] = 2$. Also $[221] \ne 1$; indeed, $[xxy] = x$, for all x and y. Similarly, $[010] = 0$ implies $[xyx] = x$, and $[100] = 0$ implies $[yxx] = x$. But $[012]$ can't be equal to $[100]$ or $[101]$ or $[220]$; so we can't have $[001] = [010] = [100] = 0$. At least one of $[001]$, $[010]$, $[100]$ must be 1.

In fact, exactly one must be 1. Assume, for example, that $[001] = 1$; then we have $[112] = 2$, $[221] = 1$, …, $[xxy] = y$. And $[010] = 1$ or $[100] = 1$ would imply $[221] \ne 1$.

Since $[001] = 1$ implies $[xxy] = y$, we can conclude that $[xyz] = z$ whenever x, y, z are distinct: $[100] = 0$ implies $[xyz] \ne [yxx] = x$; $[010] = 0$ implies $[xyz] \ne [yxy] = y$.

Finally, if π is any permutation of $\{0,1,2\}$ and $g(x,y,z)$ is any polymorphism of '$x \ne y$', so is $f(x,y,z) = g(x,y,z)\pi$. We can choose π so that f is idempotent, because $g(0,0,0) \ne g(1,1,1) \ne g(2,2,2) \ne g(0,0,0)$. We've proved that f is a projection; hence g is a permutation of a projection: $g(x_1,x_2,x_3) = x_j\pi^-$ for some j, $1 \le j \le 3$.

decomposable
median
online program
Gault
Jeavons
symmetry
monotone Boolean function
Historical notes
Rosenberg
strongly rigid
potency
Boolean ternary relations
shortcut
idempotent
conservative
projection

[Exercise 398's XCC version of the indicator problem $\mathcal{I}_3$ has 1296 options and $216+27$ items. Algorithm C$^+$ finds all 18 solutions in 79 Mμ, with a search tree of 229 kilonodes. Algorithm B WTD finds them in 3 Mμ, with a search tree of only 6208 nodes.]

[The same argument finds all polymorphisms of '$x \neq y$' on $\{0, 1, \ldots, d-1\}$ for all d, if we extend it slightly: Suppose we have $[x^{d-1}y] = y$, but $[x^{j-1}yx^{d-j}] = x$ for $1 \leq j < d$. Then if $[x_1 \ldots x_d] = y \neq x_d$, we reach a contradiction either with $[y^{j-1}zy^{d-j}] = y$ for some $j < d$ and $z \neq y$, or with $[z^{d-1}y] = y$ for some $z \notin \{x_1, \ldots, x_d\}$.]

XCC
projection
Jeavons
indicator problem

402. Eight constraints '$[\alpha] \in \{0,1\}$' constrain nothing. Nineteen constraints '$[\alpha][\beta] \in \{00, 01, 11\}$', each repeated three times, occur when $\alpha \subset \beta$ (that is, when $\alpha \mathbin{\&} \beta = \alpha \neq \beta$). (Seven of them, with $\nu\beta > \nu\alpha + 1$, are redundant by transitivity.) The other ten constraints '$[\alpha][\beta][\gamma] \in R$', each repeated six times, are trickier. Six of them have $\alpha + \beta = \gamma$. Three of them have $\nu\alpha = \nu\beta = 2$ and $\gamma = 111$. The other one has $\nu\alpha = \nu\beta = \nu\gamma = 2$. Option 110 of R is redundant in all but the last of them.

403. Inspired by answer 402, with $x = [011]$, $y = [100]$, $z = [101]$, and $w = [111]$, we find $Q(x,y,z) = R(x,y,w) \wedge R(x,z,w) \wedge R(y,z,z)$ for some w.

[Conversely, although Q is an awkward relation to start with, one can show from $\mathcal{I}_3(\{Q\})$ that $R(x,y,z) = Q(x,y,w) \wedge Q(y,z,w) \wedge Q(z,x,w)$ for some w.]

404. Now there are only $4^3 = 64$ constraints. But they're actually the same as in answer 402, except (i) not repeated as often; (ii) no constraint for $\{[011], [101], [110]\}$; (iii) constraints on three variables have options from $\widehat{R}$, not R. The solution, with $x = [011]$, $y = [101]$, $z = [110]$, $w = [111]$, is $R(x,y,z) = \exists w(\widehat{R}(x,y,w) \wedge \widehat{R}(y,z,w) \wedge \widehat{R}(z,x,w))$.

405. The $625 = 16 \cdot 1 + 65 \cdot 3 + 69 \cdot 6$ constraints of $\mathcal{I}_4(\{R\})$ have 16 non-constraints '$[\alpha] \in \{0,1\}$'; 65 $\leq$-constraints '$[\alpha][\beta] \in \{00, 01, 11\}$'; and 69 trickier constraints '$[\alpha][\beta][\gamma] \in R$'. The latter divide into (6, 12, 4, 12, 4, 3, 6, 12, 4, 6) cases where the row sums and column sums of the 4×3 matrix A (whose columns are (α, β, γ)) are respectively $((0022, 112), (0222, 123), (0222, 222), (0223, 223), (2222, 134), (2222, 224), (2222, 233), (2223, 234), (2223, 333), (2233, 334))$.

Now we play a little interactive game, repeatedly trying to eliminate constraints whose removal doesn't make the projection of [0101][1010][1100] grow from T to the full set of eight triples. And we discover that thirteen variables can be removed. The survivors, $v = [0100]$, $w = [1000]$, $x = [0101]$, $y = [1010]$, $z = [1100]$, yield (156).

406. $T(x,y,z) = R(x,y,w) \wedge R(w,w,z)$ for some w. [This formula *cannot* be obtained by simplifying an indicator problem for R as we did in exercise 405, because a constraint '$[\alpha][\beta][\gamma] \in R$' is symmetrical between the positions of 0 in α and β.]

407. $S^{(3)}(x,y,z) = S^{(m)}(v,w,w,\ldots,w) \wedge S^{(m)}(x,y,z,w,\ldots,w)$ for some v and w.

408. (a) $N(x,y,z) = S(w,x,y,z)$ for some w.
(b) $S(w,x,y,z) = N(w,x,y) \wedge N(w,x,z) \wedge N(w,y,z) \wedge N(x,y,z)$.

409. Notice that $[a_{11} \ldots a_{k1}] \ldots [a_{1m} \ldots a_{km}] = a_{j1} \ldots a_{jm}$ when the solution to $\mathcal{I}_k(\Gamma)$ is the projection π_j. Hence all tuples of R occur among the solutions; we must show that there are no others, when a gadget exists. If indeed there's a k-ary polymorphism of Γ for which $[a_{11} \ldots a_{k1}] \ldots [a_{1m} \ldots a_{km}] \notin R$, then f would be a polymorphism of all gadgets derivable from Γ. But f isn't a polymorphism of R, by the definition of polymorphism. [P. Jeavons, *LNCS* **1520** (1998), 2–16, Theorem 6.]

410. The stated indicator problem $\mathcal{I}_4$ has 512 solutions. However, it involves only 253 of the $5^4 = 625$ possible variables; the true number of polymorphisms is $512 \cdot 5^{625-253}$. We'll get a gadget only if both variables $[a_{11}a_{21}a_{31}a_{41}]$ and $[a_{12}a_{22}a_{32}a_{42}]$ for a given relation belong to the set of 253. (For example, the variable [0012] cannot be used.)

It turns out that 1225 of the 12650 cases pass this test. And, surprisingly, 1171 of those 1225 have no pairs that fail the corresponding binary relation, among all 512 solutions. The first success is the relation $\{00, 01, 03, 10\}$; the last is $\{34, 42, 43, 44\}$; the first failure in qualifying columns is $\{00, 01, 03, 41\}$.

411. For R_0, $[x_1 \ldots x_k]$ is unconstrained when $\{1, 2\} \subseteq \{x_1, \ldots, x_k\}$, because such k-tuples never arise. Otherwise $[x_1 \ldots x_k]$ is a projection. The number of unconstrained tuples is $u_k = \sum_l \binom{k}{l}(2^l - 2 + \delta_{l0}) = 3^k - 2^{k+1} + 1$. So R_0 has $k \cdot 3^{u_k}$ polymorphisms.

R_1 and R_2 have the same polymorphisms, because $R_1(x, y, z) = R_2(x, y, z) \wedge R_2(x, z, y)$ and $R_2(x, y, z) = \exists v\, \exists w \big(R_1(x, v, z) \wedge R_1(y, w, z) \wedge R_1(v, w, z)\big)$.

Every polymorphism in this case is derived from a projection π_j in a curious way: $[x_1 \ldots x_k] = 0$ if and only if $x_j = 0$; and $[x_1 \ldots x_k] = 2$ only if $2 \in \{x_1, \ldots, x_k\}$. Given j, there are $v_k = 2 \cdot 3^{k-1} - 2^{k-1}$ places where 2 is allowed. Hence $k \cdot 2^{v_k}$ is the total.

For R_3, 2 is allowed in $w_k = 2 \cdot 3^{k-1}$ places, for each j, and the total is $k \cdot 2^{w_k}$.

412. (a) Suppose $[0] = x$ and $(x{+}1) \bmod d = y$. Then $[1] = y$ and $[2] = y$, if $x \neq 0$; and $[2] = y$ implies $[1] = x$ if $y \neq 2$. Also $[2] = 2$ implies $[1] \in \{0, 1\}$. Consequently $[0] = 0$. Hence $[1] \in \{1, 2\}$. But $[1] = 2$ would imply $[0] = 1$. Hence $[1] = 1$; and $[2] = 2$, etc.

(b) $x\, \vec{C_d}\, y = x \;\Theta_d\; y \;\Theta_d\; z_1 \;\Theta_d \cdots \Theta_d\; z_{d-2} \;\Theta_d\; x$ for some $z_1, \ldots, z_{d-2}$.

(c) The m-fold composition, namely '$x\, R\, y = x \;\Theta_d\; z_1 \;\Theta_d\; \cdots \;\Theta_d\; z_m \;\Theta_d\; y$ for some $z_1, \ldots, z_m$', is (amazingly) true for all $(x, y) \neq (1, 1)$ if we set $m = (d-1)^2$.

(d) (Solution by D. Zhuk.) Let $R \circ R'$ denote the composition of binary relations R and R'; that is, '$x\, R \circ R'\, y = x\, R\, z\, R'\, y$ for some z'. Also let $R^{(0)} = I$ be the identity relation and $R^{(m+1)} = R^{(m)} \circ R$, so that $R^{(m+1)}$ is the m-fold composition of R with itself.

Now if R is any relation that is true for all pairs (x, y) except (a, b), then the composition $\vec{C_d}^{(s)} \circ R \circ \vec{C_d}^{(t)}$ is true for all pairs (x, y) except $\big((a - s) \bmod d, (b + t) \bmod d\big)$.

From (b) and (c) we can therefore make a gadget R_x from Φ_d that omits only the pair (x, x), for $0 \leq x < d$. Hence $N = R_0 \wedge \cdots \wedge R_{d-1}$ is a gadget for the not-equal relation '$x \neq y$'—whose k-ary polymorphisms are known (see answer 401).

Every polymorphism of Φ_d is a polymorphism of every gadget made from it. From (a), every k-ary polymorphism must satisfy $[x \ldots x] = x$, for $0 \leq x < d$, because $[x \ldots x]$ is a unary polymorphism. So the argument of answer 401 proves that Φ_d is potent.

413. (After a solution by C. Tardif.) No. Let the relation be specified by a digraph G on d vertices, with $m \leq d$ arcs. If G has both a source vertex a and a sink vertex b, G has the binary polymorphism $[xy] = (xy = ab?\ \bar{a}\colon x)$, for any $\bar{a} \neq a$. There will be both a source and a sink if $m < d$. Thus we can assume that $m = d$, and (by symmetry) that G has no sink. But then every vertex v has a unique successor v'; and there's a unary polymorphism $[x] = x'$.

414. The set A itself is always a closed set that contains X. Furthermore, if X_1 is closed and $A \supseteq X_1 \supseteq X$, then $X_1 = \overline{X_1} \supseteq \overline{X}$.

415. Append '3' to the eight subsets X to get eight new subsets.

(a) The new X all get polarity ϵ. Both $\epsilon^{\swarrow}$ and the closed set 012 become 0123.

(b) The eight new X get the polarities they had without '3'. Append '3' to all of the 32 polarities $Y^{\swarrow}$, and to all seven of the closed sets $\overline{X}$.

The closed sets $\overline{Y}$ remain unchanged in both (a) and (b).

416. If $R' \subseteq R$, $A' = A$, and $B' = B$, the new closed sets $\{x\}^{\nearrow}$ are contained in the old ones. Hence every new $X^{\nearrow}$, which is $\bigcap_{x \in X} \{x\}^{\nearrow}$, is contained in its old counterpart.

If $A' \subset A$, $B' = B$, and $R' = R$ restricted to A', the new closed subsets of A' are the intersections of A' with the old ones. (Notice that old closed subsets of A become

projection
m-fold composition
composition of binary relations
Zhuk
identity relation
nonequality
Tardif
source
sink

identical when they have the same intersection with A'.) The old closed subsets $\overline{Y}$ of B become equivalent (merged together) when their polarity mates become identical in A'.

417. (a) $X^{\nearrow} = \{y \mid y \text{ is a multiple of } \operatorname{lcm}(X)\}$; $Y^{\swarrow} = \{x \mid x \text{ divides } \gcd(Y)\}$. Hence $\{\overline{X}\} = \{A\} \cup \bigcup_{n\ge1}\{\{d \mid d \text{ divides } n\}\}$; $\{\overline{Y}\} = \{\emptyset\} \cup \bigcup_{n\ge1}\{\{dn \mid d \ge 1\}\}$.

(b) $X^{\nearrow} = \{y \mid y^3 \text{ is a multiple of } \operatorname{lcm}(X)^2\}$; $Y^{\swarrow} = \{x \mid x^2 \text{ divides } \gcd(Y)^3\}$. If $m = 2^{e_2}3^{e_3}5^{e_5}\ldots$ and $n = 2^{f_2}3^{f_3}5^{f_5}\ldots$, then m^3 is a multiple of $n^2 \iff 3e_p \ge 2f_p$ for all $p \iff m$ is a multiple of $2^{\lceil\frac{2}{3}f_2\rceil}3^{\lceil\frac{2}{3}f_3\rceil}5^{\lceil\frac{2}{3}f_5\rceil}\ldots$; m^2 divides $n^3 \iff 2e_p \le 3f_p$ for all $p \iff m$ is a divisor of $2^{\lfloor\frac{3}{2}f_2\rfloor}3^{\lfloor\frac{3}{2}f_3\rfloor}5^{\lfloor\frac{3}{2}f_5\rfloor}\ldots$. Hence $\{\overline{X}\} = \{A\} \cup \bigcup_{n\in Q}\{\{d \mid d \text{ divides } n\}\}$, where $Q = \{1,2,3,5,6,7,8,10,\ldots\}$ is the set of integers whose prime factorization has $f_p \bmod 3 \ne 2$ for all p; $\{\overline{Y}\} = \{\emptyset\} \cup \bigcup_{n\ge1}\{\{dn \mid d \ge 1\}\}$ as in (a).

(c) This relation R is symmetric. We can restrict consideration to the subset $A_0 = B_0$ of nonzero matrices having the form $\left(\begin{smallmatrix} x & y\\ z & 0\end{smallmatrix}\right)$, by subtracting multiples of $\left(\begin{smallmatrix} 1 & 0\\ 0 & 1\end{smallmatrix}\right)$ (which commute with everything). If M and M' are 2×2 matrices, let's write $M \equiv M'$ if M or M' is zero or if M' is a rational multiple of M; also write $E(M) = \{M' \mid M \equiv M'\}$. Notice that $\left(\begin{smallmatrix} x & y\\ z & 0\end{smallmatrix}\right)$ commutes with $\left(\begin{smallmatrix} a & b\\ c & 0\end{smallmatrix}\right)$ if and only if $ay = bx$, $bz = cy$, and $cx = az$. Consequently two matrices M, M' of A_0 commute if and only if $M \equiv M'$.

Let X be a nonempty subset of A_0. If X contains two inequivalent matrices, then $X^{\nearrow} = \emptyset$ (with respect to B_0); otherwise $X^{\nearrow} = E(X_0)$, for every matrix $X_0 \in X$.

The closed sets in A and B, other than A itself, are therefore all sets of the form $\bigl\{M + \left(\begin{smallmatrix} t & 0\\ 0 & t\end{smallmatrix}\right) \bigm| t \text{ integer and } M \in E(\left(\begin{smallmatrix} a & b\\ c & 0\end{smallmatrix}\right))\bigr\}$, where $(a,b,c) = (0,0,0)$ or $\gcd(a,b,c) = 1$.

(d) $X^{\nearrow} = \{y \mid \sup X \le y\}$; $Y^{\swarrow} = \{x \mid x \le \inf Y\}$. Hence $\{\overline{X}\} = \{\emptyset, A\} \cup \bigcup_{\alpha\text{ real}}\{(-\infty\,.\,.\,\alpha] \cap A\}$; $\{\overline{Y}\} = \{\emptyset, A\} \cup \bigcup_{\alpha\text{ real}}\{[\alpha\,.\,.\,\infty) \cap A\}$. (This is essentially how R. Dedekind famously constructed the real numbers from the rational numbers by means of "cuts." See *Stetigkeit und irrationale Zahlen* (Braunschweig: Vieweg, 1872).)

(e) $X^{\nearrow} = \{y \mid \sup X < y\}$; $Y^{\swarrow} = \{x \mid x < \inf Y\}$. Hence $\{\overline{X}\} = \{\emptyset, A\} \cup \bigcup_{\alpha\text{ real}}\{(-\infty\,.\,.\,\alpha) \cap A\}$; $\{\overline{Y}\} = \{\emptyset, A\} \cup \bigcup_{\alpha\text{ real}}\{(\alpha\,.\,.\,\infty) \cap A\}$. (Think about it.)

(f) If X is infinite or contains a transcendental number, $X^{\nearrow} = \{0\}$. Otherwise $X^{\nearrow}$ is the set of all polynomial multiples of the least common multiple of $\mu(x)$ for all $x \in X$, where $\mu(x)$ is the minimal polynomial of x times the least positive integer that makes all coefficients integers. (Every algebraic number has a unique minimal polynomial, which is the monic polynomial of smallest degree, with rational coefficients, for which x is a root. The other roots of this polynomial, if any, are called "conjugates" of x.)

On the other hand, $Y^{\swarrow} = \{x \mid x \text{ is a root of } \gcd(Y)\}$.

Hence the closed subsets of A are A itself and the finite sets of algebraic numbers that are closed under taking conjugates. The closed subsets of B are the "principal ideals" of squarefree polynomials, namely the sets $\{p(x)f(x) \mid p(x) \in B\}$, for polynomials f such that $\gcd(f(x), f'(x)) = 1$.

418. If $X \subseteq A$, $X^{\nearrow}$ is the set of automorphisms that fix X. This set also fixes the smallest subfield $\overline{X}$ of A that contains X. (If α is an automorphism that fixes x and y, then $\alpha(x+y) = \alpha(x) + \alpha(y) = x + y$ and $\alpha(xy) = \alpha(x)\alpha(y) = xy$.) Thus every closed set of B is a set of automorphisms that fixes a particular subfield. If $Y \subseteq B$, $Y^{\swarrow}$ is the set of all elements of the field A that are fixed by all automorphisms of Y. (Notice that every automorphism permutes the roots. Indeed, $\alpha(z_i)$ must equal z_j for some j, because α fixes all integers: $\alpha(p(z_i)) = p(\alpha(z_i))$.) Those field elements are also fixed by all automorphisms in the smallest subgroup $\overline{Y}$ that contains Y's automorphisms.

419. (a) This is an immediate consequence of (162) and 1.2.6–(33).

(b) If $d = 1$, all operations are constant; so $e_n = [n=0]$ and $f_n = 1$. If $d > 1$, all operations in the smallest clone are projections; so $e_n = [n=1]$ and $f_n = n$.

least common multiple
greatest common divisor
prime factorization
symmetric relation
commute
Dedekind
real numbers
cuts
transcendental number
least common multiple
minimal polynomial
algebraic number
conjugates
principal ideals
squarefree polynomials
projections

Willard
projection functions
nullary

(c) In general there are $f_n = d^{d^n}$ n-ary operations on $\{0,1,\ldots,d-1\}$. Set $d=2$ and apply (a) to get $e_4 = \binom{4}{4}2^{16} - \binom{4}{3}2^8 + \binom{4}{2}2^4 - \binom{4}{1}2^2 + \binom{4}{0}2^1 = 64594$.

420. (a) When $d \ge k \ge 3$, the function $f(x_1,\ldots,x_k) = \big[\{x_1,\ldots,x_k\}$ has k distinct elements$\big]$ is full; but f_{ij} is constant (identically 0).

(b) True. For example, we can't have both $f(a_2,a_2,a_3,\ldots) = f(a_2,a_2',a_3,\ldots)$ and $f(a_2,a_2',a_3,\ldots) = f(a_2',a_2',a_3,\ldots)$ when $f(a_2,a_2,a_3,\ldots) \ne f(a_2',a_2',a_3,\ldots)$.

(c) We must have $0 < r \le k-3$. By renumbering variables we can assume that $i=1$, $j=k$, and $f(x_k,x_2,\ldots,x_k) = h(x_{k+1-r},\ldots,x_k)$, where h is full. Let $g = f_{23}$, and notice that $g_{1k}(x_1,\ldots,x_k) = f(x_k,x_3,x_3,\ldots,x_k) = h(x_{k+1-r},\ldots,x_k)$ depends on x_k. Hence by (b), g depends on x_1 or x_k. It also depends on x_{k+1-r}, …, x_{k-1}; so it can't depend on x_3, by maximality of r.

(d) There are values b_1, …, b_r, b_1' such that $f(b_1,b_2,\ldots,b_r,x_{r+1},\ldots,x_k) \ne f(b_1',b_2,\ldots,b_r,x_{r+1},\ldots,x_k)$ whenever $x_{k-1} = x_k$. Hence f_{ij} depends on x_1 when $i,j > r$. (Proof: That inequality holds when $x_i = x_j = x_{k-1} = x_k$.) Similarly, f_{ij} depends on x_2, …, x_r. Hence f_{ij} is independent of x_{r+1}, …, x_k, by maximality of r. It follows that $a_i = a_j$ implies $f(a_1,\ldots,a_k) = f(a_1,\ldots,a_r,a_k,\ldots,a_k) = h(a_1,\ldots,a_r)$.

Now suppose $a_1 = a_j$ for some $j > r$, and let $g = f_{1j}$. Since g depends on x_2, …, x_r, it depends on at most one other variable; it must be independent of some variable x_l with $l > r$, $l \ne j$. And $a_j = a_1$ implies $f(a_1,\ldots,a_k) = f(a_1,\ldots,a_{l-1},a_1,a_{l+1},\ldots,a_k)$, which is $h(a_1,\ldots,a_r)$. The same argument works when $a_i = a_j$ for $i \le r$ and $j > r$.

Finally, if $a_i = a_j$ and $i,j \le r$, f_{ij} might be independent of x_l for some $l > r$. If so, we can replace a_l by a_i and apply previous results. If not, f_{ij} depends on x_l for all $l > r$. But it can depend on at most r variables; so it must be independent of some x_l where $l \le r$, $i \ne l \ne j$. Then h_{ij} is also independent of x_l. We can evaluate $f(a_1,\ldots,a_k)$ by changing a_l to a_k and applying previous results.

Hey, we've proved that f depends on only r variables. Contradiction! QED lemma. *Reference:* R. Willard, *Discrete Mathematics* **149** (1996), 239–243, Lemma 1.2.

421. The hinted procedure can be implemented thus: If $m=0$, visit only $i_1\ldots i_l = 0\ldots 0$. Otherwise begin with $i_0 i_1 \ldots i_{l-1} i_l \leftarrow 00\ldots 0m$ and $r \leftarrow l$; r will be the position of the leftmost 'm'. After visiting $i_1\ldots i_l$, set $p \leftarrow l$; and while $i_p = m$, set $i_p \leftarrow 0$, $p \leftarrow p-1$. Terminate if $p=0$. Otherwise set $i_p \leftarrow i_p + 1$. If $p < r$ and $i_p = m$, set $r \leftarrow p$; if $p < r$ and $i_p < m$, set $r \leftarrow l$ and $i_l \leftarrow m$. Then visit $i_1\ldots i_l$.

To generate the k-ary clone, represent the ith k-ary operation as a sequence F_i of d^k function values in D. (Think of F_i as a string of $d^k\lceil \lg d\rceil$ bits — for example, a 54-bit number if $d=k=3$.) Begin with the k projection functions x_j, as well as all of the constant (nullary) operations of O, if any. This gives us $n \ge k$ initial functions F_0, F_1, …, F_{n-1}; we shall maintain a dictionary of the functions that we currently know are in the clone. Set $m \leftarrow 0$.

For every operator $g \in O$, if g is l-ary for $l > 0$, and for every l-tuple $i_1\ldots i_l$ with $\max\{i_1,\ldots,i_l\} = m$, we compute the k-ary function $F = g(F_{i_1},\ldots,F_{i_l})$. If F is not in the current dictionary, enter it by setting $F_n \leftarrow F$ and $n \leftarrow n+1$.

After doing that for all g, terminate if $m = n$. Otherwise set $m \leftarrow m+1$ and repeat the loop over all g. This process will eventually terminate with $n \le d^{d^k}$.

422. All functions $a_1x_1 + \cdots + a_kx_k$ whose coefficients a_j are nonnegative integers and (i) exactly one a_j is $\equiv 1$, (ii) no a_j is $\equiv 2$ (modulo 3). [This necessary condition is also sufficient, because we can obtain any such function f with $\max\{a_1,\ldots,a_k\} = a_j > 1$ by computing $g + 3x_j$, where $g = f - 3x_j$ also satisfies (i) and (ii).]

423. All functions $a_1x_1 + \cdots + a_kx_k$ whose coefficients are nonnegative integers and (i) at most one coefficient is 1; (ii) the coefficients aren't all 0; and (iii) the nonzero coefficients aren't all 2.

424. $m_5(x_1, x_2, x_3, x_4, x_5) = m_4(x_1, x_2, x_3, x_4) \vee \cdots \vee m_4(x_2, x_3, x_4, x_5)$; and $x \vee y = m_4(x, x, y, y)$. But every $f(x_1, x_2, x_3, x_4)$ in the clone of m_5 contains an element x_j such that $f(x_1, x_2, x_3, x_4) \geq x_j$. [Post published his complete classification of Boolean clones in *Annals of Mathematics Studies* **5** (Princeton, New Jersey: 1941), 122 pages. He called them "iteratively closed systems." Philip Hall began to call them "clones" in the 1950s and passed the idea to his students, notably P. M. Cohn. An excellent exposition of Post's classification appears in Chapter 1 of N. Pippenger's book *Theories of Computability* (Cambridge University Press, 1997).]

425. (a) Every element of $\mathcal{C}$ corresponds to a tree, as in (175). By induction on the size of that tree, every k-ary operation has the form $(a_1x_1 + \cdots + a_kx_k) \bmod d$, where the coefficients a_j are integers with $(a_1 + \cdots + a_k) \bmod d = 1$.

Conversely, every operation of that form is in $\mathcal{C}$: If $k = 1$, $f(x_1) = x_1$. Otherwise

$$f_{a_1 \ldots a_{k-1}}(x_1, \ldots, x_k) = t\bigl(f_{a_1}(x_1, x_k), x_k, f_{a_2 \ldots a_{k-1}}(x_2, \ldots, x_k)\bigr),$$

where $f_0(x, y) = y$, $f_1(x, y) = x$, and $f_{a+1}(x, y) = t(f_a(x, y), y, x)$.

(b) $f_0 = 0$ and $f_k = d^{k-1}$, because a_k is determined by $a_1, \ldots, a_{k-1}$.

(c) A k-ary WNU has $a_1 \equiv \cdots \equiv a_k$ (modulo d); so it can be in $\mathcal{C}$ only if $k \perp d$. Thus if d is the product of all primes < 1000, the smallest suitable k is 1009.

[Thus there is no "uniform" way to construct a WNU polymorphism from a given polymorphism that is known to satisfy $\mu_d(x, x, y) = \mu_d(y, x, x) = y$. See M. Maróti and R. McKenzie, *Algebra Universalis* **59** (2008), 466.]

426. (a) Equality between variables defines a set partition that determines the outcome; hence it suffices to evaluate f when (w, x, y, z) is a restricted growth string (RGS). For example, the relevant RGS when $w = y \neq x = z$ is 0101. We find $f(0, 1, 0, 1) = t(1, 1, 0) = 0$; therefore it's also true that $f(0, 2, 0, 2) = 0$, $f(0, 3, 0, 3) = 0$, $f(1, 0, 1, 0) = 1$, etc. There are $\varpi_4 = 15$ RGSs of length 4: $0000 \mapsto 0$ ($\{w, x, y, z\}$); $0001 \mapsto 1$ ($\{z\}$); $0010 \mapsto 1$ ($\{y\}$); $0011 \mapsto 1$ ($\{y, z\}$); $0012 \mapsto 2$ ($\{z\}$); $0100 \mapsto 0$ ($\{w, y, z\}$); $0101 \mapsto 0$ ($\{w, y\}$); $0102 \mapsto 2$ ($\{z\}$); $0110 \mapsto 0$ ($\{w, z\}$); $0111 \mapsto 0$ ($\{w\}$); $0112 \mapsto 2$ ($\{z\}$); $0120 \mapsto 0$ ($\{w, z\}$); $0121 \mapsto 0$ ($\{w\}$); $0122 \mapsto 0$ ($\{w\}$); $0123 \mapsto 3$ ($\{z\}$). Altogether $(8, 1, 5, 10)$ hits for (w, x, y, z), respectively.

(b, c) Say that the *height* of an RGS $\sigma = a_1 \ldots a_k$ is $h(\sigma) = 1 + \max\{a_1, \ldots, a_k\}$. Every k-ary combo f of t defines a family of labelings $\{\lambda(\sigma) = f(\sigma)\}$, one for every RGS σ of length k, where $0 \leq \lambda(\sigma) < h(\sigma)$. Since there are exactly $\left\{ {k \atop h} \right\}$ RGSs of length k and height h, the total number of k-ary combos is at most $N(k, d) = 1^{\left\{ {k \atop 1} \right\}} 2^{\left\{ {k \atop 2} \right\}} \ldots d^{\left\{ {k \atop d} \right\}}$. In fact, equality occurs: Every possible set of labelings is indeed a combo of t.

The following proof by P. Weigel uses the *quaternary discriminator* $q(w, x, y, z) = t(t(w, x, y), t(w, x, z), z) = (w = x?\ y\text{: } z)$. For every RGS $\sigma = a_1 \ldots a_k$ and label $\lambda(\sigma)$, let $\mu(\sigma) = \min\{j \mid a_j = \lambda(\sigma)\}$. Then the following combo works when $k = 3$:

$$f(x_1, x_2, x_3) = q\bigl(x_2, x_1, q(x_3, x_1, x_{\mu(000)}, x_{\mu(001)}),$$
$$q(x_3, x_1, x_{\mu(010)}, q(x_3, x_2, x_{\mu(011)}, x_{\mu(012)}))\bigr).$$

And we can clearly construct a similar combo for any given k.

[The discriminator was introduced by A. F. Pixley, *Mathematische Zeitschrift* **114** (1970), 361–372, §3; *Mathematische Annalen* **191** (1971), 167–180. See Barto, Kozik, Maróti, McKenzie, and Niven, *Algebra Universalis* **61** (2009), 366.]

Post
Boolean clones
iteratively closed systems
Hall
clones
Cohn
Pippenger
MIX
Maróti
McKenzie
set partition
restricted growth string
RGS
height
Stirling partition numbers
Weigel
quaternary discriminator
Pixley
Barto
Kozik
Maróti
McKenzie
Niven

427. These functions are the elements of the median algebra; see, for example, exercise 7.1.1–84. The numbers for $k \geq 1$ are 1, 2, 4, 12, 81, 2646, 1422564, ... (OEIS A1206).

428. Exactly 1918040. See D. Zhuk and S. Moiseev, *IEEE International Symposium on Multiple-Valued Logic* **43** (2013), 129–134.

429. Let O_m denote the m-ary elements of O. The hinted R is in $\mathrm{Inv}(O)$, because R is preserved by every $h(x_1, \ldots, x_m) \in O_m$: Given $f_1, \ldots, f_m$ in O_k, with corresponding d^k-tuples $f_j(0, \ldots, 0) \ldots f_j(d{-}1, \ldots, d{-}1) \in R$ for $1 \leq j \leq m$, we have $h\bigl(f_1(x_1, \ldots, x_k), \ldots, f_m(x_1, \ldots, x_k)\bigr) \in O_k$ because O is a clone; so h's d^k-tuple belongs to R.

Now let P be the relation in $\mathrm{Inv}(O)$ that contains just the d^k-tuples for the projection operators $\pi_1, \ldots, \pi_k$. Since g preserves $\mathrm{Inv}(O)$, the d^k-tuple $g(0, \ldots, 0) \ldots g(d{-}1, \ldots, d{-}1)$ belongs to R. Hence $g \in O_k$, by the definition of R.

430. If and only if $k \neq m$. For if (144) does not hold, we have $a_{i1} + \cdots + a_{im} = 1$ for $1 \leq i \leq k$ and $a_{1j} + \cdots + a_{kj} = 1$ for $1 \leq j \leq m$. [Consequently, if $\mathcal{C}_K$ is the smallest clone that contains $\{f_k \mid k \in K\}$, we have $\mathcal{C}_K \neq \mathcal{C}_{K'}$ when $K \neq K'$. See Yu. I. Yanov and A. A. Muchnik, *Doklady Akad. Nauk SSSR* **127** (1959), 44–46.]

431. They're trivially preserved by every operation. Having them in the relational clone doesn't help to build nonempty relations; but it's "mathematically clean."

432. True. (a) $\Pi_{(1^m)}D$; (b) $\Pi_{(1^m)}\emptyset$; (c) $\Sigma_{12}D^2$.

433. $R \circ R' = \Pi_{14}\Sigma_{23}(R \otimes R')$.

434. $R \cap R' = \Pi_{12\ldots m}\Sigma_{1(m+1)}\Sigma_{2(m+2)} \ldots \Sigma_{m(m+m)}(R \otimes R')$.

435. There are 12 solutions $\sigma = (aa'bb'cc')$ with $a < a'$, $b < b'$, $c < c'$: 132546, 134526, 142536, 152346, 153624, 162345, 234516, 253614, 351426, 351624, 361425, 451623.

436. (Solution by R. Willard.) It consists of (i) the empty relation; (ii) the unit unary relations $\{a\}$ for $a \in D$; (iii) all n-ary relations R that have at least two tuples, where every pair of tuples $\{a_1 \ldots a_n, b_1 \ldots b_n\}$ satisfies $a_1 \neq b_1, \ldots, a_n \neq b_n$; (iv) products $R_1 \otimes \ldots \otimes R_t$ of types (ii) and (iii).

437. (a) $d^{\,d^k-d-kd(d-1)}$, when $k \geq 3$, since $d + kd(d-1)$ values are prescribed.
(b) Multiply that formula by $d^{\,d(d-1)}$; there also are $d^{\,d(d-1)/2}$ when $k = 2$.
(c) Multiply answer (b) by $d^{\,d}$.

Historical notes: The significance of the NU condition was first pointed out by A. Huhn, *Acta Facultatis Rerum Naturalium Universitatis Comenianæ Mathematica*, Mimoriadne Čislo (1971), 51–56. It was amplified in an influential paper by K. A. Baker and A. F. Pixley [*Math. Zeitschrift* **143** (1975), 165–174], who named it "near unanimity." The WNU generalization was conceived and named by E. Kiss in unpublished work with M. Valeriote (2004), eventually published in a key paper by M. Maróti and R. McKenzie, *Algebra Universalis* **59** (2008), 463–489. And D. Zhuk, in *JACM* **67** (2020), 30:3, defined WNU operations that needn't be idempotent. A. Kazda, in *J. Multiple-Valued Logic and Soft Comput.* **36** (2021), 337–352, called them *quasi*-WNU.

438. We can use a completion algorithm like the procedure of answer 421. The jewel-like pattern in Fig. A–17 illustrates how the 21 strong components interact. Each component is labeled with a typical representative of its mutually derivable NU operations. Components `000000`, `111111`, `222222` have just a single operation, which leads to no others. Component `001122` includes also the operations `120201` and `212010`; these three operations return respectively x_1, x_2, x_3 when the arguments are distinct. The other three "sink" components, namely `100110`, `122112`, and `200220`, each have 8 operations.

median algebra
OEIS
Zhuk
Moiseev
clone
Yanov
Muchnik
Willard
Historical notes
Huhn
Baker
Pixley
Kiss
Valeriote
Maróti
McKenzie
Zhuk
Kazda
quasi-WNU
sink

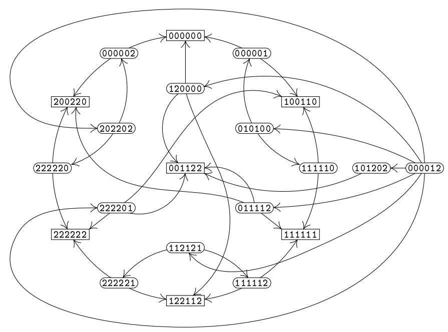


Fig. A–17. A ternary near unanimity operation v on $\{0,1,2\}$ can be derived from another such operation u if and only if there is an oriented path from the strong component that contains u in this digraph to the strong component that contains v.

giant component
source
superachievers
shortcut
lax Siggers terms
idempotent
conservative operators
NP-complete
constant

Component 101202 has 6 operations; components 120000, 011112, 222201 each have 15; nine other components each have 18. The giant component, represented by 000012, is the only "source"; each of its 486 operations leads to all of the others.

439. Notice that different NU operations in the same strong component will support exactly the same relations (see Fig. A–17). Those in the source component, 000012, support the fewest (150). Those in 120000, 011112, 222201 support 151; those in 101202 support 152; those in 001122 support 155. Components 010100, 112121, 202202 support 189; components 000001, 000002, 111110, 111112, 222220, 222221 support 190; components 100110, 122112, 200220 support 192. And those in the singleton sink components 000000, 111111, 222222, are the superachievers who support the most (289).

The superachievers by themselves support 450 of the relations. They curiously fail to support the two relations '$y=(x\pm 1) \bmod 3$', which in fact are supported only by the nine NU operations of components 101202 and 001122. And the remaining $60 = 512 - 450 - 2$ relations aren't supported by any of the NU operations.

Those 60 include, of course, all of the intractable relations, of which there are seven, namely '$\neq$' and six that are equivalent to the "shortcut" relation of (159). Also the six that have five off-diagonal 1s. The other 47 (e.g., '$x \neq (y+1) \bmod 3$') have matrices with at least one 1 on the diagonal; hence they are trivially satisfied when every variable is set to an appropriate constant.

[Exactly 412922 of the $2^{27} = 134217728$ *ternary* relations on $\{0,1,2\}$ are invariant under a ternary NU polymorphism — indeed, 412413 under at least one superachiever.]

440. Introduce new primary items '$!xy$' for all $x, y \in D$ with $x \neq y$. Each of them has d options, namely '$!xy\ yxx\ldots x{:}a\ xyx\ldots x{:}a\ \ \ldots\ \ xx\ldots xy{:}a$' for $0 \le a < d$.

[A similar scheme, with d options of length 4, finds only the lax Siggers terms.]

441. (a) All the idempotent operators: d^{d^k-d} of them, when $|D| = d$.

(b) All the conservative operators (see answer 391), of which there are $\prod_{t=1}^{d} t^{\left\{{k \atop t}\right\} d^{\underline{t}}}$.

[A tractable relation such as $\{01, 02, 12, 20, 33\}$ on $D = \{0,1,2,3\}$ will become NP-complete if we add the unary constraint $D \setminus \{3\}$.]

442. (a) *Every* $f \in O$ has a unique power f^n with $f^n f^n = f^n$ (exercise 3.1–6(b)).

(b) No. There might be two *constant* functions in O. Or we might have, say, $D = \{0,1,2,3\}$ and $O = \{\pi_1, f, g\}$, where $f(x) = x \bmod 2$ and $g(x) = 2 + (x \bmod 2)$.

(c) Let $eD = \{x_1, \ldots, x_m\}$ be the fixed points of e. If $f(x_i) = f(x_j)$ for some $i \neq j$, we would have $|feD| < |eD|$.

(d) $ef \in O$, so it permutes eD; hence there's $m > 0$ with $(ef)^{m-1} efe = e$.

(e) Let Γ be the constraints of $\mathcal{P}$, including unary constraints for each individual domain D_v, and let e be a squashing function with $ee = e$. Define $\widehat{\mathcal{P}}$ to be the CSP whose constraints are the relations eR for each constraint R of $\mathcal{P}$.

[The unary constraints $\{a\}$ for each $a \in eD$ can also be added to $\widehat{\mathcal{P}}$, without changing the complexity, because they merely break symmetry. Then all polymorphisms of $\widehat{\mathcal{P}}$ are idempotent, by answer 441(a). See A. Bulatov, P. Jeavons, and A. Krokhin, *SICOMP* **34** (2005), 720–742, Corollary 4.8. The term "squashing function" was suggested by D. Cohen and P. Jeavons for expository purposes; the term "core" was suggested by T. Feder and M. Vardi by analogy with graph theory.]

443. (Solution by R. Willard.) Suppose $w(x_1,\ldots,x_k)$ is an LWNU in $\mathrm{Pol}(\Gamma)$, and let $e = ee$ be a squashing function. Let $f(x) = w(x,\ldots,x)$, and $\widetilde{w}(x_1,\ldots,x_k) = f^{\sim}(w(e(x_1),\ldots,e(x_k)))$. Then $\widetilde{w}$ is also an LWNU in $\mathrm{Pol}(\Gamma)$; and $\widetilde{w}(x,\ldots,x) = e(x)$ by exercise 442(d). So the restriction $\widehat{w}$ of $\widetilde{w}$ to the subdomain eD is a WNU for $\widehat{\Gamma}$.

Conversely, if $\widehat{w}$ is a WNU of $\widehat{\Gamma}$, let $w(x_1,\ldots,x_k) = \widehat{w}(e(x_1),\ldots,e(x_k))$. [See L. Barto, J. Opršal, and M. Pinsker, *Israel J. Math.* **223** (2018), 363–398, Corollary 8.1.]

444. Notice that $x \bmod 2$ is an endomorphism of Γ. We use the notation of answer 401, and derive some facts about every k-ary LWNU in $\mathrm{Pol}(\Gamma)$ for $k \ge 3$. Relation U implies that $[x_1\ldots x_k] \ne 2$ whenever $2 \notin \{x_1,\ldots,x_k\}$; also $[1{-}x_1,\ldots,1{-}x_k] = 1-[x_1,\ldots,x_k]$ in that case. Suppose $[0^{k-1}1] = 1$ and $[1^{k-1}0] = 0$. Then $[0^{k-2}10] = 0$, by relation V. But that contradicts LWNU; hence $[0^{k-1}1] = 0$ and $[1^{k-1}0] = 1$.

Now $[0^{k-1}2] \ne 0$ implies $[0^{k-1}1] \ne 0$ (by V); $[1^{k-1}2] \ne 1$ implies $[1^{k-1}0] \ne 1$ (by V). And $[2^{k-1}0] = 1$ implies $[2^{k-1}1] = 1$ implies $[0^{k-1}1] = 1$ (by W).

Now let $s = \min\{x_1 + \cdots + x_k \mid 0 \le x_1,\ldots,x_k \le 1 \text{ and } [x_1\ldots x_k] = 1\}$. Then we have $2 \le s \le k-s$. Renumber the coordinates so that $[1^s0^{k-s}] = 1$. Then $[0^s1^{k-s}] = 0$ (by U); hence $[0^s2^{k-s}] = 0$ (by V). Assume that $[2^{k-1}0] = 2$. Then $[2^{s-1}02^{k-s}] = 2$ (by LWNU). Hence $[1^{s-1}02^{k-s}] \ne 0$ (by V). And $[1^{s-1}02^{k-s}] = 1$ implies $[1^{s-1}0^{k+1-s}] = 1$ (by W), contradicting the minimality of s. Therefore $[1^{s-1}02^{k-s}] = 2$; but this implies $[0^s2^{k-s}] = 2$ (by W). Consequently $[2^{k-1}0] = 0$.

Finally, $[2^{k-1}1] = 0$, because $[2^{k-1}1] = 2$ implies $[2^{k-1}0] = 2$ (by W). And $[2^{k-1}1] = 0$ implies $[2^k] = 0$ (by V). It's obvious that $[0^k] = 0$ and $[1^k] = 1$.

(In fact, the LWNU polymorphisms of Γ are precisely the functions $f(x_1 \bmod 2, \ldots, x_k \bmod 2)$, where f is a monotone self-dual Boolean function, not a projection.)

445. The function $\pi_j(x_1,\ldots,x_k) = x_j$ doesn't satisfy (184) when $i = j$. [Taylor terms were introduced by W. Taylor, *Canadian J. Math.* **29** (1977), 498–527, Corollary 5.3. Notice that the operation μ_d in exercise 392 is a Taylor term because $\mu_d(x,y,y) = \mu_d(x,x,x) = \mu_d(y,y,x)$. A *single* identity like '$f(x,y,y,x,x) = f(y,x,x,y,y)$' satisfies (184) for all i; but that's not very interesting, because it just says that the function $g(x,y) = f(x,y,y,x,x)$ is commutative.]

446. The construction for $k = 5$ reveals the general pattern:

$$t(x_1,\ldots,x_8) \;=\; f\bigl(f(x_1,x_4,x_4,x_4,x_4), f(x_2,x_5,x_5,x_5,x_5), \\ f(x_3,x_6,x_6,x_6,x_6), f(x_4,x_7,x_7,x_7,x_7), f(x_5,x_8,x_8,x_8,x_8)\bigr).$$

Evaluate t at (x,x,x,x,y,y,y,y) and (y,y,y,y,y,y,y,y), to cover $i \in \{1,2,3,4\}$; evaluate it at (x,x,x,x,y,x,x,x) and (y,x,x,x,x,x,x,x), to cover $i \in \{1,5\}$; evaluate it at (x,x,x,x,x,x,x,x), (x,x,x,x,x,x,x,y), (x,x,x,x,x,x,y,x), (x,x,x,x,x,y,x,x), to cover $i \in \{6,7,8\}$.

(D. Zhuk has suggested an interesting family of GMM ops when k is odd: If $t \in \{0,1\}$, let $t[x,y] = x + t(y-x)$. Define $f\bigl(t_1[x,y],\ldots,t_k[x,y]\bigr) = g(t_1,\ldots,t_k)[x,y]$, where $g(t_1,\ldots,t_k) = (\{x,y\}$ a majority pair? $\langle t_1\ldots t_k\rangle$: $(t_1+\cdots+t_k) \bmod 2)$.)

Historical notes: If the minority law holds for all x and y, then $g(x,y,z) = f(x,y,\ldots,y,z)$ satisfies the minority law for $k = 3$. A. I. Maltsev introduced that

unary constraints
symmetry
idempotent
Bulatov
Jeavons
Krokhin
Cohen
Feder
Vardi
Willard
squashing function
Barto
Opršal
Pinsker
monotone self-dual
self-dual
historical note
Taylor
commutative
Zhuk
notation: t-of-the-way
Historical notes
Maltsev

3-ary identity in a classic paper of universal algebra, *Matematicheskiĭ Sbornik* (N.S.) **35** (1954), 3–20. V. Dalmau introduced GMM ops in *Logical Methods in Computer Science* **2** (4:1) (2006), 1–15.

447. (a) If we set two arguments equal in a lax Siggers term, we get $s(x,x,x,y) = s(x,x,y,x)$; $s(x,y,x,x) = s(y,x,x,y)$; $s(y,x,y,x) = s(x,y,x,x)$. [The latter two identities, incidentally, appear in (185).] The first two imply that we get a lax Maróti pair by defining $p(x,y,z) = s(z,x,x,y)$, $q(x,y,z) = s(z,x,y,z)$.

(b) When $p = \pi_1$, q is NU. When $p = \pi_2$, q is a special kind of Maltsev polymorphism, also a WNU. When $p = \pi_3$, q is a special kind of Maltsev polymorphism, with $q(x,y,x) = x$. And when $q = \pi_2$, $m(x,y,z) = p(x,z,y)$ is Maltsev.

Historical notes: M. H. Siggers found a pair of 6-ary identities that suffice for tractability, and his surprising discovery was soon improved by other researchers. See *Algebra Universalis* **64** (2010), 15–20; K. Kearnes, P. Marković, and R. McKenzie, *Algebra Universalis* **72** (2014), 91–100; L. Barto, A. Krokhin, and R. Willard, *Dagstuhl Follow-Ups* **7** (2017), 1–44. The "rare area" mnemonic is due to R. O'Donnell.

448. The values of $q(x,y,z)$ are prescribed when $\{x,y,z\}$ aren't distinct. So the probability is $1/d^{d^3-d(d-1)(d-2)}$. [It's the same as $\Pr(q \text{ is NU})$! See exercise 437.]

449. $\mathcal{I}_3(\Gamma)$ has just seven solutions: π_1, π_2, π_3, $e\pi_1$, $e\pi_2$, $e\pi_3$, and the stated operation $m(x,y,z) = (xy+yz+zx) \bmod 2 = \langle x \bmod 2, y \bmod 2, z \bmod 2\rangle$.

So the only lax Maróti pair is $p = e\pi_1$, $q = m(x,y,z)$.

450. The indicator problem $\mathcal{I}_3(\{G\})$ has 126760 solutions, of which 10256 are candidates for q because they satisfy $q(x,x,y) = q(y,x,x)$. Each of the latter have a "score sheet" consisting of 32 values, namely $q(x,x,y)$ and $q(x,y,x)$ for $0 \le x,y < 4$. And each of the original 126760 is a candidate for p; its score sheet consists of the 32 values $p(x,y,y)$ and $p(x,y,x)$ for $0 \le x,y < 4$.

The candidate p's have 152 distinct score sheets, while the candidate q's have only 56. Exactly 8 of those 56 appear also among the 152; and each of those 8 matches a unique q, but more than one p. Furthermore, each of the 8 q's happens to have the form

$$\square(h(x,y,z)) \;=\; \bigl(0 \in \{x,y,z\}?\ 0\colon 3 \in \{x,y,z\}?\ 3\colon h(x,y,z)\bigr),$$

where $h(x,y,z)$ is an operation on $\{1,2\}$. For example, $\square(\langle xyz\rangle)$ is the function in (188). Whenever h is self-dual, in the sense that $\sigma(h(x,y,z)) = h(\sigma(x),\sigma(y),\sigma(z))$ where σ is the symmetry that interchanges $1 \leftrightarrow 2$, $\square(h(x,y,z))$ is a polymorphism of G.

The simplest Maróti pairs occur when $q(x,y,z)$ is $\square(y)$ and $p(x,y,z)$ is either $\square(\langle xy\sigma(z)\rangle)$ or $\square(2-(x{+}y{+}z) \bmod 2)$. The latter case is especially interesting because it's a *double* Maróti pair — we can switch p and q when $p(x,x,y) = p(y,x,x)$.

Six pairs have $q(x,y,z) = \square(\langle x\sigma(y)z\rangle)$. The matching p's are $\square(z)$; $\square(\langle xyz\rangle)$ [a double pair!]; and four of the form $(x{=}y{=}0?\ z \text{ or } 3\colon xyz{=}0?\ 0\colon [x=3]{\ne}[y=3]?\ 3\colon z)$.

When $q(x,y,z) = \square(\langle xyz\rangle)$ there are 66 p's with a matching scorecard: $\square(x)$; $\square(\langle x\sigma(y)z\rangle)$; and 64 of the form $(xy = 0?\ 0\colon y = 3 \text{ and } z > 0?\ 3\colon z > 0?\ x\colon 3 \text{ or } x)$.

Similarly, $q(x,y,z) = \square(2-(x{+}y{+}z) \bmod 2)$ has 66 mates: $\square(y)$; $\square(\langle\sigma(x)yz\rangle)$; and 64 of the form $(xy = 0?\ 0\colon z = 0?\ y \text{ or } 3\colon x = 3?\ 3\colon y)$.

Finally, each of the $2+6+66+66$ (idempotent) Maróti pairs we've described yields a lax (non-idempotent) Maróti pair when we apply σ to both p and q.

451. (Solution by R. Willard.) If r is an m-ary operator and s is an n-ary operator, their "star composition" $r \star s$ is the mn-ary operator

$$(r \star s)(x_{11},\ldots,x_{1n},\ldots,x_{m1},\ldots,x_{mn}) = r(s(x_{11},\ldots,x_{1n}),\ldots,s(x_{m1},\ldots,x_{mn})).$$

universal algebra
Dalmau
Maltsev
Historical notes
Siggers
Kearnes
Marković
McKenzie
Barto
Krokhin
Willard
mnemonic
O'Donnell
indicator problem
self-dual
symmetry
Maroti pair, double
idempotent
Willard
star composition
composition
$r \star s$

The clone generated by the unary operation $f(x) = p(x,x,x) = q(x,x,x)$ has a term $e = ee$ such that $f^\sim fe = e$ and $ef^\sim = f^\sim$ as in exercise 442. Now consider the term $t(x_1,\ldots,x_9) = ((f^\sim q) \star (f^\sim p))(e(x_1),\ldots,e(x_9))$. We have $t(x,y,x,x,y,x,x,y,x) = f^\sim f(f^\sim p(e(x),e(y),e(x))) = f^\sim q(e(x),e(y),e(x)) = t(x,x,x,y,y,y,x,x,x)$. Similarly, $t(x,y,y,x,y,y,x,y,y) = t(x,x,x,y,y,y,y,y,y) = t(y,y,y,x,x,x,x,x,x)$.

conservative law
indicator problem
Post
surprise
identity relation
contribute
universal set

452. *Sketch:* First show that $[xx\ldots x] = x$ for $0 \le x < 4$. Then show that $[x_1\ldots x_k] = 0$ only if $0 \in \{x_1,\ldots,x_k\}$. We can't have $[x_1\ldots x_k] = 1$ when $\{x_1,\ldots,x_k\} \subseteq \{2,3\}$, because that would imply $[0\ldots 0] = 2$. Hence $[x_1\ldots x_k] \in \{x_1,\ldots x_k\}$. Hence $0 \notin \{x_1,\ldots,x_k\}$ implies $[x_1\ldots x_k] = x_j$ for some fixed j. Without loss of generality we can assume that $j = k$. Then $[x_1\ldots x_{k-1}0] = 0$, $[x_1\ldots x_{k-1}1] \in \{0,1\}$, $[x_1\ldots x_{k-1}2] = 2$, $[x_1\ldots x_{k-1}3] \in \{0,3\}$. Equation (183) fails when $(x,y) = (0,2)$.

453. The hinted gadget $Ex = \exists uvwyz\,(u\,Z\,v\,Z\,w \cap z\,Z\,y\,Z\,x \cap z\,Z\,w \cap u\,Z\,x)$ can be found with the help of $\mathcal{I}_2(\{Z\})$ as in exercise 405. Now $2S = \exists uvw\,(u\,Z\,v\,Z\,w\,Z\,u \cap u\,Z\,x \cap v\,Z\,y \cap w\,Z\,z \cap Ex \cap Ey \cap Ez)$. [Is that beautiful, or what?]

454. (a) True. If $f(x_1,\ldots,x_k)$ is a k-ary NU, let $g(x_0,x_1,\ldots,x_k) = f(x_1,\ldots,x_k)$. (b) False. Consider the k-ary Boolean relation $x_1\ldots x_k \ne 0\ldots 0$. [Emil Post knew its polymorphisms; see answer 424.]

455. Suppose $\sigma \in \binom{[m]}{r}$. Clearly $\Pi_\sigma\langle R'\rangle_f \subseteq \Pi_\sigma R$. And if $a_1\ldots a_r \in \Pi_\sigma R$, there are by hypothesis tuples $\tau_j \in \langle R'\rangle_f$, for $1 \le j \le r$, such that $\Pi_\sigma \tau_j = a_1\ldots a_{j-1}b_j a_{j+1}\ldots a_r$ for some b_j. Apply (144) to $\tau_1, \ldots, \tau_k$ (*not* τ_r!), to conclude that $a_1\ldots a_r \in \langle\Pi_\sigma R'\rangle_f$.

456. 8445 of the 2^{16} possible relations (about 13%). Exactly (1, 16, 120, 330, 590, 879, 1152, 1315, 1308, 1101, 777, 468, 240, 102, 36, 9, 1) of them have (0, 1, …, 16) tuples.

457. Random trials, from among the 4^{24} possibilities for f (see answer 437), gave a surprise: The answer turned out to be exactly 451, almost half the time! The reason is that 317 relations never look at unforced values of f; and another 133 cases have an all-1s row or column that makes f's value irrelevant. Since the identity relation also works, regardless of f, the answer is always at least 451.

Ten thousand random trials found 4521 f's with that minimum value 451 (and average weight ≈ 3.5); another 2606 f's had the next-smallest observed value 580 (and average weight ≈ 3.8). The largest value was 1398; only 53 f's yielded a value exceeding 1000. The largest observed average weight was ≈ 5.1, compared to ≈ 7.4 for (191).

458. For Π_{12} we need 002, 202, 212, 302 and either 012 or 013. For Π_3 we need 013. So there are just two stubs: Omit 012 or keep all six. [Notice that if we omit 012, we can't generate the three tuples that start with 0 from the two stub elements that start with 0.]

459. Starting with the tuples {000, 001, 012, 022, 020, 102, 101, 100, 122, 200, 303, 301, 300} of $\Pi_{125}R'$, the completion $\langle\Pi_{125}R'\rangle$ adds two more, 002 and 120. Here 002 was implied by 200, 012, and 122, which are the respective projections of 203103010, 011220300, and 122220202. Hence we put 000220000 into R'_0.

460. Start with $\emptyset$ and "contribute": {023200031, 022220002, 020220000, 011220300, 003200002, 003200000, 003100030, 003100010, 002210002, 002200300, 002200030, 002100000, 001210002, 001200000, 000220000, 000210002, 000200330, 000100310}.

461. Iterate now only for $d_j \in D_{i_j}$. And when $j \notin \{i_1,\ldots,i_{k-1}\}$, let t_j be the smallest element of D_j (instead of 0).

462. The probability that a set of t randomly chosen n-tuples is *not* a stub is at most $\binom{n}{k-1}d^{k-1}(1-d^{1-k})^t$. The latter is less than 1 when $t > d^{k-1}\ln\bigl(\binom{n}{k-1}d^{k-1}\bigr)$; hence there's a stub of size $O(kd^{k-1}\log n)$. [Such a stub is called an "$(n,k{-}1)$-universal set with alphabet size d." N. H. Bshouty proved the lower bound $\Omega(d^{k-2}(\log n)/(\log d))$ in *Electronic Colloq. Computational Complexity* (2012), 11:1–11:108, Lemma 50.]

463. Let H contain the l-tuples $\tau_1, \ldots, \tau_m$ (and the associated n-tuples $\tau_1^*, \ldots, \tau_m^*$). Use the procedure of answer 421 to visit every k-tuple $\tau_{p_1} \ldots \tau_{p_k}$ in order of increasing $\max\{p_1, \ldots, p_k\}$. During each visit, if $\tau = f(\tau_{p_1}, \ldots, \tau_{p_k})$ isn't present in H, set $m \leftarrow m+1$, $\tau_m = \tau$, $\tau_m^* \leftarrow f(\tau_{p_1}^*, \ldots, \tau_{p_k}^*)$. And if $\Pi_{\hat{\rho}}\tau \in R$, also contribute τ_m^* to S.

464. First reject any relation that fails to be invariant because of (198) alone. (Only 22100 relations pass this test.) Then try to define the remaining values of $f(x,y,z)$, one by one, until either finding a true polymorphism (in 2756 cases) or failing 3^{12} times in a row. Then examine each of the 3^{12} to see how many of the 2756 it preserves.

Altogether 90684 of the 3^{12} Maltsev polymorphisms are laggards, preserving only 78 relations (the minimum); they include all but 81 of the 3^6 cases for which $f(x,y,x) \notin \{x,y\}$. At the other extreme, three superachievers preserve 1496 of the 2756 relations. They satisfy $f(x,y,x) = y$ and $f(\text{all distinct}) = \text{constant}$.

465. There are many solutions. One of the nicest, perhaps, is the formula $f(x,y,z) = (x \cdot q)/(z\backslash(y \cdot q))$, where q can be any term. (We could, for instance, let $q = x$ or $q = y$ or $q = z$.) R. Freese points out that Maltsev himself, in 1954, gave a construction that leads to the formula $q\backslash(((q \cdot x)/y) \cdot z)$.

466. The set partitions defined by the forkings on levels 1, …, 9 are 03; 0123; 02 | 13; 0 | 1 | 2 | 3; 01 | 23; 03 | 12; 0 | 1 | 2 | 3; 0 | 1 | 2 | 3; 0 | 1 | 2 | 3.

467. If $a_1 \ldots a_i \in \Pi_{1\ldots i}\, R$, there is by hypothesis a tuple $\tau = t_1 \ldots t_m \in R'$ such that $t_j = a_j$ for $1 \le j < i$. Let $t_i = a$. Then (i, a, a_i) is a forking of R, hence also of R'. So R' contains tuples $\tau' = x_1 \ldots x_{i-1} a x_{i+1} \ldots x_m$ and $\tau'' = x_1 \ldots x_{i-1} a_i y_{i+1} \ldots y_m$. By (144) and (198), $\langle R' \rangle_f$ also contains $f(\tau, \tau', \tau'') = a_1 \ldots a_i z_{i+1} \ldots z_m$, where $z_j = f(t_j, x_j, y_j)$.

468. 020000111 participates in the forking $(3,0,2)$, as well as in nine "monoforkings" $(1,0,0)$, $(2,2,2)$, …, $(9,1,1)$. But all ten are still covered.

(Each of the other eleven is essential, for at least one forking. For example, 022123122 is needed for $(9,2,2)$. But *none* of them are essential, by themselves, for generating P via (198)! For instance, $011122100 \oplus 013001133 \oplus 300023300 = 302100333$.)

469. Represent S as a trie, supplemented by arrays that tell which forkings (i,a,a') are present at a given level i. The first subroutine is a minor variation of algorithms in Section 6.3. For the second, assuming that $\tau \ne \tau'$, there's a unique i such that $t_1 = t_1'$, …, $t_{i-1} = t_{i-1}'$, $t_i \ne t_i'$. If (i, t_i, t_i') isn't already a forking in S, insert both τ and τ' (thereby gaining also (i, t_i', t_i)). Otherwise contribute τ and τ' individually by using the first subroutine twice. S remains compact, because every insertion that doesn't increase the number of forkings is followed by one that raises the number by at least 2.

470. The "only if" is trivial. And if R contains the tuples $\tau = cx_2 \ldots x_{i-1} a x_{i+1} \ldots x_m$, $\tau' = y_1 \ldots y_{i-1} a y_{i+1} \ldots y_m$, $\tau'' = y_1 \ldots y_{i-1} a' z_{i+1} \ldots z_m$, then it contains also

$$f(\tau, \tau', \tau'') = cx_2 \ldots x_{i-1} a' f(x_{i+1}, y_{i+1}, z_{i+1}) \ldots f(x_m, y_m, z_m).$$

471. $\Pi_{18} P' = \{00, 01, 02, 03, 30, 33\}$. We have $32 \in \Pi_{18} P = \langle \Pi_{18} P' \rangle$ because, e.g., $p(02, 03, 33) = 32$. Contribute $f(022123122, 013001133, 302100333) = 333022322$ to P_3'.

472. Starting with an empty stub R'_{-c}, do the following for each forking (i,a,a') in R' with $i < m$: Test first if $ac \in \langle \Pi_{im} R' \rangle$. If not, do nothing. Otherwise let $\tau = t_1 \ldots t_m$ be a tuple of R with $t_i = a$ and $t_m = c$. If $a = a'$, contribute τ to R'_{-c}. Otherwise use the procedure of the text and exercise 470 to construct compact stubs R'_{t_1}, …,

(n,k)-universal set
Bshouty
superachievers
Freese
Maltsev
historical note
set partitions
monoforkings
trie
forkings

$R'_{t_1\ldots t_{i-1}}$ for the subtries R_{t_1}, ..., $R_{t_1\ldots t_{i-1}}$ of R. If $a'c \notin \langle \Pi_{im} R'_{t_1\ldots t_{i-1}} \rangle$, do nothing. Otherwise let $\tau' = t_1 \ldots t_{i-1} a' t'_{i+1} \ldots t'_{m-1} c$ be a tuple of R; contribute τ and τ' to R'_{-c}.

Consider, for example, the forking $(6, 1, 2)$ in the stub P' of (201), trying for P'_{-2}. We have $\Pi_{69} P' = \{00, 01, 03, 10, 13, 20, 21, 30, 32, 33\}$, hence $12 \in \langle \Pi_{69} P' \rangle$; we deduce that $\tau = p(020000111, 302100333, 011121000) = 333021222 \in P$. Forming P'_3, P'_{33}, ..., leads to P'_{33302}, which has only two tuples: τ and $\tau' = 333022322$. Since $22 \in \Pi_{69} P'_{33302}$, we know that $(6, 1, 2)$ is a forking of P_{-2}; so we contribute $\{\tau, \tau'\}$ to P'_{-2}.

On other hand, the forking $(5, 0, 1)$ fares differently. A short calculation shows that $02 \notin \langle \Pi_{59} P' \rangle$. Hence $(5, 0, 1)$ is *not* a forking of P_{-2} (and neither is $(5, 0, 0)$).

Similar computations lead eventually to a compact stub. Here's one that has contributions from $(3, 2, 2)$ and $(3, 3, 3)$: $P'_{-2} = \{020010212,\ 022120022,\ 022123122,\ 022123122,\ 321210032,\ 323323302,\ 331112112,\ 333021222,\ 333021222,\ 333022322\}$.

Warning: Don't be misled by the simplicity of this example! Relations that are preserved by the parity operation (200) are highly structured; their properties are explained by the theory of binary vector spaces. But Maltsev operators in general can be highly complex (see exercise 465). Algorithm M works entirely by logic, not arithmetic.

parity
vector spaces
MERCER
contributes
complete relation

It's parity that no one understands.

— JOHNNY MERCER, *The Country's in the Very Best of Hands* (1956)

473. The following algorithm "contributes" to S as in exercise 469.

M1. [Initialize.] Set S empty. Then contribute the tuples 0^n and $0^j a^{n-j}$ for $0 < a < d$ and $0 \le j < n$. (Now S is a compact Maltsev stub for the complete relation D^n.)

M2. [Get new constraint.] If there's no more input, terminate with a solution τ, where τ is any element of S. Otherwise input a new constraint R, where $R \subseteq D^r$ is an r-ary relation that constrains variables j_1, ..., j_r. (We assume that f preserves R.)

M3. [Initialize q.] Set $q \leftarrow 1$. (We'll constrain variable j_q, given that variables j_1, ..., j_{q-1} already are good for the first $q-1$ levels of trie R.)

M4. [Prepare to constrain.] Set $S_0 \leftarrow S$, $S \leftarrow \emptyset$. Do M7 for every forking (i, a, a') of S_0.

M5. [Satisfiable?] (Now S is a compact Maltsev stub for the n-ary relation $T = \langle S_0 \rangle \cap R(j_1, \ldots, j_r)$, the set of all tuples that satisfy the former constraints as well as the new one.) Terminate unsuccessfully if $S = \emptyset$.

M6. [Loop on q.] Return to M2 if $q = r$. Otherwise set $q \leftarrow q + 1$ and go to M4.

M7. [Contribute (i, a, a')?] (This step generalizes the solution to exercise 472.) Let $\hat\sigma$ be the string $ij_1 \ldots j_q$, except leave out any j's that occur earlier in the string. Let the string $\hat\rho$ have the same relation to $\hat\sigma$ as in step N5. (For example, if $ij_1j_2j_3j_4j_5 = 577215$, we'd have $\hat\sigma = 5721$ and $\hat\rho = 22341$.) Run through all tuples of $\langle \Pi_{\hat\sigma} S_0 \rangle$, looking for a tuple τ that begins with a and such that $\Pi_{\hat\rho}\tau$ satisfies the first q levels of the trie R. If none is found, do nothing. Otherwise let $\tau^\star$ be a tuple of $T_0 = \langle S_0 \rangle$ such that $\Pi_{\hat\sigma}\tau^\star = \tau$. (A dictionary H is used for this completion process, as in Algorithm N.) If $a = a'$, contribute $\tau^\star$ to S. Otherwise, as in answer 472, construct a compact Maltsev stub S' for the elements of T_0 that begin with $t_1 \ldots t_{i-1}$. Run through all tuples of $\langle \Pi_{\hat\sigma} S' \rangle$, looking for a tuple τ' that begins with a' and such that $\Pi_{\hat\rho}\tau'$ satisfies the first q levels of the trie R. If none is found, do nothing. Otherwise let $\tau'^\star$ be a tuple of T_0 such that $\Pi_{\hat\sigma}\tau'^\star = \tau'$. Contribute $\tau^\star$ and $\tau'^\star$ to S. ▮

The running time can be shown to be polynomial because, in step M7, $|\Pi_{\hat\sigma} S|$ is at most d times the number of tuples in the first q levels of R.

474. Notice that, by exercise 447(b), either Algorithm M or Algorithm N answers this question when either p or q is a projection. Can a suitable notion of "stub" be formulated in general? (Such stubs will need to be able to include cases like Horn-SAT, where the number of n-ary relations invariant under p and q is doubly exponential.)

475. For each binary n-tuple $a_1 \ldots a_n$, visit $a_1 \ldots a_n$ if it is a solution; $O(\text{input length})$ gives us enough time to check. Then visit $(a_1{+}2) \ldots (a_n{+}2)$; it's always a solution.

[More interesting questions arise with a ternary relation like

$$\{001, 010, 100, {*}{*}2, {*}2{*}, 2{*}{*}, {*}22, 2{*}2, 22{*}, 222\}, \quad * \in \{0,1\};$$

or with, say, the binary relation '$x = 0$ or $y = 0$ or $x \neq y$' on a domain $\{0, 1, \ldots, c\}$. (Color a map partially: Adjacent vertices either have distinct positive colors, or at least one of them has color 0.) What are optimum ways to visit all solutions of such CSPs?]

476. Consider also the corresponding problem where solutions must be visited lexicographically. [These questions are quite different from asking about the total number of solutions. A. A. Bulatov [*LNCS* **5125** (2008), 646–661] has established a dichotomy theorem for that problem: *A CSP language* Γ *is either #P-complete, or there is a known algorithm to count the number of its solutions in polynomial time.* In the latter case, Γ has a Maltsev polymorphism and satisfies further conditions.]

477. True. It will set $R_{i'j} \leftarrow O$ when $k = i$, and $R_{ij'} \leftarrow O$ when $k = j$; then $R_{i'j'} \leftarrow O$.

478. (a) Let $X = \binom{0\,1}{1\,0}$. When $k = 1$, $R_{52} \leftarrow I$. Then when $k = 2$, $R_{13} \leftarrow I$, $R_{53} \leftarrow X$. Then when $k = 3$, we set $R_{14} \leftarrow X$, $R_{24} \leftarrow I$, $R_{54} \leftarrow I$. Then when $k = 4$, $R_{15} \leftarrow I$, $R_{25} \leftarrow X$, $R_{35} \leftarrow I$, $R_{55} \leftarrow O$. Soon all are O, by exercise 477.

(b) These relations say that $x_{j+1} = (x_j + 1) \bmod 3$ for $1 \leq j < 5$; but $x_5 x_1$ can be not only 01, 12, or 20, but also 02. However, it's peculiar because (for example) $R_{21} = \left(\begin{smallmatrix}1&1&1\\1&1&1\\1&1&1\end{smallmatrix}\right)$ puts *no* constraint on $x_1 x_2$! The constraints begin to propagate as in (a): When $k = 1$, $R_{52} \leftarrow \left(\begin{smallmatrix}1&0&1\\1&0&0\\0&1&0\end{smallmatrix}\right)$; then when $k = 2$, $R_{13} \leftarrow \left(\begin{smallmatrix}0&0&1\\1&0&0\\0&1&0\end{smallmatrix}\right)$, $R_{53} \leftarrow \left(\begin{smallmatrix}1&1&0\\0&1&0\\0&0&1\end{smallmatrix}\right)$; etc.; $R_{55} = \left(\begin{smallmatrix}1&0&0\\0&0&0\\0&0&0\end{smallmatrix}\right)$ after $k = 4$, meaning that x_5 must be 0. A lot happens when $k = 5$, basically forcing $x_1 x_2 x_3 x_4 x_5 = 20120$ and allowing only one possibility for $x_i x_j$ when $i \leq j$.

The first round ends with $R_{21} = \left(\begin{smallmatrix}0&1&1\\0&0&0\\0&0&0\end{smallmatrix}\right)$, $R_{31} = \left(\begin{smallmatrix}0&0&0\\0&1&1\\0&0&0\end{smallmatrix}\right)$, etc. But round 2 "purifies" each R_{ij} with $i > j$ so that only one 1 remains, thus achieving a perfectly stable state.

(c) Nothing changes. In fact, there will be no change in any scenario where $R_{kk} = I$ for all k and $R_{ik} R_{kj}$ is all 1s whenever $i \neq k \neq j$.

(d) When $k = 2$, $R_{13} \leftarrow \left(\begin{smallmatrix}1&0&0\\0&0&1\end{smallmatrix}\right)$ and $R_{33} \leftarrow \left(\begin{smallmatrix}1&0&0\\0&0&0\\0&0&1\end{smallmatrix}\right)$. Then $R_{31} \leftarrow \left(\begin{smallmatrix}1&1\\0&0\\1&1\end{smallmatrix}\right)$, $R_{32} \leftarrow \left(\begin{smallmatrix}1&1\\0&0\\1&1\end{smallmatrix}\right)$ when $k = 3$. No further changes occur; hence the final state is curiously asymmetric, with $R_{13} \neq R_{31}^T$, $R_{23} \neq R_{32}^T$, and R_{21} still equal to $\binom{1\,1}{1\,1}$. (Don't ask what that means.)

479. It's true for $r = 1$; so assume that $r > 1$. Let $k = \max\{k_1, \ldots, k_{r-1}\}$, and let (p, q) be minimum and maximum with $k_p = k_q = k$. (Possibly $p = q$.) By (206) there's a value x such that $(s, x) \in R_{ik}$, $(x, x) \in R_{kk}$, $(x, t) \in R_{kj}$. Hence by induction on r we can find suitable $x_0 \ldots x_p$ with $x_0 = s$ and $x_p = x$, suitable $x_p \ldots x_q$ with $x_p = x_q = x$, and suitable $x_q \ldots x_r$ with $x_q = x$ and $x_r = t$.

480. In each CSA below, $I = \{q_0\}$ and Q is implicit.

(a) $\Omega = \{q_1, q_3\}$; $q_r \mapsto (* \leftarrow a^*?\ q_{(r+a) \bmod 5}\colon q_r)$.

(b) Clearly $n \leq 2d$. Let $e_a = 0^a 1 0^{d-1-a}$ be the unit d-vector with 1 in coordinate a. Let $\Omega = \{q(a, p)\}$, over all $a \in [0\,.\,.\,d)$ and ternary d-vectors p with $p_0 + \cdots + p_{d-1} = n$. Use the transitions $q_0 \mapsto (* \leftarrow a^*?\ q(a, e_a)\colon q_0)$; $q(a, p) \mapsto R_{a,p}, (* \leftarrow b^*?\ q(b, p + e_b)\colon q(a, p))$, where $R_{a,p} = \emptyset$ if $p_a \neq 2$; otherwise $R_{a,p} = \{v_1 \setminus a, \ldots, v_n \setminus a\}$ excludes a from all unassigned domains.

projection
stub
Horn-SAT
Color a map
lexicographically
counting, complexity of
Bulatov
dichotomy theorem
#P-complete
Maltsev polymorphism
unit d-vector

(c) Same as (b), but with Ω restricted to the $\binom{d}{n/2}$ vectors p with no 1s.

(d) Let $\Omega = \{q(n,a,b) \mid 0 \le a < d,\ 0 \le b < 2\}$. Use transitions $q_0 \mapsto R, (v_1 \leftarrow 0?\ q(1,0,0){:}\ q_0)$ and $q(j,a,b) \mapsto R(j,a,b), \bigl(v_{j+1} \leftarrow a'^*?\ (a' = a+1?\ q(j+1,a+1,0){:}\ q(j+1,a,1)){:}\ q(j,a,b)\bigr)$, where $R = \{v_j \setminus a \mid 1 \le j \le n,\ j \le a < d\}$; $R(j,a,0) = \emptyset$; and $R(j,a,1) = \{v_{j+1} \setminus a+2,\ v_{j+2} \setminus a+3,\ \ldots,\ v_n \setminus a+(n-j)+1\}$.

(e) Let $S_j = (-1)^{v_1} + \cdots + (-1)^{v_j}$. A necessary and sufficient condition is that $S_j \ge 0$ for $1 \le j < n$, and $S_n = 0$. One solution is therefore to enter state $q(j, S_j)$ after assigning v_1 through v_j, with $\Omega = \{q(n,0)\}$: $q_0 \mapsto v_1 \setminus 1,\ v_n \setminus 0,\ (v_1 \leftarrow 0?\ q(1,1){:}\ q_0)$; $q(j,s) \mapsto (v_{j+1} \leftarrow a^*?\ q(j+1, s+(-1)^a){:}\ q(j,s))$, for $1 \le j < n$ and $0 \le s \le j$.

[These constructions can often be significantly improved by reducing the domains further. For example, if $d = 10$ and $n = 12$ in (c), $R_{a,1111120000}$ could exclude $\{6,7,8,9\}$ from the domains of all five unassigned variables. In part (d) the underlying CSP might find it much better to assign variables in a different order; if then a is in the domain D_k of v_k, we must have $a-1 \in D_1 \cup \cdots \cup D_{k-1}$. In part (e) we could assign values successively to v_1, v_n, v_2, v_{n-1}, and so on. We could even allow the CSP to assign variables in order of smallest domain; a partial assignment in which s variables have been assigned to 0 and t variables to 1 is then feasible if and only if we get a nested vector by assigning the leftmost $n/2 - s$ unassigned variables to 0 and the others to 1.]

481. Let Q be the set of all states $q(j,a)$ or $q'(j,a)$, for $j \ge 1$ and $0 \le a < d$, together with the special state $\perp$ that is reached when "symmetry is broken." Let $I = \{q(1,0)\}$. Use the transitions $q(j,a) \mapsto (v_j \leftarrow a?\ q'(j,a){:}\ q(j,a+1))$; $q'(j,a) \mapsto v_{n+1-j} \setminus 0, \ldots, v_{n+1-j} \setminus a-1,\ (v_{n+1-j} \leftarrow a?\ q(j+1,0){:}\ \perp)$; $\perp \mapsto (* \leftarrow *?\ \perp{:}\ \perp)$. (State $q'(j,a)$ makes no restrictions when $a = 0$.) We can let $\Omega = Q$; but the actual final states are $q(n/2+1, 0)$ or $q'((n+1)/2, a)$ for palindromic solutions, $\perp$ for the others.

482. The following CSA uses Duval's algorithm (see answer 7.2.1.1–106) to produce only the solutions that are powers of a prime string: Let $I = \{q_0\}$, $Q = I \cup \{q(j,k) \mid 1 \le j < k \le n+1\} \cup \{\#\}$, and $\Omega = \{q(j, n+1) \mid j \text{ divides } n\}$. (State $\#$ is "dead.") Use the transitions $q_0 \mapsto (v_1 \leftarrow *?\ q(1,2){:}\ q_0)$ and

$$q(j,k) \mapsto \bigl(v_k \leftarrow a^*?\ (a < v_{k-j}?\ \#{:}\ a = v_{k-j}?\ q(j,k+1){:}\ q(k,k+1))\bigr).$$

[This method is attractive, but additional pruning is often possible. For example, if D_k is the kth domain, we can remove from D_1 any element $> \max D_k$, for any $k > 1$.]

483. (a) Yes, but only in special cases. The middle row, when $i = \bar\imath$ (hence $i = (n-1)/2$) is special; that's the only time we can have $a_i = \bar c_i$. And we clearly have $a_i = \bar a_i$ if and only if $a_i = (n-1)/2$. Also $a_i = \bar d_i$ if and only if $R_i = C_i$; that can happen without attacking queens if and only if $R_i = i = C_i$. Similarly, $a_i = \bar b_i$ occurs if and only if $R_i = \bar\imath$ and $C_{\bar\imath} = i$. Cyclic symmetry dispenses with the other cases, like $b_i = \bar c_i$.

(b) For example, transposition $(i,j) \leftrightarrow (j,i)$ swaps $R_i \leftrightarrow C_i$; thus $(a_i, b_i, c_i, d_i) \leftrightarrow (\bar d_i, \bar c_i, \bar b_i, \bar a_i)$. In general, reflection complements the set $\{a_i, b_i, c_i, d_i\}$.

(c) Each tuple spawns seven others: $(b_{n'}, c_{n'}, d_{n'}, a_{n'}; \ldots; b_{n-1}, c_{n-1}, d_{n-1}, a_{n-1})$; $(c_{n'}, d_{n'}, a_{n'}, b_{n'}; \ldots; c_{n-1}, d_{n-1}, a_{n-1}, b_{n-1})$; $(d_{n'}, a_{n'}, b_{n'}, c_{n'}; \ldots; d_{n-1}, a_{n-1}, b_{n-1}, c_{n-1})$; $(\bar d_{n'}, \bar c_{n'}, \bar b_{n'}, \bar a_{n'}; \ldots; \bar d_{n-1}, \bar c_{n-1}, \bar b_{n-1}, \bar a_{n-1})$; and so on. Thus the eight tuples for the first solution are (2, 7, 7, 2; 4, 4, 2, 5; 6, 0, 0, 1; 3, 3, 6, 6); (7, 7, 2, 2; 4, 2, 5, 4; 0, 0, 1, 6; 3, 6, 6, 3); (7, 2, 2, 7; 2, 5, 4, 4; 0, 1, 6, 0; 6, 6, 3, 3); (2, 2, 7, 7; 5, 4, 4, 2; 1, 6, 0, 0; 6, 3, 3, 6); (5, 0, 0, 5; 2, 5, 3, 3; 6, 7, 7, 1; 1, 1, 4, 4); (0, 0, 5, 5; 5, 3, 3, 2; 7, 7, 1, 6; 1, 4, 4, 1); (0, 5, 5, 0; 3, 3, 2, 5; 7, 1, 6, 7; 4, 4, 1, 1); (5, 5, 0, 0; 3, 2, 5, 3; 1, 6, 7, 7; 4, 1, 1, 4).

partial assignment
$\perp$
symmetry is broken
palindromes
Duval
dead
transposition

The second solution has central symmetry, so it has only four distinct tuples: (1, 7, 1, 7; 7, 1, 7, 1; 5, 4, 5, 4; 3, 2, 3, 2); (7, 1, 7, 1; 1, 7, 1, 7; 4, 5, 4, 5; 2, 3, 2, 3); (0, 6, 0, 6; 6, 0, 6, 0; 3, 2, 3, 2; 5, 4, 5, 4); (6, 0, 6, 0; 0, 6, 0, 6; 2, 3, 2, 3; 4, 5, 4, 5).

The other two solutions each have eight tuples, of which the lexicographically least turn out to be (0, 1, 8, 7; 6, 3, 5, 1; 1, 8, 1, 3; 5, 6, 4, 6; 2, 4, 0, 8) and (5, 5, 5, 5; 1, 7, 1, 7; 4, 1, 4, 2; 7, 10, 10, 10; 10, 6, 7, 6; 2, 2, 2, 1).

(d) Indeed, if $f(x)$ is *any* one-to-one function that maps every solution x of some combinatorial problem into a tuple, the x's for which $f(x)$ is lexicographically least, over all solutions equivalent to x by any definition of equivalence, are canonical.

(e) True: $\min(a_{n'}, \bar a_{n'}) < n'$; and we can't have $a_{n'} = b_{n'} = c_{n'} = d_{n'} = n' - 1$.

(f) In the following, '$ij?$' is shorthand for '$R_i \leftarrow j?$' or '$C_j \leftarrow i?$' in exercise 480; it means that we either place a queen in cell (i,j) or forbid that cell. Similarly, 'not ij' is shorthand for '$R_i \neq j$, $C_j \neq i$'; this restriction is vacuous unless $0 \le i, j < n$. States q_k arise when we potentially have 4-fold symmetry; states r_k arise when we potentially have 2-fold symmetry; and states s_k are intermediary. After symmetry has been broken we reach state $\perp$, which is the wildcard state '$\perp \mapsto (* \leftarrow *?\ \perp\colon \perp)$' as in answer 481.

$$
\begin{array}{ll}
q_1(i,j) \mapsto R(i,j), (ij?\ q_2(i,j)\colon q_1(i,j{+}1)); & r_1(i,j) \mapsto \text{not } \bar\imath\,\overline{j{-}1}, (ij?\ r_2(i,j)\colon r_1(i,j{+}1));\\
q_2(i,j) \mapsto (j\bar\imath?\ q_3(i,j)\colon s_2(i,j)); & r_2(i,j) \mapsto (\bar\imath\bar\jmath?\ r_3(i,0)\colon \perp);\\
q_3(i,j) \mapsto (\bar\imath\bar\jmath?\ q_4(i,j)\colon s_4(i,j)); & r_3(i,j) \mapsto \text{not } \overline{j{-}1}\,i, (j\bar\imath?\ r_4(i,j)\colon r_3(i,j{+}1));\\
q_4(i,j) \mapsto (\bar\jmath i?\ q_1(i{+}1,0)\colon \perp); & r_4(i,j) \mapsto (\bar\jmath i?\ r_1(i{+}1,0)\colon \perp);
\end{array}
$$

$s_2(i,j) \mapsto \text{not } \bar\jmath i, (\bar\imath\bar\jmath?\ s_3(i,j{+}1)\colon \perp)$; $s_3(i,j) \mapsto \text{not } \overline{j{-}1}\,i, (j\bar\imath?\ r_4(i,j)\colon s_3(i,j{+}1))$; $s_4(i,j) \mapsto \text{not } \bar\jmath i, \perp$; $q_1(i,n) = r_1(i,n) = r_3(i,n) = s_3(i,n) = \perp$. Here $R(i,j)$ stands for the restrictions 'not $(j{-}1)\bar\imath$, not $\bar\imath\,\overline{j{-}1}$, not $\overline{j{-}1}\,i$', as well as four more when $i = \lceil n/2 \rceil$: 'not $\bar\imath j$, not ji, not $i\bar\jmath$, not $\bar\jmath\bar\imath$'. These rules suffice when $n = 2n'$ and $I = \{q_1(n',0)\}$.

If $n = 2n'+1$, let $I = \{s_1(0)\}$ and introduce $n'+1$ new states $s_1(j)$, where we have $s_1(0) \mapsto \text{not } n'j$ and not jn' for $n' < j < n$, $(n'0?\ \perp\colon s_1(1))$; $s_1(j) \mapsto \text{not } \overline{j{-}1}\,n'$, $(n'j?\ \perp\colon s_1(j{+}1))$ for $0 < j < n'$; and $s_1(n') \mapsto (n'n'?\ q_1(n'{+}1,0)\colon \perp)$.

The final state is $(q_1(n,0), r_1(n,0), \perp)$ for solutions with $(4,2,1)$-fold symmetry.

484. (14, 14, 14, 14; 16, 16, 16, 16; 31, 31, 31, 31; 29, 29, 29, 29; 26, 27, 27, 27; 24, 24, 6, 24; 3, 1, 1, 6; 27, 3, 3, 3; 10, 30, 10, 10; 22, 6, 25, 21; 5, 26, 30, 11; 11, 11, 11, 8; 23, 22, 23, 23; 12, 12, 12, 12; 7, 5, 22, 22; 13, 13, 13, 13). [Place twelve queens in extreme positions, and reduce domains accordingly. Then start the CSA of answer 483 in state $q_1(20,26)$; only six canonical solutions continue with $a_{20} = 26$ and $b_{20} = 27$. More than 32 queens could, of course, be treated similarly.]

485. With $\widehat Q(n) = 8\widehat Q_{\rm a}(n) + 4\widehat Q_{\rm s}(n) + 2\widehat Q_{\rm d}(n)$ solutions (see answer 7.2.2.1–24), we have

n =	10	11	12	13	14	15	16	17	18	19	20
$\widehat Q_{\rm a}(n)$ =	0	5	18	231	642	4040	25320	166201	1115373	8060958	61981118
$\widehat Q_{\rm s}(n)$ =	1	1	2	6	11	49	79	245	498	1192	3798
$\widehat Q_{\rm d}(n)$ =	0	0	2	2	0	0	12	17	0	0	60
$\widehat Q(n)$ =	4	44	156	1876	5180	32516	202900	1330622	8924976	64492432	495864256

[See §12.2 of V. Kotěšovec's book *Non-attacking chess pieces* (online since 2011) for detailed information about pieces that combine a queen with a leaper.]

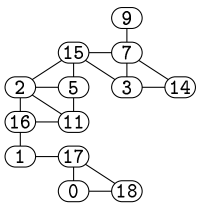
486. Filip Stappers has shown that proof logging is not extremely difficult to add to Algorithms C, F, and S, using Boolean variables for items and options. (Backtracking "learns" a clause; each valid solution $x_{i_1} \wedge \cdots \wedge x_{i_t}$ also learns $(\bar x_{i_1} \vee \cdots \vee \bar x_{i_t})$; finally we certify unsatisfiability.) [`https://archive.org/details/csp-prooflogging`]

central symmetry
one-to-one function
symmetry has been broken
Kotěšovec
leaper
Stappers
certify unsatisfiability

INDEX TO ALGORITHMS AND THEOREMS

I wish you would add an index rerum,
that when a reader recollects any incident,
he may easily find it, which at present he cannot do.

— SAMUEL JOHNSON, letter to Samuel Richardson (9 March 1750)



(see answer 69(b))

4144	4424	4441	4144	3122	4441
4414	4424	4224	4454	3441	2433
1434	1344	2444	2454	3144	2313
2233	3322	2122	2555	1221	3322
(i)	(ii)	(iii)	(iv)	(xi)	(xii)

(see answer 293)

```
144441m53334
432122m54224
432333m54314
434444m54324
455555m54321
mmmmmmmmmmmm
12345m555554
42345m444434
41345m333234
42245m221234
43335m144441
```

(see answer 294)

```
517755444    418759623
557335541    259386741
577731599    367241589
572922997    576812934
832999977    832694175
833423227    941573268
884423357    794128356
188465557    123465897
886666657    685937412
```

(see answer 297)

No intuitive answers, please.

— LIFE INTERNATIONAL (17 December 1962)

INDEX AND GLOSSARY

HUNT

Index making has been held to be the driest
as well as lowest species of writing.
We shall not dispute the humbleness of it;
but . . . the task need not be so very dry.
— LEIGH HUNT, in *The Indicator* (1819)

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.