

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/mmixture.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).

Copyright © 2023 by Addison-Wesley

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

Zeroth printing (revision 4), 04 December 2024

December 5, 2024

## 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?

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

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

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<sup>+</sup>, 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*  
*99 Umbruary 2019*

D. E. K.

**P.S.: A note on notations.** Some formulas in this booklet use the notation ' $\nu x$ ' for the "sideways sum" or "population count" function, as well as the notation ' $\rho x$ ' 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 ' $\nu x$ '  
sideways sum  
population count  
notation ' $\rho x$ '  
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  $v \rightarrow u$ :

- 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 (shrunk) 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

$$u_1, u_2, \dots, u_t, \quad \text{where} \quad \text{PRE}(u_1) < \text{PRE}(u_2) < \dots < \text{PRE}(u_t), \quad (*)$$

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  $u_{j_1}, u_{j_2}, \dots, u_{j_a}$ , with  $j_1 < j_2 < \dots < j_a$ , the depth-first strategy tells us that  $\text{PARENT}(u_{j_1}) = \text{SENT}$ ,  $\text{PARENT}(u_{j_2}) = u_{j_1}$ ,  $\dots$ ,  $\text{PARENT}(u_{j_a}) = u_{j_{a-1}}$ , 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  $S_1, \dots, S_k$  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*

$$u''_1 \rightarrow u'_2, \quad u''_2 \rightarrow u'_3, \quad \dots, \quad u''_{k-1} \rightarrow u'_k,$$

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  $\{C_1, \dots, C_s\}$ , 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  $\text{POST}(v_1) < \dots < \text{POST}(v_s)$ . Then the vertices of  $C_j$  are*

$$\text{desc}(v_j) \setminus (\text{desc}(v_1) \cup \dots \cup \text{desc}(v_{j-1})), \quad \text{for } 1 \leq j \leq s,$$

where  $\text{desc}(v)$  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  $v_1, \dots, v_s$  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,  $\text{LOW}(v)$  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

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

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.

**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  $\text{LOW}(v) = \text{PRE}(v)$ .*

*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  $\text{REP}(v) = \text{REP}(w)$  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  $\text{LOW} = \text{PRE}$ . While vertex  $v$  is active, we shall therefore let

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

Moreover, after  $v$  has been seen, we shall let

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

**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  $\text{SENT} = \text{VERTICES}(g) + \mathbf{N}(g)$  is assumed to be present, with  $\text{REP}(\text{SENT}) = 0$ .

- T1.** [Initialize.] Set  $\text{PARENT}(w) \leftarrow \Lambda$  for  $\text{VERTICES}(g) \leq w < \text{SENT}$ , until eventually  $w = \text{SENT}$ . Also set  $p \leftarrow 0$  and  $\text{SINK} \leftarrow \text{SENT}$ .
- T2.** [Done?] (All vertices reachable from a vertex  $v \geq w$  are now settled.) Terminate if  $w = \text{VERTICES}(g)$ . Otherwise set  $w \leftarrow w - 1$ , and repeat this step if  $\text{PARENT}(w) \neq \Lambda$ . Otherwise set  $v \leftarrow w$ ,  $\text{PARENT}(v) \leftarrow \text{SENT}$ , and  $\text{ROOT} \leftarrow v$ .
- T3.** [Begin to explore from  $v$ .] Set  $a \leftarrow \text{ARCS}(v)$ ,  $p \leftarrow p + 1$ ,  $\text{REP}(v) \leftarrow p$ , and  $\text{LINK}(v) \leftarrow \text{SENT}$ .
- T4.** [Done with  $v$ ?] If  $a = \Lambda$ , go to T7.

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



- T5.** [Visit the next arc,  $v \rightarrow u$ .] Set  $u \leftarrow \text{TIP}(a)$  and  $a \leftarrow \text{NEXT}(a)$ .
- T6.** [If  $u$  is new, move to it.] If  $\text{PARENT}(u) = \Lambda$ , set  $\text{PARENT}(u) \leftarrow v$ ,  $\text{ARC}(v) \leftarrow a$ ,  $v \leftarrow u$ , and go to T3. Otherwise, if  $u = \text{ROOT}$  and  $p = N(g)$  (we're in the last component), while  $v \neq \text{ROOT}$  set  $\text{LINK}(v) \leftarrow \text{SINK}$ ,  $\text{SINK} \leftarrow v$ ,  $v \leftarrow \text{PARENT}(v)$ ; then set  $u \leftarrow \text{SENT}$  and go to T8. Otherwise, if  $\text{REP}(u) < \text{REP}(v)$ , set  $\text{REP}(v) \leftarrow \text{REP}(u)$ ,  $\text{LINK}(v) \leftarrow \Lambda$  (see (\*\*), (\*\*\*)). Go to T4.
- T7.** [Finish with  $v$ .] Set  $u \leftarrow \text{PARENT}(v)$ . If  $\text{LINK}(v) = \text{SENT}$ , go to T8. (See Theorem T.) Otherwise, if  $\text{REP}(v) < \text{REP}(u)$ , set  $\text{REP}(u) \leftarrow \text{REP}(v)$  and  $\text{LINK}(u) \leftarrow \Lambda$ . Then set  $\text{LINK}(v) \leftarrow \text{SINK}$ ,  $\text{SINK} \leftarrow v$ , 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  $\text{REP}(\text{SINK}) \geq \text{REP}(v)$ , set  $\text{REP}(\text{SINK}) \leftarrow \text{SENT} + v$  and  $\text{SINK} \leftarrow \text{LINK}(\text{SINK})$ . (See (\*\*).) Finally set  $\text{REP}(v) \leftarrow \text{SENT} + v$ .
- T9.** [Tree done?] If  $u = \text{SENT}$ , go back to T2. Otherwise set  $v \leftarrow u$ ,  $a \leftarrow \text{ARC}(v)$ , and go to T4. ■

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

**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  $\text{GMATE}[g]$  for  $1 \leq g \leq M$  and  $\text{BMATE}[b]$  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  $\text{QUEUE}[k]$  for  $0 \leq k < M$ , which records the girls currently present. If  $f$  girls are free, they appear in the first  $f$  positions of  $\text{QUEUE}$ . There’s also a partial inverse,  $\text{IQUEUE}[g]$  for  $1 \leq g \leq M$ : If  $0 \leq k < f$  and  $\text{QUEUE}[k] = g$ , then  $\text{IQUEUE}[g] = k$ . Yet another array,  $\text{MARK}[b]$  for  $1 \leq b \leq N$ , equals  $l$  if  $b \in B_l$ ; otherwise  $\text{MARK}[b] = 0$ . There’s also  $\text{MARKED}[t]$ , for  $0 \leq t < N$ ; it lists the boys for which  $\text{MARK}[b] \neq 0$ .

The algorithm also involves a depth-first process, to remove SAPs after the dag has been built. Those steps use the array  $\text{STACK}[l]$ , 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  $\text{GTIP}$ ,  $\text{BTIP}$ ,  $\text{GNEXT}$ ,  $\text{BNEXT}$ . An edge between girl  $g$  and boy  $b$  is represented by an edge node  $e$  for which  $\text{GTIP}(e) = g$  and  $\text{BTIP}(e) = b$ ; here  $1 \leq e \leq E$ , where  $E$  is the total

number of edges. The first edge involving  $g$ , for  $1 \leq g \leq M$ , is  $\text{GLINK}[g]$ ; the next one is  $\text{GNEXT}(\text{GLINK}[g])$ ; and so on, until 0 terminates the list. The values of  $\text{GTIP}$ ,  $\text{BTIP}$ , and  $\text{GNEXT}$  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  $\text{BLINK}[b]$ , for  $1 \leq b \leq N$ , and the next is  $\text{BNEXT}(\text{BLINK}[b])$ , etc. The contents of  $\text{BLINK}$  and  $\text{BNEXT}$  are therefore *not* fixed. Every girl  $g$  in the dag is the source of exactly one arc, which leads to  $\text{GMATE}[g]$ . If  $\text{GMATE}[g] = 0$ , 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  $\text{GMATE}$  and  $\text{BMATE}$  arrays. It also uses the auxiliary arrays  $\text{QUEUE}$ ,  $\text{IQUEUE}$ ,  $\text{MARK}$ ,  $\text{MARKED}$ , and  $\text{STACK}$ , defined above. The  $\text{MARK}$  array must be initially zero.

- H1.** [Prime the pump.] Set  $\text{GMATE}$  and  $\text{BMATE}$  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  $\text{QUEUE}$ .
- 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  $\text{QUEUE}[k]$  for  $i \leq k < q$ , 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  $g \leftarrow \text{QUEUE}[i]$ ,  $i \leftarrow i + 1$ , and  $e \leftarrow \text{GLINK}[g]$ .
- H5.** [Let  $b$  be a suitor for  $g$ .] If  $e = 0$ , return to H4; otherwise set  $b \leftarrow \text{BTIP}(e)$ .
- H6.** [Is  $b$  new?] If  $\text{MARK}[b] = 0$ , go to H8. Otherwise if  $\text{MARK}[b] > l$ , set  $\text{BNEXT}(e) \leftarrow \text{BLINK}[b]$ ,  $\text{BLINK}[b] \leftarrow e$ .
- H7.** [Loop on  $b$ .] Set  $e \leftarrow \text{GNEXT}(e)$  and return to H5.
- H8.** [Enter  $b$  into  $B_{l+1}$ .] If  $L > 0$  and  $\text{BMATE}[b] \neq 0$ , go to H7. Otherwise set  $\text{MARK}[b] \leftarrow l + 1$ ,  $\text{MARKED}[t] \leftarrow b$ ,  $t \leftarrow t + 1$ ,  $\text{BLINK}[b] \leftarrow e$ ,  $\text{BNEXT}(e) \leftarrow 0$ .
- H9.** [Is  $b$  free?] If  $\text{BMATE}[b] \neq 0$ , set  $\text{QUEUE}[q] \leftarrow \text{BMATE}[b]$ ,  $q \leftarrow q + 1$ . 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  $q \neq q'$ , set  $l \leftarrow l + 1$  and return to H3. (Otherwise the dag is complete, and the last  $r$  elements of  $\text{MARKED}$  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  $\text{MARK}[\text{MARKED}[k]] \leftarrow 0$  for  $0 \leq k < t$  and return to H2. Otherwise set  $b \leftarrow \text{MARKED}[t - r]$ ,  $r \leftarrow r - 1$ ,  $l \leftarrow L$ .
- H12.** [Enter level  $l$ .] Set  $\text{STACK}[l] \leftarrow b$ .
- H13.** [Advance.] Set  $e \leftarrow \text{BLINK}[b]$ , and go to H15 if  $e = 0$ . Otherwise set  $\text{BLINK}[b] \leftarrow \text{BNEXT}(e)$ ,  $g \leftarrow \text{GTIP}(e)$ . If  $\text{MARK}[\text{GMATE}[g]] < 0$ , repeat this step ( $g$  has been deleted). Otherwise set  $b \leftarrow \text{GMATE}[g]$ .
- H14.** [SAP complete?] If  $b = 0$  ( $g$  is free), go to H16. Otherwise set  $l \leftarrow l - 1$  and return to H12.

maximum cardinality matching

- H15.** [Resume higher level.] Set  $l \leftarrow l + 1$ . Then go to H11 if  $l > L$ ; otherwise set  $b \leftarrow \text{STACK}[l]$  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  $\text{STACK}[1] = b_1$  in a SAP. The other boys are  $\text{STACK}[2], \dots, \text{STACK}[L]$ .) Set  $f \leftarrow f - 1$ ,  $k \leftarrow \text{IQUEUE}[g]$ ,  $i \leftarrow \text{QUEUE}[f]$ ,  $\text{QUEUE}[k] \leftarrow i$ , and  $\text{IQUEUE}[i] \leftarrow k$ . (Those operations removed  $g$  from the list of free girls.) Set  $b \leftarrow \text{STACK}[1]$ .
- H17.** [Augment.] Set  $\text{MARK}[b] \leftarrow -1$ ,  $g' \leftarrow \text{BMATE}[b]$ ,  $\text{BMATE}[b] \leftarrow g$ , and  $\text{GMATE}[g] \leftarrow b$ . Then if  $g' \neq 0$ , set  $g \leftarrow g'$ ,  $l \leftarrow l + 1$ ,  $b \leftarrow \text{STACK}[l]$ , and repeat this step. Otherwise go back to H11. ■

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

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  $\text{QUEUE}[k]$  for  $0 \leq k < q$  are adjacent to only  $t$  boys in the graph, namely the boys in  $\text{MARKED}[k]$  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  $O((M + N + E)\sqrt{s})$ .*

*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.

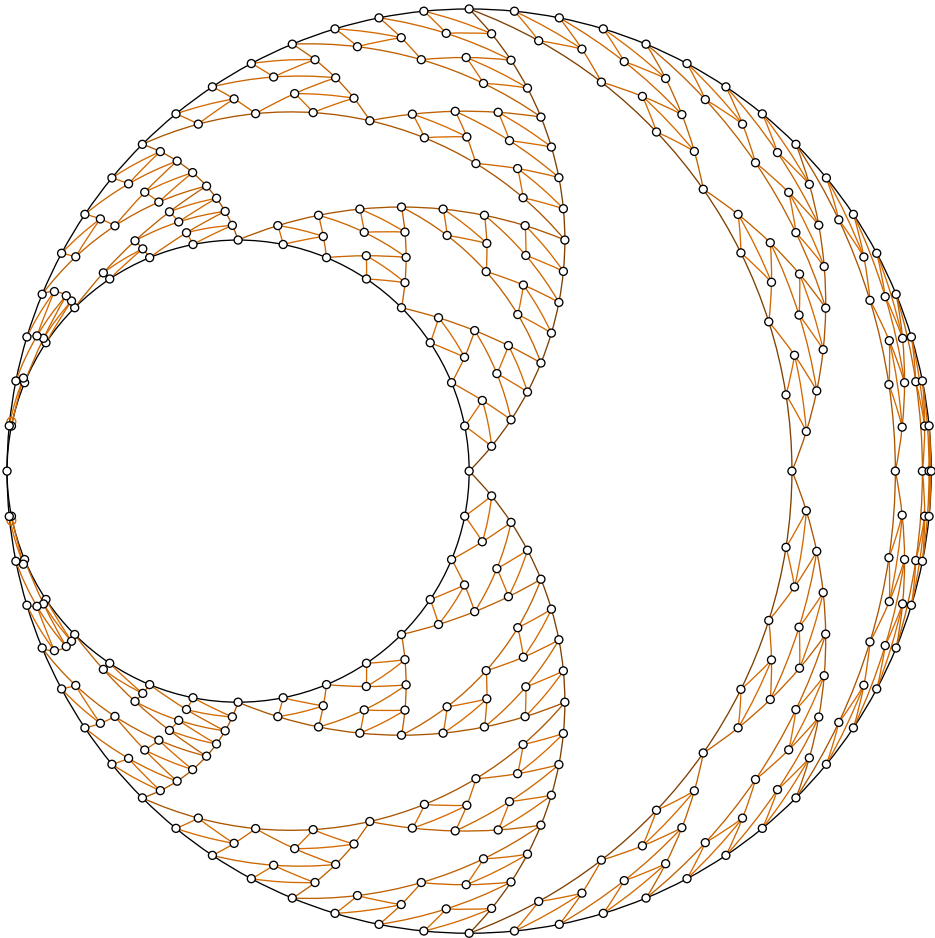
# CONTENTS

<b>Chapter 7 — Combinatorial Searching</b>	[4A.1]
7.2. Generating All Possibilities	[4A.281]
7.2.1. Generating Basic Combinatorial Patterns	[4A.281]
7.2.2. Backtrack Programming	[4B.30]
7.2.2.1. Dancing links	[4B.65]
7.2.2.2. Satisfiability	[4B.185]
7.2.2.3. Constraint satisfaction	1
Related models	2
*Statistical mechanics	4
A simple example	6
Automating automobiles	7
Line labeling in computer vision	9
Graph labeling	16
*Graceful digraphs	24
Graph embedding	28
*Supplemental labels and graphs	35
Special cases of subgraph isomorphism	37
Solving a CSP	38
Translating CSP to SAT	39
SAT encodings of general relations	46
Consistency	48
Efficiency	54
Representing the domains	56
*Dancing cells	61
*Dynamic variable ordering heuristics	65
*Maintaining XCC supports	69
Performance on benchmarks	73
*Sparse-set methods for MCC problems	76
Tractable families of CSPs	79
Implicational constraints	80
Max-closed constraints	82
CRC constraints	84
Polymorphisms	86
The indicator problem	87
NP-complete families of CSPs	90
*Polarities and clones	92
*The dichotomy theorem	96

*Exploiting a polymorphism . . . . .	99
*Visiting all solutions . . . . .	106
A brief history . . . . .	107
For further reading . . . . .	111
Exercises . . . . .	113
<b>Answers to Exercises . . . . .</b>	<b>158</b>
<b>Index to Algorithms and Theorems . . . . .</b>	<b>262</b>
<b>Answers to Puzzles in the Answers . . . . .</b>	<b>263</b>
<b>Index and Glossary . . . . .</b>	<b>264</b>

*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



*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 Holiness  
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*  $(x_1, x_2, \dots, x_n)$ , to which we can assign values that belong to given finite *domains*  $(D_1, D_2, \dots, D_n)$ . And we’re also given a set of *constraints*  $\{R_1, R_2, \dots, R_m\}$ , each of which specifies that a certain subset of the values  $(x_1, x_2, \dots, x_n)$  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:

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

Thus there are  $2 \times 2 \times 3 \times 2 \times 2 = 48$  possible settings of  $x_1 x_2 x_3 x_4 x_5$ , from BCAED to SLUON. Let’s also impose three constraints:

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

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  $F = \{1\bar{2}, 23, \bar{1}3, \bar{1}23\}$  in 7.2.2.2–(3) has four constraints,

$$\begin{aligned} x_1 x_2 &\in \{00, 10, 11\}; & x_2 x_3 &\in \{01, 10, 11\}; & x_1 x_3 &\in \{00, 01, 10\}; \\ x_1 x_2 x_3 &\in \{000, 001, 010, 011, 100, 101, 111\}. \end{aligned} \quad (3)$$

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  $\{0, 1, \dots, d-1\}$ , Section 7.2.2.2 discussed the “log encoding,” with  $l = \lceil \lg d \rceil$  Boolean variables meaning that  $x = (x_{l-1} \dots x_1 x_0)_2$ . There’s also the “direct encoding,” with  $d$  variables  $x_k = [x = k]$ , as well as the “order encoding,” which has  $d-1$  variables  $x^j = [x \geq j]$ . 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  $o_i \neq o_j$ , 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  $\{R_1, R_2, \dots\}$  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  $(D_1, \dots, D_n)$ , denoted by  $D_1 \times \dots \times D_n$ , is the set of all  $n$ -tuples  $(x_1, \dots, x_n)$  such that  $x_i \in D_i$  for  $1 \leq i \leq n$ . Thus,  $D_1 \times \dots \times D_n$  is the set of all solutions to a CSP with domains  $(D_1, \dots, D_n)$ , in the case when there are no constraints. An  $n$ -tuple such as  $(x_1, \dots, x_n)$  is often written simply as  $x_1 \dots x_n$ , when commas aren’t necessary. We also write  $D \times \dots \times D = D^n$  when the  $n$  domains are all identical.

A  $k$ -ary *relation* on sets  $(D_1, \dots, D_k)$  is a subset of  $D_1 \times \dots \times D_k$ . We write either  $R(x_1, \dots, x_k)$  or  $x_1 \dots x_k \in R$  when we want to say that the  $k$ -tuple  $(x_1, \dots, x_k)$  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  $u \rightarrow v$ , instead of writing  $A(u, v)$  or  $uv \in A$ .

Furthermore, if the lone binary relation is symmetric (meaning that  $u \rightarrow v$  implies  $v \rightarrow u$ ) and irreflexive (meaning that  $v \not\rightarrow v$ ), we usually call it  $E$  instead of  $A$ ; and we write  $u \sim v$  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  $h : V \rightarrow V'$  is called a *homomorphism* if  $E(u, v)$  implies  $E'(h(u), h(v))$ ; in other words, it's a homomorphism if we have

$$h(u) \sim h(v) \text{ in } G' \quad \text{whenever } u \sim v \text{ in } G. \quad (4)$$

For example, if  $G'$  is the complete graph  $K_d$  on vertices  $V' = \{1, 2, \dots, d\}$ , we have  $j \sim k$  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  $\{h(1), \dots, h(d)\}$  form a  $d$ -clique in  $G'$ .

Things get even more interesting when there's more than one relation. If  $S = (D, R_1, \dots, R_t)$  and  $S' = (D', R'_1, \dots, R'_t)$  are relational structures, we say that  $S$  and  $S'$  are *similar* if  $R_i$  and  $R'_i$  both have the same “arity,” for  $1 \leq i \leq t$ . (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

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

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:

$$h(u) \sim h(v) \text{ in } G' \text{ whenever } u \sim v \text{ in } G; \quad h(u) \neq h(v) \text{ whenever } u \neq v. \quad (6)$$

Consequently  $G$  is *embedded* in  $G'$ : The vertices  $\{h(v) \mid v \in V\}$  and edges  $\{h(u) \sim h(v) \mid u \sim v \text{ in } G\}$  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  $u \rightarrow v$  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

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

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

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  $(h(1), \dots, h(5)) = (x_1, \dots, x_5)$  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  $\Sigma = \sigma_1 \dots \sigma_N$  called a *configuration*, where  $\sigma_j$  is the  $j$ th 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  $\Sigma$  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

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

where each spin  $\sigma_j$  is  $\pm 1$ , 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

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

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

$\Sigma$	$E(\Sigma)$	$\Sigma$	$E(\Sigma)$	$\Sigma$	$E(\Sigma)$	$\Sigma$	$E(\Sigma)$	$\Sigma$	$E(\Sigma)$	$\Sigma$	$E(\Sigma)$
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  $\Sigma$ . 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  $\beta$  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  $\Sigma$ . [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,  $\beta$  is near 0; hence all configurations are almost equally likely. But at low temperatures,  $\beta$  approaches  $\infty$ ; 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  $\beta \rightarrow 0$ , but probability  $\frac{1}{2} - \frac{9}{2}e^{-\beta} + O(e^{-2\beta})$  when  $\beta \rightarrow \infty$ .

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.

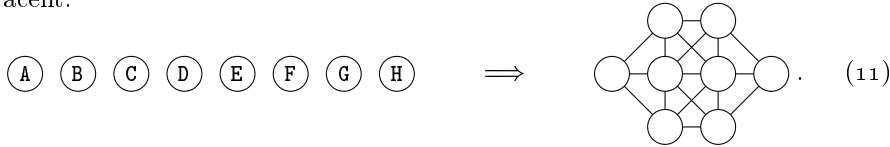
Considerations such as these account for the fact that physicists have contributed significantly to the understanding of random satisfiability problems, in particular by introducing Algorithm 7.2.2.2S. Further discussion of statistical mechanics is, of course, beyond the scope of a book on computer programming; 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

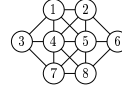
**A simple example.** To warm up, let's look at a little puzzle that appeared on a British TV show called *The Crystal Maze* in 1994. The task is simple—but you've got only two minutes to do it: “Place eight large disks, marked with the letters A through H, onto the eight circles shown; consecutive letters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

hello world  
crystal maze puzzle—  
global  
all-different



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  $\{x_1, x_2, \dots, x_8\}$ , one for each vertex, each with domain  $\{A, B, \dots, H\}$ . Then there are seventeen constraints, one for each edge of the graph; for example, the constraint for edge 1 — 2 is



$$x_1 x_2 \in \{AC, AD, AE, AF, AG, AH, BD, BE, BF, BG, BH, CA, CE, CF, CG, CH, DA, DB, DF, DG, DH, EA, EB, EC, EG, EH, FA, FB, FC, FD, FH, GA, GB, GC, GD, GE, HA, HB, HC, HD, HE, HF\}, \quad (12)$$

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  $\{A, B, \dots, H\}$ , one for each *disk*, each with domain  $\{1, 2, \dots, 8\}$ . Then only *seven* constraints are needed, one for each pair of consecutive letters; e.g.,

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

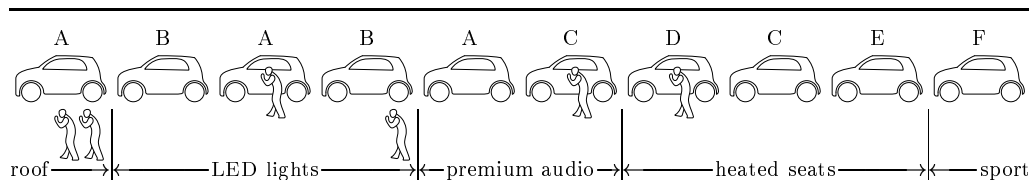
And each of these constraints has only 22 tuples, compared to 42 in (12). It's a win! Of course we also need the global all-different constraint. (See exercise 13.)

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.

**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  $w$ .*

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



**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?	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	2	5	(16)
LED lights?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	2	3	
premium audio?	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	2	1	2	
heated seats?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	3	1	3	
sport suspension?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	4	1	5	

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  $0 \leq i < t$ , 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  $\{0, 1, \dots, t-1\}$ .

Our example of the 10 cars in Fig. 100 and (16) would therefore have 10 variables  $\{x_0, \dots, x_9\}$  with the 6-element domain  $\{A, \dots, F\}$ , 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  $\{0, \dots, 9\}$ . These variables are related to each other by so-called “channelling constraints”: For example, we can’t have  $a_1 = j$  unless  $x_j = A$ ; 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  $\{A, A, A, B, B, C, C, D, E, F\}$ . 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  $x_7x_8x_9 = \text{CEF}$  that delays the lineup in Fig. 100. For this purpose it’s convenient to introduce Boolean variables  $f_{iw}$  for  $0 \leq i < t$  and  $0 \leq w < m$ , 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,  $x_i = B$  implies that  $f_{i0}f_{i1}f_{i2}f_{i3}f_{i4} = 10100$ . The assembly-line constraints are then

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

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

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

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

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  $t = 200$ ,  $m = 5$ ,  $(p_0/q_0, \dots, p_4/q_4) = (1/2, 2/3, 1/3, 2/5, 1/5)$ , 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

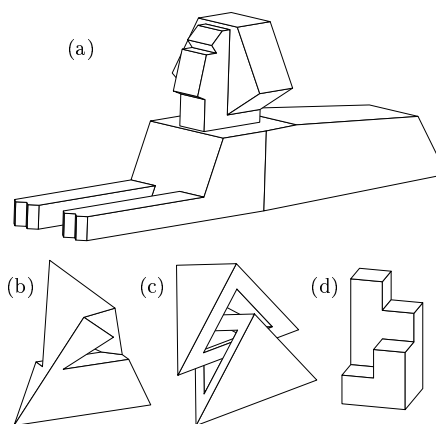
An international competition was held in 2005, based on actual industrial data. It included additional constraints, such as the colors of paint to be used and the initial contents of the assembly line, and it inspired many creative solutions. [See C. Solnon, V. D. Cung, A. Nguyen, and C. Artigues, *EJOR* **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.

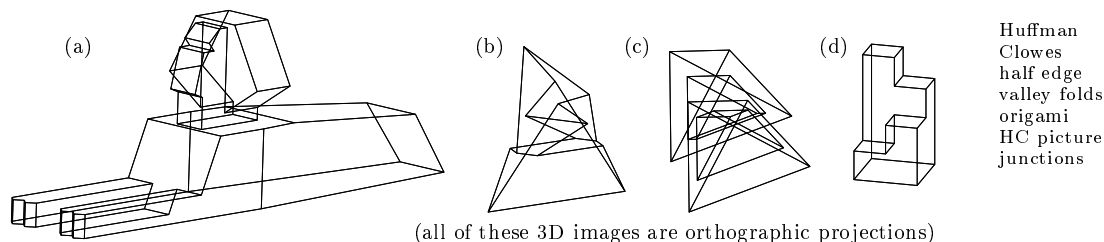
More precisely, the 3D objects we shall deal with have no curved surfaces. They are defined by vertices, edges, and faces, where the vertices are “corners” at which edges and faces come together. All of the faces are “flat,” meaning that their points all lie on some plane. Each face is bounded by an 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.

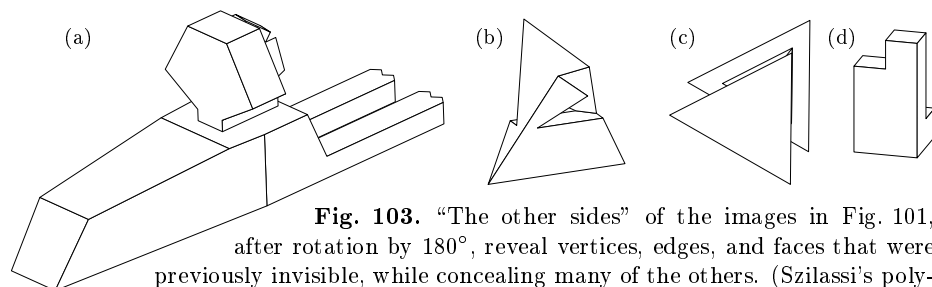


The two-dimensional images shown here make sense to us, somehow, although significant depth information has been lost. In some mysterious way we’ve learned to rely on visual cues in order to understand what’s really present.

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



**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  $180^\circ$ , 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  $180^\circ$  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:

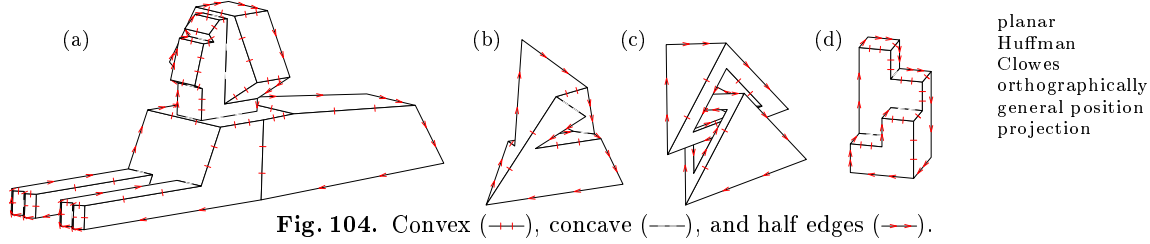
- a *convex* edge (+), where points between the adjacent faces belong to  $X$ ;
- a *concave* edge (−), where points between adjacent faces aren’t part of  $X$ ;
- a *half* edge ( $>$  or  $<$ ), where only one of its adjacent faces can be seen.

(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  $j - j'$  between designated junctions, for which (i) every junction has degree 2 or 3; (ii) two lines intersect only at junctions;



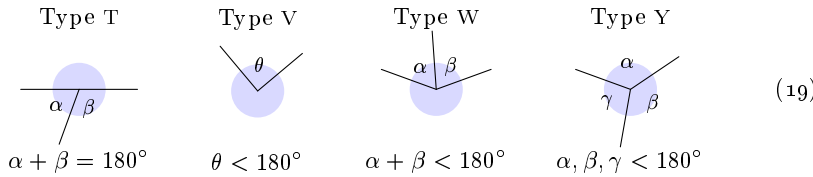


(iii) the two lines at a junction of degree 2 aren't collinear. Property (ii) means that the associated graph is planar. Property (iii) means that we can “see” all the junctions just by looking at the lines. (The “HC” in this definition stands for Huffman and Clowes.)

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:



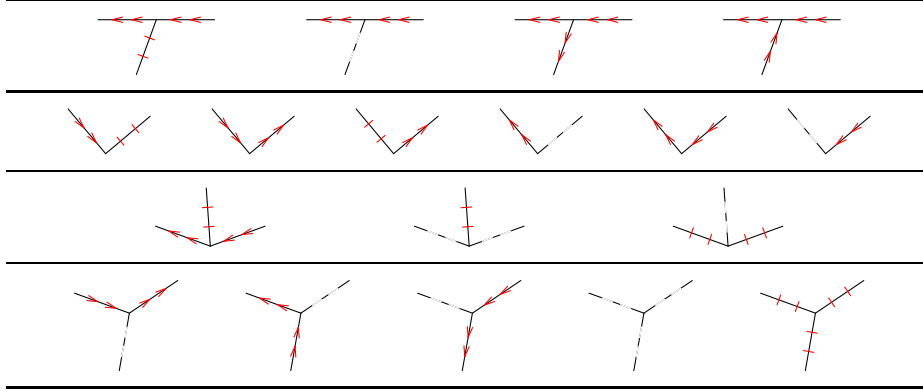
(Type T junctions are the artifacts of projection, mentioned above.)

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.

**Table 1**  
LEGAL LABELS FOR EACH JUNCTION TYPE

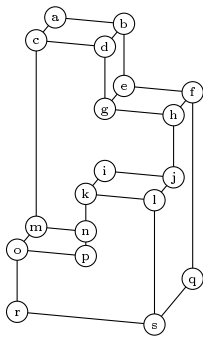


HC network  
line labeling problem

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):



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

(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 'd =  $Y(g, c, b)$ ' or 'd =  $Y(c, b, g)$ ' instead of 'd =  $Y(b, g, c)$ ' in (20); and there also are three equivalent ways to define each of the other  $Y$  junctions  $\{e, h, l, m\}$ . But 'd =  $Y(b, c, g)$ ' 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

oriented half edge ( $<$  or  $>$ ), 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  $>$ . This is, of course, a constraint satisfaction problem: The variables are the lines; the domains are the symbols  $\{+, -, <, >\}$ ; and the constraints are given by Table 1.

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

$$\{ab, ac, bd, be, cd, cm, dg, ef, eg, fh, fq, gh, hj, ij, ik, jl, kl, kn, ls, mn, mo, np, op, or, qs, rs\} \quad (21)$$

and 19 constraints defined by the following sets of tuples:

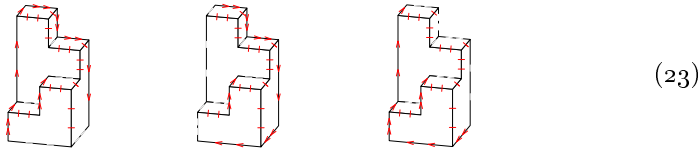
$$\begin{aligned} (ab, ac) &\in \{<+, <+, >+, >-, ><, -<\}; & (ik, kl, kn) &\in \{<+<, -+-, +-+\}; \\ (be, bd, ab) &\in \{>+<, -+-, +-+\}; & (jl, ls, kl) &\in \{<->, -<<, >>-, ---, +++ \}; \\ (ac, cd, cm) &\in \{<+<, -+-, +-+\}; & (cm, mn, mo) &\in \{<-<, -<<, >>-, ---, +++ \}; \\ (bd, dg, cd) &\in \{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

(Here ' $<+$ ' stands for the ordered pair  $(<, +)$ ; ' $>+>$ ' stands for  $(>, +, >)$ ; and so on.)

Notice that the constraint for junction  $b$  was not written ' $(be, bd, ba) \in \{>+<, -+-, +-+\}$ ', 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  $\{ab, ac, ba, bc, \dots, sr\}$  instead of 26, by introducing 26 further constraints such as  $(ab, ba) \in \{++ , -- , <> , ><\}$ . But that would have wasted time and space.

Notice also that the constraint for junction  $n$  was not written ' $(kn, mn, np) \in \{<+<, <-<, <<<, <><\}$ '. 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



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.

tuples  
boundary cycle  
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.

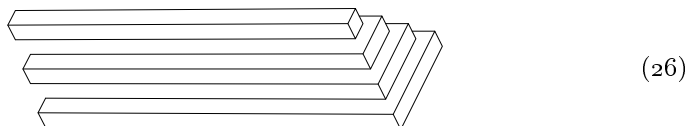
It's not difficult to construct HC pictures that *cannot* be labeled. For example, any picture that contains a subpicture of the forms

$$\begin{array}{c} \nearrow \\ \searrow \end{array} \text{ (TT) } \text{ or } \begin{array}{c} \nearrow \\ \nearrow \end{array} \text{ (VTT) } \text{ or } \begin{array}{c} \nearrow \\ \searrow \end{array} \text{ (YTT) } \text{ or } \begin{array}{c} \nearrow \\ \nearrow \\ \searrow \end{array} \text{ (WTVT) } \quad (24)$$

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

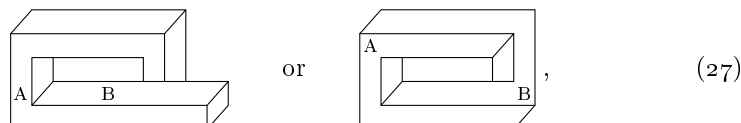
$$\begin{array}{c} \nwarrow \\ \nwarrow \end{array} \text{ (WT) } \text{ or } \begin{array}{c} \nwarrow \\ \nwarrow \\ \nearrow \end{array} \text{ (WWT) } \text{ or } \begin{array}{c} \nwarrow \\ \nwarrow \\ \nwarrow \end{array} \text{ (WWT) } \text{ or } \begin{array}{c} \nwarrow \\ \nwarrow \\ \nearrow \end{array} \text{ (WYT) } \quad (25)$$

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



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

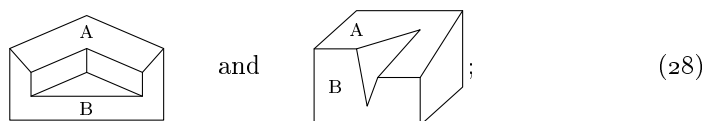


which look locally right although they're globally wrong. They “fail to compute” because each of them has two plane regions ('A' and 'B') that intersect in two *different* lines, contradicting a well-known principle of geometry.

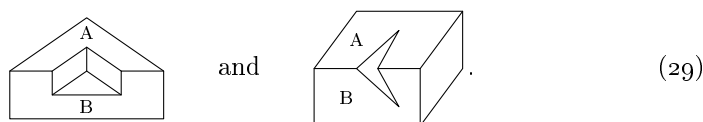
A somewhat subtle distinction arises here, noted by Huffman in his original paper of 1971: There are locally consistent pictures that are globally inconsistent

sphinx  
Szilassi polyhedron  
free boundary  
Reutersvärd  
intersection of planes+  
Huffman

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

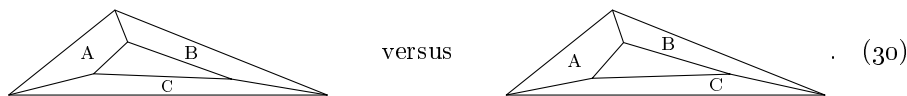


yet certain globally *consistent* pictures have exactly the same HC networks:



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

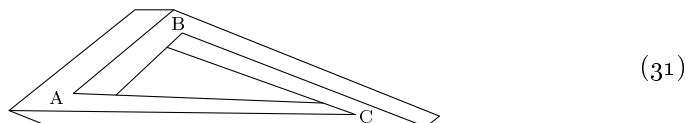


The left picture is strongly realizable, but the right picture is not! In this case three planes are involved ('A', 'B', 'C'); three of the lines show the intersections of planes AB, BC, and CA. Those three planes always intersect in a single point, ABC, because no two of them are parallel. The relevant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



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.

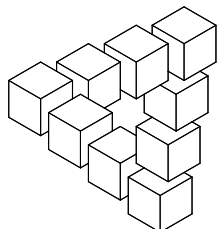
The most famous impossible HC picture is probably the “Penrose triangle”



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.

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

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)

Reutersvärd  
fun  
HUFFMAN  
Graph labeling  
graceful labeling  
complement

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)

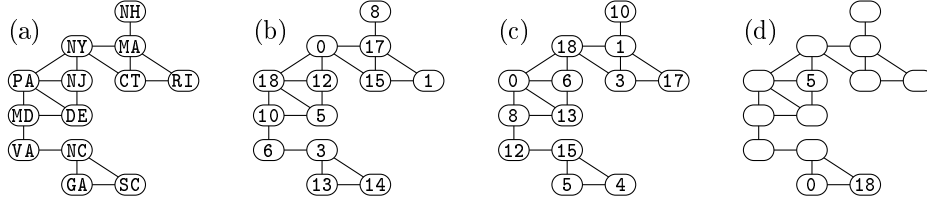
**Graph labeling.** Let’s turn now to a completely different but equally fascinating way to attach labels to the vertices and edges of a graph. Our new goal is to give an identifying number to each vertex while simultaneously identifying each edge.

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:

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

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  $0 \leq l(v) \leq m$ , 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  $v - w$ . 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  $\{1, \dots, m\}$ .

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  $\alpha$  is an automorphism (a permutation of the vertices for which  $v - w$  implies  $v\alpha - w\alpha$ ), 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  $0, 1, \dots, m$ , and that  $l(v) = v$  for  $0 \leq v \leq m$ . (In other words, a vertex’s label is also its name.) Then, if edge  $d$  connects vertices  $v$  and  $v + d$ , we set  $LO[d] \leftarrow v$ . Consequently *two arbitrary vertices  $v$  and  $w$  with  $v < w$  are adjacent if and only if  $LO[w - v] = v$* . With three further arrays, FIRST, NEXTL, and NEXTH, we can also visit all neighbors  $w$  of any given vertex  $v$  using a simple loop:

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

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

$$\begin{aligned} l &= 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \\ LO[l] &= - & 13 & 15 & 3 & 6 & 5 & 12 & 5 & 10 & 8 & 3 & 3 & 0 & 5 & 1 & 0 & 1 & 0 & 0 \\ \text{FIRST}[l] &= 12 & 15 & -1 & 6 & -1 & 10 & 3 & -1 & 17 & -1 & 6 & -1 & 18 & 14 & 13 & 17 & -1 & 15 & 12 \\ \text{NEXTL}[l] &= - & 3 & 8 & 10 & 5 & 18 & 10 & 0 & 5 & 1 & -1 & -1 & -1 & 0 & 0 & -1 & 0 & -1 & -1 \\ \text{NEXTH}[l] &= - & 3 & 1 & 13 & -1 & 12 & 5 & 18 & -1 & -1 & 14 & -1 & 15 & -1 & 17 & 17 & -1 & 18 & -1 \\ \text{NAME}[l] &= \text{NY} & \text{RI} & - & \text{NC} & - & \text{DE} & \text{VA} & - & \text{NH} & - & \text{MD} & - & \text{NJ} & \text{GA} & \text{SC} & \text{CT} & - & \text{MA} & \text{PA} \end{aligned} \quad (35)$$

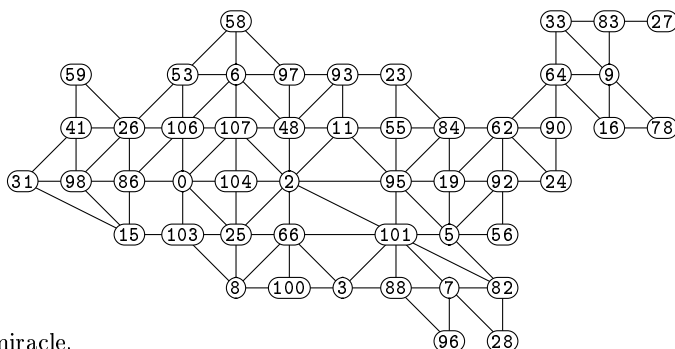
(The NAME array shown here gives an optional *external* name for printouts. Entries marked ‘—’ in these example arrays are unused.)

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.

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,  $\dots$ , 83, 27—the first 30 digits of  $\pi$ !! This has to be seen to be believed (see Fig. 106).

$C_5$   
contiguous USA graph  
USA graph  
randomized algorithm  
miracle  
 $\pi$   
Rokicki



**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  $\{0, \dots, m\}$  and the domain of each edge is  $\{1, \dots, m\}$ ; the constraints are that  $l(e) = |l(v) - l(w)|$  when  $e$  is the edge  $v - w$ ; 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  $\text{LO}$  and  $\text{NAME}$  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  $\{0, \dots, m\}$ ; and they all have the domain  $V \cup \{?\}$ , meaning that each label  $l$  should be assigned a  $\text{NAME}[l]$ , 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  $\{1, \dots, m\}$ ; and its value  $\text{LO}[l]$  has the domain  $\{0, \dots, m - l\}$ . The constraint is that

$$\text{NAME}[\text{LO}[l]] \text{ --- } \text{NAME}[\text{LO}[l] + l] \text{ is an edge of } G, \quad \text{for } 1 \leq l \leq m. \quad (36)$$

Let's call this the "reverse model."

The reverse model has a big advantage, because  $\text{LO}[l]$  has a very small domain when  $l$  is large. Indeed,  $\text{LO}[m]$  must be 0; and  $\text{LO}[m - 1]$  must be either 0 or 1. We can in fact assume that  $\text{LO}[m - 1] = 0$ , because complementation changes  $\text{LO}[m - 1]$  to  $1 - \text{LO}[m - 1]$ . (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  $\text{NAME}[\text{LO}[l]]$  and  $\text{NAME}[\text{LO}[l] + l]$  are both defined, meaning that  $\text{LO}[l]$  and  $\text{LO}[l] + l$  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  $\text{LO}[m] = 0$ . Hence 0 and  $m$  are real vertices, and we're done if  $n = 2$ .

Otherwise  $m > 1$ , and we may assume that  $\text{LO}[m - 1] = 0$  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  $\text{LO}[2] = 0$  and  $\text{LO}[1] = 2$ . That settles  $K_3$ .

If  $m > 2$ , edge  $m - 2$  is always either  $0 \text{ --- } (m - 2)$  or  $1 \text{ --- } (m - 1)$  or  $2 \text{ --- } m$ , 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  $\text{LO}[3]$  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  $n \leq 4$ , but  $K_n$  is ungraceful when  $n \geq 5$ .

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  $\text{LO}[n] = 0$ , and also  $\text{LO}[n - 1] = 0$ . But that means 0 must be the central vertex, because no other vertex has more than one neighbor. Consequently  $\text{LO}[n - 2] = 0$ ,  $\text{LO}[n - 3] = 0$ , 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  $p_0 p_1 \dots p_{n-1}$  of labels on the path is a permutation of  $\{0, 1, \dots, n - 1\}$ . The permutation  $p_0 p_1 \dots p_{n-1}$  is graceful if and only if

$$|p_0 - p_1| |p_1 - p_2| \dots |p_{n-2} - p_{n-1}| \text{ is a permutation of } \{1, \dots, n - 1\}. \quad (37)$$

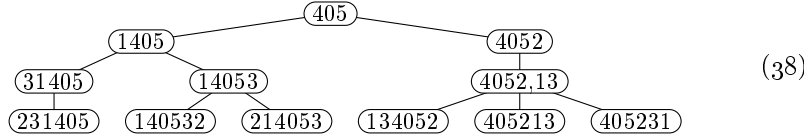
domain  
reverse model  
complementation  
*unique* graceful labeling  
star graph  
unique graceful labeling  
path graph  
graceful permutation

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  $p_0p_1 \dots p_{n-1}$ ,  $p_{n-1} \dots p_1p_0$ ,  $(n-1-p_0) \dots (n-1-p_{n-1})$ , and  $(n-1-p_{n-1}) \dots (n-1-p_0)$  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  $n > 2$ .

Let's take a look at  $P_6$ . We can assume that edges 5 and 4 will be  $0-5$  and  $0-4$ , which we can abbreviate to 05 and 04, respectively. Thus  $p_0p_1 \dots p_5$  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,



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

Notice that this procedure chooses the values of  $L0[5]$ ,  $L0[4]$ ,  $L0[3]$ , ... 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  $K_5 \square P_7$ :

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

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  $\{|0-96|, |96-4|, \dots, |22-84|\} = \{96, 92, 89, 88, 85, 79, 77, 66, 62\}$ . Those  $60+36$  differences are all distinct! And so are the  $70+30$  differences on the right!!

unique graceful labeling  
reversal  
complementation  
data structure  
ZDD  
KP graphs  
cliques

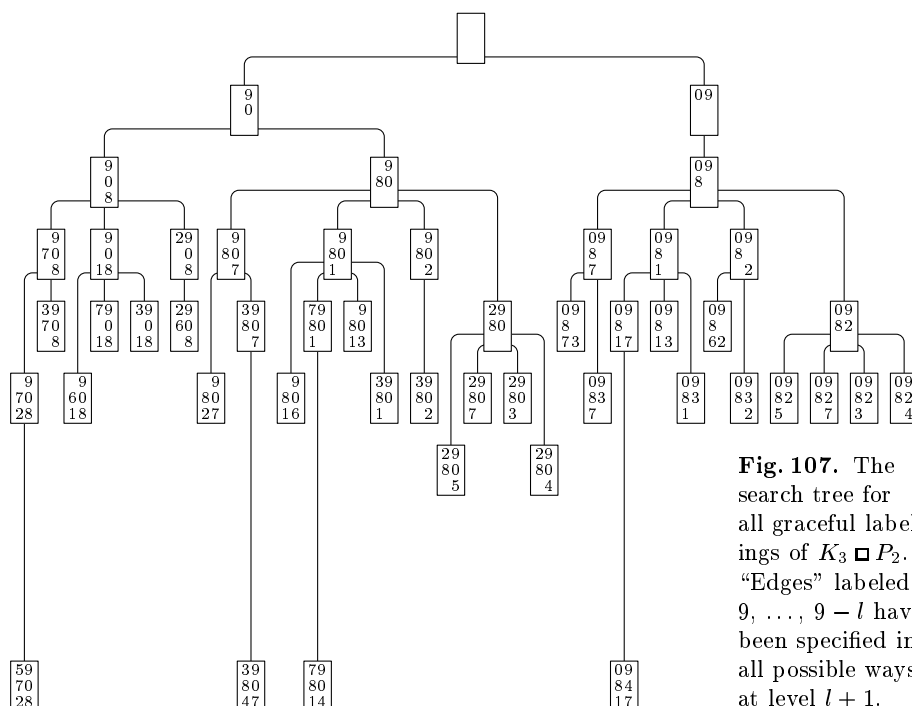
In general  $K_n \square P_r$  has  $rn$  vertices and  $m = r\binom{n}{2} + (r-1)n$  edges; that's exactly  $n^2$  edges when  $r = 2$ . It has  $2n!$  symmetries (aka automorphisms), because we can permute the rows of the matrix and/or reflect it left  $\leftrightarrow$  right.

Every graceful labeling of a KP graph can be represented as an  $n \times r$  matrix  $(x_{ij})$ , for  $1 \leq i \leq n$  and  $1 \leq j \leq r$ , 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

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

(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  $9, \dots, 9-l$  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  $3 \times 2$  matrix with the labels of vertices that we’ve chosen to be the endpoints of edges  $m, m-1, m-2, \dots$ . Sometimes the placement of a single new vertex will create more than one necessary edge (see exercise 77).

symmetries  
automorphisms  
complementation  
break this symmetry  
canonical form

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

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 K $\mu$ , 80 M $\mu$ , 3.6 G $\mu$ , 60 G $\mu$ , 360 G $\mu$ ) 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 T $\mu$  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.

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

**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  $K_5 \square C_5$ .

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

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  $\{1, 2, \dots, 75\}$ .

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  $r = 4, 6, \dots$  grows very rapidly: 3809, 41928684,  $\dots$ . 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  $\sum_{u \sim v} |l(u) - l(v)| = 1 + 2 + \dots + m = \binom{m+1}{2}$  when there are  $m$  edges; and a given vertex  $v$  appears exactly  $\deg(v)$  times in this sum. Working modulo 2, we also have  $|l(u) - l(v)| \equiv l(u) + l(v)$ . 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  $C_5, C_6, C_9, C_{10}, C_{13}, \dots$  can't be graceful.

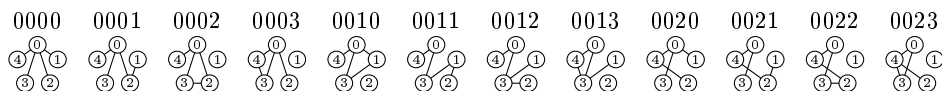
The reverse model tells us another basic fact about gracefulness in general:

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

*Proof.* There are exactly  $k+1$  ways to make  $0 \leq L_0[m-k] \leq k$ , for  $0 \leq k < m$ . ■

More precisely, if we insist that  $L_0[m-1] = 0$  in order to rule out complementary solutions, *there are exactly  $m!/2$  essentially distinct graceful labelings with  $m$  edges, for all  $m \geq 2$ .* [D. A. Sheppard, *Discr. Math.* **15** (1976), 379–388.]

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



Each instance is accompanied by its four-digit  $L_0$  string,  $L_0[4]L_0[3]L_0[2]L_0[1]$ . There are  $m+1$  vertices in general, namely  $\{0, 1, \dots, m\}$ ; 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  $\text{---}\text{---}\text{---}\text{---}$ , which is often called the “fork,” has three distinct labelings (cases 0001, 0012, 0020). The “paw”  $\text{---}\text{---}\text{---}$ , otherwise known as  $K_1 \text{ --- } (K_1 \oplus K_2)$ , 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.)

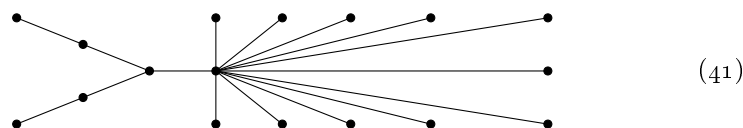
In particular, how many of those graceful labelings yield a *free tree* on the vertices  $\{0, 1, \dots, m\}$ ? Equivalently, how many of those  $m!/2$  sets of  $m$  edges have no cycles? In such cases no vertex is isolated. (See Theorem 2.3.4.1A.) The free trees shown above when  $m = 4$  are 0000, 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–(g). 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  $2 \times 10^{16}$  labelings per tree!

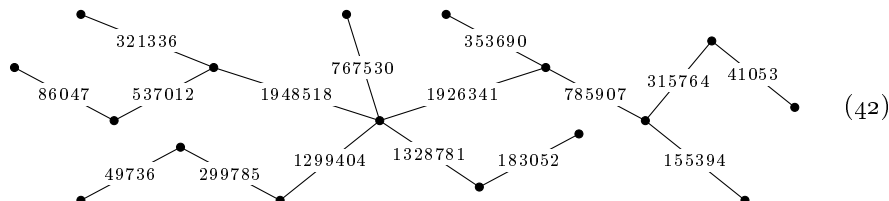
Sheppard  
isolated vertex  
symmetry  
unique graceful labeling  
fork  
paw  
free tree  
superexponentially  
Otter

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:



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:



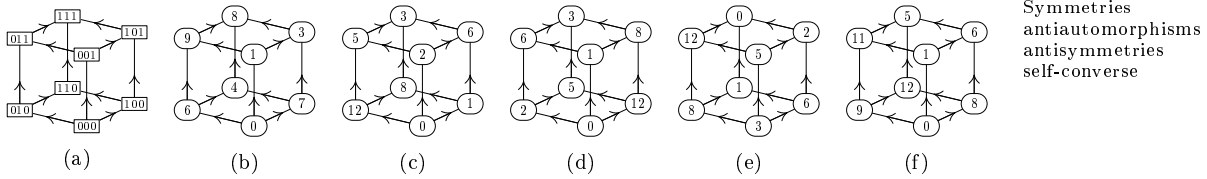
(Long edges seem to prefer vertices of high degree.) Anick’s analysis suggests strongly that all trees of larger sizes will also be easy to label.

**\*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  $0 \leq l(v) \leq m$ . But now we say that each directed arc  $v \rightarrow w$  implicitly receives the label  $(l(w) - l(v)) \bmod (m+1)$ , 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  $y \ominus x = (y - x) \bmod 13$ :

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

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



**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  $\mathbf{GA} \leftrightarrow \mathbf{SC}$  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  $\alpha$  of the vertices for which  $v \rightarrow w$  implies  $v\alpha \leftarrow w\alpha$ . 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  $b \bmod q$ , or (ii) multiplying by  $a \bmod q$  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.

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):

$$\begin{array}{rcccccccccccccc}
 l = & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
 \text{LO}[l] = & \text{—} & 0 & 1 & 6 & 4 & 3 & 0 & 0 & 1 & 7 & 7 & 6 & 9 \\
 \text{FIRST}[l] = & 7 & 9 & -1 & 8 & 8 & -1 & 4 & 4 & -1 & 8 & -1 & -1 & -1 \\
 \text{NEXT}[l] = & \text{—} & -1 & -1 & -1 & -1 & -1 & 1 & 6 & 3 & -1 & 3 & 9 & -1 \\
 \text{NAME}[l] = & 000 & 001 & \text{—} & 101 & 110 & \text{—} & 010 & 100 & 111 & 011 & \text{—} & \text{—} & \text{—}
 \end{array} \tag{44}$$

data structures  
digraph representation  
compact representation

As before, the general idea is to include isolated vertices if necessary so that the vertices of the graceful digraph  $D$  are  $\{0, 1, \dots, m\}$ , 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  $1 \leq l \leq m$ , we have  $\text{LO}[l] = v$  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  $v \rightarrow w$  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  $\text{LO}[(w - v) \bmod q] = v$ .*

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:

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

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

Every array LO with  $0 \leq \text{LO}[l] \leq m$  for  $1 \leq l \leq m$  defines a graceful digraph with  $m$  arcs on the vertices  $\{0, \dots, m\}$ . 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  $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_m$ , 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,  $(l(v_1) - l(v_0)) \bmod q + \dots + (l(v_m) - l(v_{m-1})) \bmod q$ , 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  $C_{4n+2}^\rightarrow$  is graceful by Theorem H. On the other hand, exercise 135 proves that the graphs  $C_{4n+1}$  are *not* digraceful.

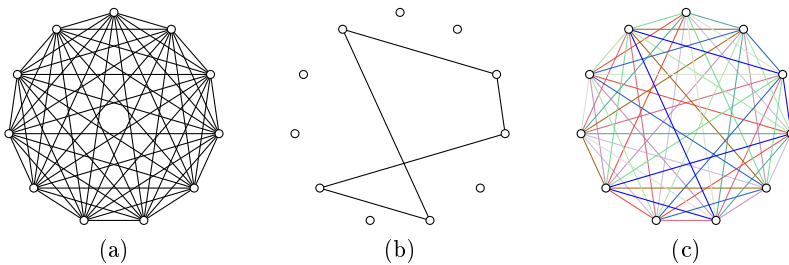
Is the complete graph  $K_n$  digraceful? This is probably the most interesting unsolved question about digracefulness, because every orientation of  $K_n$  is called a *tournament*. Graceful tournaments have been studied in other disguises, and they are known to exist for  $n = 1, 2, 3, 4, 5$ , and 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  $u-v$  has been replaced by two arcs  $u \rightarrow v$  and  $v \rightarrow u$ . 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  $u \rightarrow v$  and  $v \rightarrow u$  are then negatives of each other, modulo  $q$ ; and there are just  $m$  possibilities, namely  $\{\pm 1, \pm 2, \dots, \pm m\}$ . Consequently we define the label of edge  $u-v$  in  $G^\leftrightarrow$  to be

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

(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,  $\dots$ , 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,  $\dots$ , 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

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  $-m \leq l(v) \leq m$ , in such a way that the edge labels  $d_L(l(u), l(v))$  defined in (46) are distinct for all  $m$  edges  $u - v$ .

A graceful graph is automatically rainbow graceful, because  $d_L(l(u), l(v)) = |l(u) - l(v)|$  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  $n \geq 3$ , because of the elegant labeling

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

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  $v \neq w$  then  $f(v) \neq f(w)$ ;
- 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.

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

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  $v \not\sim w$  in  $G$  then  $f(v) \not\sim f(w)$  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

$$\text{share} \text{ --- } \text{spare} \text{ --- } \text{stare} \text{ --- } \text{store} \text{ --- } \text{shore} \text{ --- } \text{share}. \quad (48)$$

Formally we could say that the vertices of  $G$  are  $\{0, 1, 2, 3, 4\}$ , and that  $G$ 's edges are  $v \text{ --- } ((v+1) \bmod 5)$  for  $0 \leq v < 5$ ; then  $f(0) = \text{share}$ ,  $\dots$ ,  $f(4) = \text{shore}$ . 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  $\text{share} \text{ --- } \text{stare}$  in  $H$  but  $0 \not\sim 2$  in  $G$ . We could in fact have come up with a five-cycle such as

$$\text{share} \text{ --- } \text{shape} \text{ --- } \text{shade} \text{ --- } \text{shake} \text{ --- } \text{shame} \text{ --- } \text{share}, \quad (49)$$

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  $\{\text{right}, \text{might}, \text{night}, \text{light}, \text{sight}, \text{fight}, \text{tight}, \text{eight}\}$ ; 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,  $C_7$ :

$$\begin{aligned} \text{likes} \text{ --- } \text{lakes} \text{ --- } \text{cakes} \text{ --- } \text{caves} \\ \text{--- waves --- wives --- lives --- likes.} \end{aligned} \quad (50)$$

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

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

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.

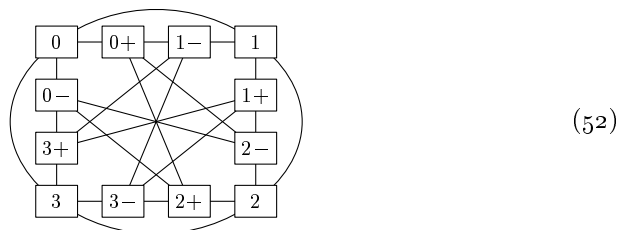
Let's write  $G \subseteq H$  if the SIP for pattern  $G$  and target  $H$  is solvable, and  $G \sqsubseteq H$  if the ISIP is solvable. (This is a slight abuse of notation; the relation  $G \subseteq H$  really means that  $G \cong H'$  for some  $H' \subseteq H$ , and  $G \sqsubseteq H$  really means that  $G \cong H|U$  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).

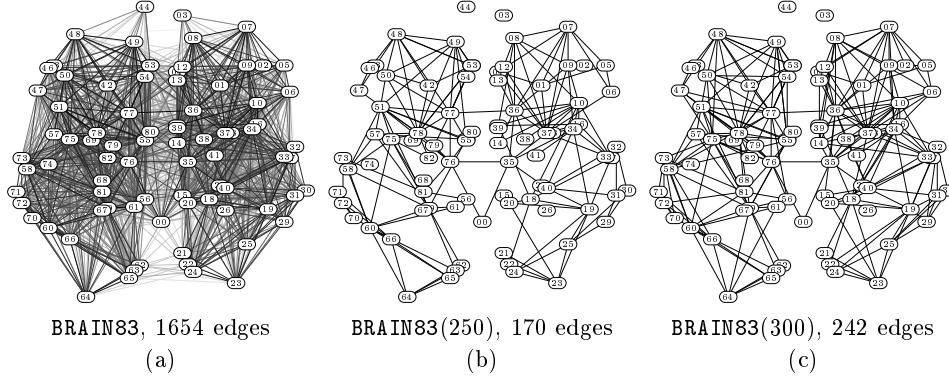
Consider, for example, the attractive 4-regular graph called Chvátal's graph. We looked at it long ago in Figure 2(f), near the beginning of Chapter 7; here it is again, with convenient names given to the vertices:



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

**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

$$'v \ V \ \bigcup \{e \cdot E \mid e = (u - v) \text{ and } E = (U \not- V) \text{ for some } u \text{ and } U\}'. \quad (53)$$

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  $u - v$  and  $U \not- V$ .

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).

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  $\geq 4$  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  $\geq 4$ . If we keep pruning vertices of low valency from the graph, we finally reach a graph  $\hat{H}$  that has only 22 vertices, all of degree  $\geq 4$ . 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  $G \not\subseteq \hat{H}$  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  $\hat{H}$  with only 58 vertices and 211 edges. Now we get an exact cover problem that Algorithm 7.2.2.1X can input in  $3 \text{ G}\mu$  and solve in  $8 \text{ G}\mu$ ; 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  $\alpha$  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  $1 \mapsto 48, 1+ \mapsto 49, 2- \mapsto 51, 2 \mapsto 47, \dots, 1- \mapsto 54$ , and to six others. (This solution does not belong to BRAIN83(298), because the edge  $48-54$  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:

$$\text{Is } \begin{array}{cccccc} \textcircled{00} & \textcircled{01} & \textcircled{02} & \textcircled{03} & \textcircled{04} \\ \textcircled{10} & \textcircled{11} & \textcircled{12} & \textcircled{13} & \textcircled{14} \\ \textcircled{20} & \textcircled{21} & \textcircled{22} & \textcircled{23} & \textcircled{24} \\ \textcircled{30} & \textcircled{31} & \textcircled{32} & \textcircled{33} & \textcircled{34} \end{array} \subseteq \begin{array}{c} \textcircled{00} \\ \textcircled{04} \textcircled{05} \textcircled{06} \textcircled{07} \textcircled{08} \textcircled{09} \\ \textcircled{10} \textcircled{11} \textcircled{12} \textcircled{13} \textcircled{14} \textcircled{15} \textcircled{16} \textcircled{17} \textcircled{18} \textcircled{19} \\ \textcircled{20} \textcircled{21} \textcircled{22} \textcircled{23} \textcircled{24} \textcircled{25} \textcircled{26} \textcircled{27} \textcircled{28} \textcircled{29} \\ \textcircled{30} \textcircled{31} \textcircled{32} \textcircled{33} \textcircled{34} \textcircled{35} \textcircled{36} \textcircled{37} \textcircled{38} \textcircled{39} \\ \textcircled{40} \textcircled{41} \textcircled{42} \textcircled{43} \textcircled{44} \textcircled{45} \textcircled{46} \textcircled{47} \textcircled{48} \textcircled{49} \\ \textcircled{50} \textcircled{51} \textcircled{52} \textcircled{53} \textcircled{54} \textcircled{55} \textcircled{56} \textcircled{57} \textcircled{58} \textcircled{59} \\ \textcircled{60} \textcircled{61} \textcircled{62} \textcircled{63} \textcircled{64} \textcircled{65} \textcircled{66} \textcircled{67} \textcircled{68} \textcircled{69} \end{array} ? \quad (54)$$

automorphisms  
essentially different solutions  
United States

On the left is  $P_4 \square P_5$ , a  $4 \times 5$  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  $4 \times 5$  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  $\{\text{CA, CT, DC, DE, FL, LA, ME, MI, ND, NH, NJ, RI, SC, VT, WA}\}$  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  $11 \mapsto \text{GA}$ . (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  $11 \mapsto \text{NY}$ , but it does show that we can't map  $12 \mapsto \text{NY}$  or  $22 \mapsto \text{NY}$ . 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  $12 \mapsto \text{MS}$ . For if we did, there would be a matching of size 4 in the bipartite graph



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  $11 \mapsto \text{OR}$ ,  $02 \mapsto \text{MA}$ , 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  $12 \mapsto \text{AZ}$ . 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  $\{\text{NM, NV, UT}\}$ ; but LAD filtering proves that impossible.

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

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

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

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.

GAD filtering  
all-different  
Sudoku

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:

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

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:

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

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



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

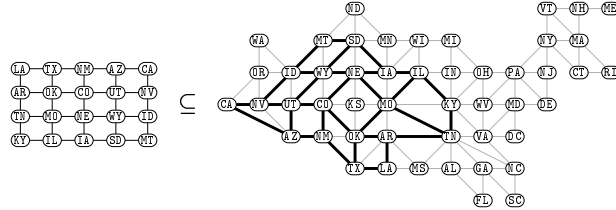
$$\begin{pmatrix} i & e & d & e & i \\ h & b & a & b & h \\ g & c & b & c & g \\ j & g & f & g & j \end{pmatrix}; \quad \begin{array}{ll} a = \{CO\}, & c = x \cup \{AZ, ID, MO, SD, TX\}, \\ b = d, & d = x \cup \{KS, NM, UT\}, \\ f = h, & e = c \cup \{KS\}, \\ g = i, & h = e \cup \{AR, IA, MT, NV, UT\},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

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

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

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

**Fig. 112.** One of the three ways to embed  $P_4 \square P_3$  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:

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

Here  $G^{\leq 2}$  denotes the graph whose vertices are the same as those of  $G$ , but whose edges  $u - v$  exist if and only if there's a path of length  $\leq 2$  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  $\leq 2$  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.

GAD filtering  
Supplemental  
McCreesh  
Prosser  
Trimble  
Chvátal

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:

If  $G \subseteq H$  via embedding function  $f$ , then  $d_G(v) \leq d_H(f(v))$  for all  $v \in G$ . (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  $\ell_G$  for which we can prove the following:

If  $G \subseteq H$  via embedding function  $f$ ,  
then  $\ell_G(u, v) \leq \ell_H(f(u), f(v))$  whenever  $u \text{ --- } v$  in  $G$ . (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,  $s \text{ --- } t$ , 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  $\ell_G^S$  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  $C_k$ ,  $\ell_G^S$  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  $u \text{ --- } v$  by the number  $\ell_G(u, v)$  of 4-cycles that it supports. Then  $\ell_G$  equals 2 on every internal edge of  $G = P_4 \square P_5$ ; and  $\ell_H$  has interesting diversity on the edges of  $H = \text{USA}$ . We can now, for example, prove that  $11 \mapsto \text{NY}$  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  $\ell_H$  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  $\{\text{CT}, \text{MA}, \text{VT}\}$ . Similar reasoning shows that  $11 \not\mapsto \text{AZ}$ , 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*  $\ell_G$  satisfies a stronger condition:

If  $G \subseteq H$  via embedding function  $f$ ,  
then  $\ell_G(u, v) \leq \ell_H(f(u), f(v))$  for all vertices  $u$  and  $v$  in  $G$ . (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  $\ell_G^S$  just as we did above. A supplemental pair label obtained in this way might turn out to be nonzero when  $u \text{ --- } v$ .

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

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:

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

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

For example, if  $\ell_G$  is a supplemental pair label, we get a supplemental graph by letting  $u \rightarrow v$  if and only if  $\ell_G(u, v) \geq k$ , for any threshold  $k$ . (And we conventionally write  $u \sim v$  if and only if we have both  $u \rightarrow v$  and  $v \rightarrow u$ .) Let's say that  $G^{S,k}$  is the supplemental graph we obtain in this way from the supplemental pair label  $\ell_G^S$ . (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  $\leq k$  if and only if  $G \subseteq P_n^k$ , where  $P_n^k$  is the graph on  $\{0, 1, \dots, n-1\}$  with  $u - v$  if and only if  $|u - v| \leq k$ .

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,  $P_2 \oplus \dots \oplus P_2$ , 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 *nonexact* 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

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

\* 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!

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.

**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  $\{0, 1, \dots, d-1\}$  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  $v_0 v_1 v_2$  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:

$$(v_0 \vee v_1 \vee v_2); \quad (66)$$

$$(\bar{v}_0 \vee \bar{v}_1) \wedge (\bar{v}_0 \vee \bar{v}_2) \wedge (\bar{v}_1 \vee \bar{v}_2). \quad (67)$$

Clause (66) says that  $v$  has *at least one* value, namely that  $v_0 + v_1 + v_2 \geq 1$ ; clauses (67) say that  $v$  has *at most one* value, namely that  $v_0 + v_1 + v_2 \leq 1$ . 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  $v_1 v_2 = \{00, 10, 01\}$  are distinct. In fact, if we read those pairs in the opposite order,  $v_2 v_1$ , 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,

$$(\bar{v}_1 \vee \bar{v}_2), \quad (68)$$

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.)

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

**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	$(\bar{u}^2 \vee u^1)$	$(\bar{v}^2 \vee v^1)$	$(u^1 \vee v^1)$ $(\bar{u}^1 \vee u^2 \vee \bar{v}^1 \vee v^2)$ $(\bar{u}^2 \vee \bar{v}^2)$

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

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

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  $u_2 u_1 v_2 v_1$  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  $u_0 u_1 v_0 v_1$  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  $v^1 v^2 = 00, 10, 11$  are equivalent to the log-encoded values  $v_2 v_1 = 10, 00, 01$ , 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  $\{0, 1, \dots, d-1\}$  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  $\alpha = a_1 \dots a_{n-1}$ , and its corners are labeled  $\alpha 0$  (top),  $\alpha 1$  (lower left),  $\alpha 2$  (lower right). Every vertex whose label has the form  $\alpha = a_1 \dots a_{k-1} a_k a_n \dots a_n$ , so that  $a_k \neq a_{k+1} = \dots = a_n$ , is pasted together with the vertex labeled  $\alpha' = a_1 \dots a_{k-1} a_n a_k \dots a_k$ . This rule gives two labels to all vertices, except for  $\{0 \dots 0, 1 \dots 1, 2 \dots 2\}$ ; hence there are  $(3^n + 3)/2$  distinct vertices altogether.

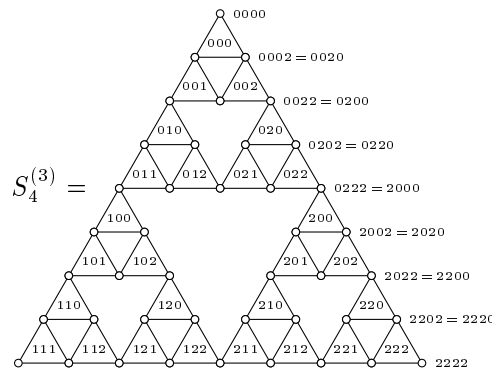


Figure 113 illustrates a family of graphs for which 3-coloring is particularly instructive. [See W. Sierpiński, *Comptes Rendus Acad. Sci.* **160** (Paris, 1915),

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  $\pm 30^\circ$  (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*  $\hat{S}_n^{(3)}$  to be the same as  $S_n^{(3)}$  but with the corner vertices  $0 \dots 0$  and  $1 \dots 1$  pasted together. This graph *cannot* be 3-colored. (Notice that  $\hat{S}_n^{(3)}$  has  $\lceil 3^n/2 \rceil$  vertices, each of which has degree 4 except for the remaining corner vertex  $2 \dots 2$ ; 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  $\hat{S}_n^{(3)}$ , with each encoding. We might save a factor of six if we introduce clauses to force the colors of the top three vertices  $0 \dots 00$ ,  $0 \dots 01$  and  $0 \dots 02$  (see exercise 224).

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

Log  $\approx$  Reduced  $<$  Prefix  $\approx$  Direct  $\approx$  Multi  $\approx$  Support  $<$  Weakened  $\ll$  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  $\hat{S}_9^{(3)}$ , 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  $\hat{S}_n^{(3)}$  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

$$(u_0 \vee v_0 \vee w_0) \wedge (u_1 \vee v_1 \vee w_1) \wedge (u_2 \vee v_2 \vee w_2) \quad (69)$$

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  $\hat{S}_9^{(3)}$  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

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

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:

$$\begin{array}{cccccccccc} \widehat{S}_3^{(3)} & \widehat{S}_4^{(3)} & \widehat{S}_5^{(3)} & \widehat{S}_6^{(3)} & \widehat{S}_7^{(3)} & \widehat{S}_8^{(3)} & \widehat{S}_9^{(3)} & \widehat{S}_{10}^{(3)} & \widehat{S}_{11}^{(3)} \\ 1.36 \text{ K}\mu & 27.6 \text{ K}\mu & 345 \text{ K}\mu & 2.98 \text{ M}\mu & 23.0 \text{ M}\mu & 299 \text{ M}\mu & 4.77 \text{ G}\mu & 72.9 \text{ G}\mu & 1460 \text{ G}\mu \end{array}$$

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  $2.34 \text{ G}\mu$  (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 *waarden*(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

$$\begin{aligned} \alpha = 0.6, \quad \rho = 0.6, \quad \varrho = 0.99, \quad \Delta_p = 10000, \quad \delta_p = 5000, \\ \tau = 20, \quad w = 1, \quad p = 0.02, \quad P = 0, \quad \psi = 0.15. \end{aligned} \quad (71)$$

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

$$\begin{array}{cccccccccc} \widehat{S}_3^{(3)} & \widehat{S}_4^{(3)} & \widehat{S}_5^{(3)} & \widehat{S}_6^{(3)} & \widehat{S}_7^{(3)} & \widehat{S}_8^{(3)} & \widehat{S}_9^{(3)} & \widehat{S}_{10}^{(3)} & \widehat{S}_{11}^{(3)} \\ 1.84 \text{ K}\mu & 49.0 \text{ K}\mu & 583 \text{ K}\mu & 2.84 \text{ M}\mu & 18.0 \text{ M}\mu & 90.8 \text{ M}\mu & 521 \text{ M}\mu & 2.27 \text{ G}\mu & 13.2 \text{ G}\mu \end{array}$$

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  $13 \text{ G}\mu$  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  
*waarden*  
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  $fsnark(q)$ , 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  $q \leq 99$ ; 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 M $\mu$	1.10 G $\mu$	4.66 G $\mu$	21.2 G $\mu$	48.2 G $\mu$	171.7 G $\mu$	630 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  $(\alpha, \rho, \varrho, \dots, \psi)$  to the various encodings. SAT solvers are full of surprises!

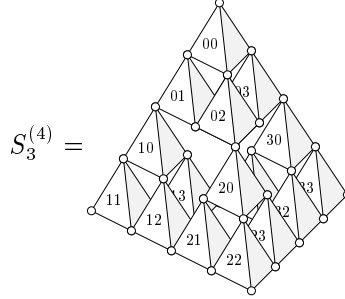
So far we’ve been looking only at ternary domains. Domains of size 4 or more lead of course to many further questions, with seemingly endless possibilities to explore. The encodings for  $d = 3$  in Table 2 can be extended to arbitrary  $d$  in interesting ways (see exercise 240). And the graphs  $S_n^{(3)}$  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  $0 \dots 0$  and  $1 \dots 1$  together as we did before. Exercise 239 points out that the graph  $\widehat{S}_n^{(d)}$  cannot be

flower snark line graphs  
line graphs  
*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

**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  $\alpha = a_1 \dots a_{n-1}$ , and its “corner vertices” are labeled  $\alpha j$  for  $0 \leq j < d$ . Each vertex has two  $d$ -ary labels  $\alpha$  and  $\alpha'$  as before, except for the  $d$  “pure” vertices labeled  $j \dots j$ ; exercise 234 gives examples. Therefore there are  $(d^n + d)/2$  vertices altogether.



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

$d$ -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

$$\overline{S}_n^{(d)} = S_n^{(d)} \text{ plus } d-1 \text{ edges } 0 \dots 0 - j \dots j \text{ for } 0 < j < d. \quad (72)$$

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  $n \leq 7$  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 K $\mu$	1.99 M $\mu$	23.8 M $\mu$	1.16 G $\mu$	135 G $\mu$

(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 M $\mu$	1.78 G $\mu$	17.9 G $\mu$	172 G $\mu$

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 ‘ $u \leq v$ ’—or perhaps ‘ $u < v$ ’, which is the same as ‘ $u \leq v - 1$ ’. The order encoding is particularly good for constraints such as this. Indeed, in the  $d$ -ary domain  $\{0, 1, \dots, d-1\}$ , with the Boolean variable  $u^j$  standing for  $[u \geq j]$ , the relation ‘ $u \leq v - t$ ’ for any fixed  $t$  is equivalent to the clauses

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

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

$$'u < v' \iff (v^1) \wedge (\bar{u}^1 \vee v^2) \wedge (\bar{u}^2 \vee v^3) \wedge \dots \wedge (\bar{u}^{d-2} \vee v^{d-1}) \wedge (\bar{u}^{d-1}). \quad (74)$$

And we can even go much further: Exercises 7.2.2.2–405 and 406 give encodings for ‘ $au + bv \leq c$ ’ as well as ‘ $uv \leq a$ ’ and ‘ $uv \geq a$ ’, 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 ‘ $u + v \leq w$ ’, when all three variables are order-encoded.

There’s also a good way to translate the relation ‘ $u \leq v$ ’ into SAT clauses when  $u$  and  $v$  have the log encoding, thanks to Eq. 7.2.2.2–(169). For example, suppose  $d = 16$ ,  $u = (u_8 u_4 u_2 u_1)_2$ , and  $v = (v_8 v_4 v_2 v_1)_2$ , using four bits to represent each variable. Then we have  $u \leq v$  if and only if

$$(\bar{u}_8 \vee v_8) \wedge (\bar{u}_8 \vee a_1) \wedge (v_8 \vee a_1) \wedge (\bar{a}_1 \vee \bar{u}_4 \vee v_4) \wedge (\bar{a}_1 \vee \bar{u}_4 \vee a_2) \wedge (\bar{a}_1 \vee v_4 \vee a_2) \wedge \\ (\bar{a}_2 \vee \bar{u}_2 \vee v_2) \wedge (\bar{a}_2 \vee \bar{u}_2 \vee a_3) \wedge (\bar{a}_2 \vee v_2 \vee a_3) \wedge (\bar{a}_3 \vee \bar{u}_1 \vee v_1). \quad (75)$$

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  $v \leq w$ , we’d need to introduce auxiliaries called  $a_4, a_5$ , and  $a_6$ , say.) Exercise 248 shows that a similar scheme can encode ‘ $u \leq v - t$ ’ for any  $t$ .

In general, however, a CSP can involve arbitrary constraints that don’t have nice properties like the relation ‘ $u \leq v - t$ ’. 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 ‘ $u \neq v$ ’ 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  $(a_1, \dots, a_k)$  don’t satisfy a given relation on the variables  $(v_1, \dots, v_k)$ , the preclusion clause is  $(\bar{v}_{1a_1} \vee \dots \vee \bar{v}_{ka_k})$ .

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.* 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”

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

(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  $R(v_1, \dots, v_k)$ . 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  $\{a_1 \dots a_k \mid R(a_1, \dots, a_k)\}$  of all tuples that satisfy  $R$ . If  $a \in D_{v_j}$  and  $a_1 \dots a_k \in D_R$ , then we have

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

(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

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

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

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

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

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

- *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:

$$uv \in \{00, 01, 02, 12, 21\}; \quad (83)$$

$$uw \in \{00, 01, 02, 11, 21\}; \quad (84)$$

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

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,

$$(\bar{u}_0 \vee \bar{v}_1 \vee \bar{w}_1), \quad (86)$$

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  $\bar{R}_{000}$ ,  $\bar{R}_{001}$ ,  $\bar{R}_{010}$ ,  $\bar{R}_{012}$ ,  $\bar{R}_{020}$ ,  $\bar{v}_0$ ,  $\bar{w}_0$ ,  $\bar{w}_2$ ; 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.

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:

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

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):

$$\begin{array}{llll} \bar{w}_0 \wedge \bar{w}_1 \Rightarrow \bar{x}_2 & \bar{w}_1 \wedge \bar{w}_2 \Rightarrow \bar{z}_2 & \bar{x}_0 \wedge \bar{x}_2 \Rightarrow \bar{y}_1 & \bar{y}_2 \Rightarrow \bar{z}_0 \\ \bar{x}_0 \wedge \bar{x}_1 \Rightarrow \bar{w}_2 & \bar{z}_0 \wedge \bar{z}_1 \Rightarrow \bar{w}_0 & \bar{x}_0 \Rightarrow \bar{y}_2 & \Rightarrow \bar{z}_1 \\ \bar{w}_0 \Rightarrow \bar{y}_0 & \bar{x}_1 \wedge \bar{x}_2 \Rightarrow \bar{z}_0 & \bar{y}_0 \Rightarrow \bar{x}_1 & \bar{z}_2 \Rightarrow \bar{y}_0 \\ \bar{y}_1 \wedge \bar{y}_2 \Rightarrow \bar{w}_1 & \bar{x}_1 \wedge \bar{x}_2 \Rightarrow \bar{z}_2 & \bar{y}_0 \wedge \bar{y}_1 \Rightarrow \bar{x}_2 & \bar{z}_2 \Rightarrow \bar{y}_1 \\ \bar{y}_1 \wedge \bar{y}_2 \Rightarrow \bar{w}_2 & \bar{z}_1 \Rightarrow \bar{x}_0 & & \bar{z}_0 \wedge \bar{z}_2 \Rightarrow \bar{y}_2 \end{array} \quad (88)$$

(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,

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

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.

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

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  $\alpha$  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  $\alpha$ . 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  $\alpha$  has no solution; recursion stops. Otherwise, when node  $\alpha'$  is a child of node  $\alpha$ , 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,  $\{a_1, \dots, a_d\}$ . Then node  $\alpha$  has  $d$  children,  $\alpha_j$  for  $1 \leq j \leq d$ , and we have  $D_{\alpha_j,v} = \{a_j\}$ . We say that  $v$  *has been assigned the value*  $a_j$  in branch  $\alpha_j$ . (ii) A *binary branch on*  $v=a$  can be made whenever  $a \in D_{\alpha,v}$ . Then node  $\alpha$  has two children,  $\alpha_=\$  and  $\alpha_\neq$ , and we have  $D_{\alpha_-=,v} = \{a\}$ ,  $D_{\alpha_\neq,v} = D_{\alpha,v} \setminus a$ . 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  $\alpha_\neq$  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  $\alpha$  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  $\alpha$  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  $\alpha$  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,  $\beta$  is also called an *assignment*.

**Definition V.** The binding  $(v, a)$  is said to be “viable” in subproblem  $\alpha$  when every constraint involving  $v$  contains at least one tuple  $\tau$  such that (i)  $v = a$  in  $\tau$ , and (ii) every other variable  $v'$  in that constraint has a value  $v' = a'$  in  $\tau$  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  $\alpha$  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).

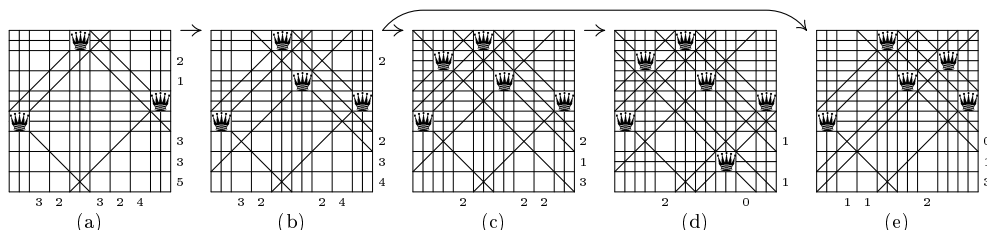
recursively  
current domain  
branch variable  
assigned  
binary branch  
instantiation, see assignment  
inactive  
active  
binding  
viable  
weakly viable  
forward consistency-  
domain consistency-



• *Forward consistency* holds at node  $\alpha$  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.

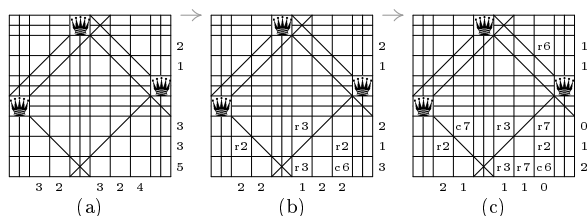
• *Domain consistency* holds at node  $\alpha$  if and only if every active binding is viable. In other words, no active variable  $v$  has a value  $a$  in its domain for which the assignment  $v = a$  would “wipe out” (reduce to  $\emptyset$ ) the domain of any other active variable. In other words, 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.)

wipe out  
supporting tuple  
8 queens problem



**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  $\{r_1, \dots, r_8, c_1, \dots, c_8\}$ . 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  $R(v, v_1, \dots, v_k, w)$ , for  $k \geq 0$ , where  $w$  is currently active but we’ve already assigned  $v_1=a_1, \dots, v_k=a_k$ . Whenever  $R$  is such a constraint, we simply restrict the domain of  $w$  to values for which  $aa_1 \dots a_kw \in R(v, v_1, \dots, v_k, w)$ .

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  $\{+, -, >, <\}$ ; hence it begins with  $26 \cdot 4 = 104$  active bindings, namely  $(ab, +)$  through  $(rs, <)$ . 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  $(kn, np) \in \{<<\}$  supports only one value in the domains of variables  $kn$  and  $np$ . Thus we can immediately shrink  $D_{kn}$  and  $D_{np}$  to  $\{<\}$ .

Furthermore the ternary constraint  $(be, bd, ab) \in \{>+>, -+-, +-+\}$  allows us to shrink  $D_{bd}$  to  $\{+, -\}$ , while  $D_{be}$  and  $D_{ab}$  become  $\{+, -, >\}$ . 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  $(ab, ac)$  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:

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

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  $\text{revise}(c, v)$  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"  $\text{STAMP}(v)$  and  $\text{STAMP}(c)$  to each variable and each constraint. Variables also have a field  $\text{INQ}(v)$ , 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  $\text{STAMP}(c) \leftarrow 0$  for each constraint  $c$ ;  $\text{STAMP}(v) \leftarrow \text{INQ}(v) \leftarrow [v \text{ is active}]$  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  $\text{INQ}(v) \leftarrow 0$ .
- D3.** [Loop over constraints.] Do steps D4–D5 for every constraint  $c$  for which  $v \in S_c$  and  $\text{STAMP}(v) > \text{STAMP}(c)$ . 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  $\text{STAMP}(c) \leftarrow t$  and  $t \leftarrow t + 1$ , then return to D3.
- D6.** [Revise  $w$ .] Do  $\text{revise}(c, w)$  (see (90)). If that routine doesn't change  $D_w$ , do nothing more. Otherwise, if  $D_w = \emptyset$ , terminate with wipeout. Otherwise set  $\text{STAMP}(w) \leftarrow t$ ; and if  $\text{INQ}(w) = 0$ , also set  $Q \leftarrow w$  and  $\text{INQ}(w) \leftarrow 1$ . ■

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

The elements  $v \in Q$  whose domain has changed since  $c$  was certified are precisely those with  $\text{STAMP}(v) > \text{STAMP}(c)$ . 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:

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

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.)

**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—analogue 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  $9\text{ G}\mu$  to find all solutions. That was a winner over Algorithm 7.2.2.1X, which needed  $40\text{ G}\mu$  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  $26\text{ G}\mu$  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:

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

where ‘ $\{\text{colored items}\}$ ’ 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  $x_i$ , 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  $i_1, \dots, i_t$  that are accessible from  $\text{RLINK}(0)$  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  $\text{LEN}(i)$ .

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  $x_0 x_1 \dots x_{n-1}$  such that we have

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

This problem has  $n$  variables  $x_k$ , each with domain  $D_k = \{0, 1, \dots, d-1\}$ , 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  $\{0$

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 megamems. 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  $p_1 p_2 \dots p_n$  of  $\{1, 2, \dots, n\}$  for which we have

$$p_{k+1} \leq p_k + 1 \quad \text{for } 1 \leq k < n. \quad (94)$$

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  $n \geq 8$ , and on the slow growth problem for all  $n \geq 12$ .

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.

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

	$x$	$y$	Trail	$\sigma$	$x, x'$	$y, y'$	Trail
01	begin $\alpha$		0 0	1	0,0	0,0	
02	$x \leftarrow 1$		1 0	1	1,1	0,0	$x$
03	$y \leftarrow x$		1 1	1	1,1	1,1	$x$ $y$
04	$x \leftarrow 2$		2 1	1	2,1	1,1	$x$ $y$
05	begin $\alpha_1$		2 1	2	2,1	1,1	$x$ $y$
06	$y \leftarrow x$		2 2	2	2,1	2,2	$x$ $y$   $y$
07	$x \leftarrow 3$		3 2	2	3,2	2,2	$x$ $y$   $y$ $x$
08	back to $\alpha$		2 1	3	2,2	1,2	$x$ $y$   $y$ $x$
09	$x \leftarrow y$		1 1	3	1,3	1,2	$x$ $y$ $x$
10	begin $\alpha_2$		1 1	4	1,3	1,2	$x$ $y$ $x$
11	$y \leftarrow 4$		1 4	4	1,3	4,4	$x$ $y$ $x$   $y$
12	$y \leftarrow x$		1 1	4	1,3	1,4	$x$ $y$ $x$   $y$
13	begin $\alpha_{21}$		1 1	5	1,3	1,4	$x$ $y$ $x$   $y$
14	$x \leftarrow 5$		5 1	5	5,5	1,4	$x$ $y$ $x$   $y$   $x$
15	back to $\alpha_2$		1 1	6	1,5	1,4	$x$ $y$ $x$   $y$   $x$
16	$y \leftarrow 2$		1 2	6	1,5	2,6	$x$ $y$ $x$   $y$   $x$
17	$y \leftarrow x$		1 1	6	1,5	1,6	$x$ $y$ $x$   $y$   $x$
18	back to $\alpha$		1 1	7	1,5	1,6	$x$ $y$ $x$   $y$   $x$
19	$x \leftarrow 8$		8 1	7	8,7	1,6	$x$ $y$ $x$   $y$   $x$
20	begin $\alpha_3$		8 1	8	8,7	1,6	$x$ $y$ $x$   $y$   $x$
21	$y \leftarrow x$		8 8	8	8,7	8,8	$x$ $y$ $x$   $y$   $x$
22	$x \leftarrow 3$		3 8	8	3,8	8,8	$x$ $y$ $x$   $y$   $x$
23	$y \leftarrow 5$		3 5	8	3,8	5,8	$x$ $y$ $x$   $y$   $x$
24	begin $\alpha_{31}$		3 5	9	3,8	5,8	$x$ $y$ $x$   $y$   $x$
25	$y \leftarrow 4$		3 4	9	3,8	4,9	$x$ $y$ $x$   $y$   $x$
26	back to $\alpha_3$		3 5	10	3,8	5,9	$x$ $y$ $x$   $y$   $x$
27	$x \leftarrow y$		5 5	10	5,10	5,9	$x$ $y$ $x$   $y$   $x$
28	begin $\alpha_{32}$		5 5	11	5,10	5,9	$x$ $y$ $x$   $y$   $x$
29	$y \leftarrow 4$		5 4	11	5,10	4,11	$x$ $y$ $x$   $y$   $x$
30	back to $\alpha_3$		5 5	12	5,10	5,11	$x$ $y$ $x$   $y$   $x$
31	$x \leftarrow 6$		6 5	12	6,12	5,11	$x$ $y$ $x$   $y$   $x$
32	$y \leftarrow x$		6 6	12	6,12	6,12	$x$ $y$ $x$   $y$   $x$
33	back to $\alpha$		8 1	13	8,12	1,12	$x$ $y$ $x$   $y$   $x$
34	$y \leftarrow 7$		8 7	13	8,12	7,13	$x$ $y$ $x$   $y$   $x$
35	back out		0 0	14	0,12	0,13	$x$ $y$ $x$   $y$   $x$

**Fig. 117.** Reversible storage is implemented by keeping a trail of changes that need to be undone. An entry like  $\begin{smallmatrix} x \\ 0 \end{smallmatrix}$  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  $\sigma$ .

In Fig. 117, node  $\alpha$  of a search tree has three children  $\alpha_1$ ,  $\alpha_2$ , and  $\alpha_3$ ;  $\alpha_1$  is a “leaf” (either a solution or a contradiction), while  $\alpha_2$  has a child  $\alpha_{21}$  and  $\alpha_3$  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  $\text{STAMP}(x) = x'$  and  $\text{STAMP}(y) = y'$  to avoid placing some redundant entries on the trail; the current stamp  $\sigma$  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  $\{0, 1, \dots, d-1\}$ . 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  $0 \leq a < d$ ;
- 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  $d \leq 64$ , it’s attractive to work with the binary number

$$\text{BITS}(v) = \sum \{2^a \mid a \in D_v\}, \quad (95)$$

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

$$a \in D_v \iff \text{BITS}(v) \& (1 \ll a) \neq 0; \quad (96)$$

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

$$|D_v| \leq 1 \iff \text{BITS}(v) \& (\text{BITS}(v) - 1) = 0. \quad (98)$$

To compute  $|D_v| = \nu \text{BITS}(v)$ , 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:

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

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

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

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  $0 \leq a \leq d$  contains two tetrabytes called  $\text{PREV}_v(a)$  and  $\text{NEXT}_v(a)$  that link to the neighbors of  $a$  in  $v$ ’s list. For simplicity we’ll write  $\text{PREV}$  and  $\text{NEXT}$  instead of  $\text{PREV}_v$  and  $\text{NEXT}_v$ .

stamping  
Bit vectors  
bitwise operations  
MMIX  
SADD  
sideways addition  
ruler function  $\rho$   
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  $\{a_1, \dots, a_s\}$ , where  $0 \leq a_1 < a_2 < \dots < a_s < d$  and  $s > 0$ , we'll have

$$\text{NEXT}(d) = a_1, \quad \text{NEXT}(a_j) = a_{j+1} \text{ for } 1 \leq j < s, \quad \text{NEXT}(a_s) = d; \quad (101)$$

$$\text{PREV}(d) = a_s, \quad \text{PREV}(a_j) = a_{j-1} \text{ for } 1 < j \leq s, \quad \text{PREV}(a_1) = d; \quad (102)$$

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

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

Let's also add a Boolean array

$$\text{IN}_v[a] = [a \in D_v], \quad \text{for } 0 \leq a < d; \quad \text{IN}_v[d] = 0. \quad (104)$$

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

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:

$$\text{NEXT}(\text{PREV}(a)) \leftarrow a \quad \text{and} \quad \text{PREV}(\text{NEXT}(a)) \leftarrow a. \quad (105)$$

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  $\text{DOM}_v[k]$  and  $\text{IDOM}_v[k]$  for  $0 \leq k < d$ , together with another variable  $\text{SIZE}(v)$ . Both  $\text{DOM}_v$  and  $\text{IDOM}_v$  are permutations of  $\{0, 1, \dots, d-1\}$ ; and  $\text{IDOM}_v$  is the inverse of  $\text{DOM}_v$ :

$$\text{DOM}_v[\text{IDOM}_v[a]] = \text{IDOM}_v[\text{DOM}_v[a]] = a, \quad \text{for } 0 \leq a < d. \quad (106)$$

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

$$D_v = \{\text{DOM}_v[k] \mid 0 \leq k < \text{SIZE}(v)\}, \quad (107)$$

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

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

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

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

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

Set  $k \leftarrow \text{IDOM}_v[a]$ .

If  $k < \text{SIZE}(v)$ , set  $\text{SIZE}(v) \leftarrow \text{SIZE}(v) - 1$ ,  $a' \leftarrow \text{DOM}_v[\text{SIZE}(v)]$ , (110)

$\text{DOM}_v[\text{SIZE}(v)] \leftarrow a$ ,  $\text{IDOM}_v[a] \leftarrow \text{SIZE}(v)$ ,  $\text{DOM}_v[k] \leftarrow a'$ ,  $\text{IDOM}_v[a'] \leftarrow k$ .

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

In fact, a whole round of deletions can be undone by just restoring the previous value of  $\text{SIZE}(v)$ ; only  $\text{SIZE}(v)$  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  $\text{DOM}_v[d-1], \text{DOM}_v[d-2], \dots, \text{DOM}_v[\text{SIZE}(v)]$  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  $\text{SIZE}(v), \text{SIZE}(v)+1, \dots$  of  $\text{DOM}_v$ .

Another advantage is the fact that the sequentially accessed array  $\text{DOM}_v$  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  $0 \leq k < q$ . 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,

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

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,  $\text{D}[j]$  for  $0 \leq j < q$ , and an integer  $\text{S}$ , with

$$b_{\text{D}[j]} \neq 0 \iff j < \text{S}, \quad \text{for } 0 \leq j < q. \quad (112)$$

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

	$b_0$	$b_1$	$b_2$	$b_3$	$b_4$	$b_5$	$b_6$	$\text{D}$	$\text{S}$
Initial set $\{0, 1, \dots, 19\}$ :	111	111	111	111	111	111	110	0123456	7
AND with 011 001 001 000 011 111 10:	011	001	001	000	011	111	100	012645 3	6
AND with 010 001 000 010 110 100 01:	010	001	000	000	010	100	000	0154 623	4
AND with 101 001 100 010 011 000 11:	000	001	000	000	010	000	000	41 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  $\text{S}$  elements of  $\text{D}$  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  $O(\text{S})$  steps, while changing  $\text{D}$  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  $\text{S}$  before those changes were made.

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

**\*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  $\{p, q, r\}$ , two secondary items  $\{x, y\}$ , and five options:

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

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,  $\text{ITEM}[1] = 11$  is the internal number for ‘q’.

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

Table 3

THE INITIAL CONTENTS OF MEMORY CORRESPONDING TO (113)

$i$ SET[ $i$ ]			$i$ SET[ $i$ ]			$k:$ 0    1    2    3    4							
LNAME	0	p	• 17	7		ITEM[ $k$ ]:	4	11	17	23	31		
RNAME	1		18	17									
POS	2	0	LNAME	19	x	$x:$	0	1	2	3	4	5	6
SIZE	3	3	RNAME	20		ITM( $x$ ):	0	4	11	23	31	−4	4
• 4	1		POS	21	3	LOC( $x$ ):	4	4	11	23	31	4	5
5	6		SIZE	22	4	CLR( $x$ ):	—	0	0	0	A	—	0
6	11		• 23	3									
LNAME	7	q	24	8		$x:$	7	8	9	10	11	12	13
RNAME	8		25	12		ITM( $x$ ):	17	23	31	−4	4	23	−2
POS	9	1	26	15		LOC( $x$ ):	17	24	32	2	6	25	2
SIZE	10	2	LNAME	27	y	CLR( $x$ ):	0	A	0	—	0	B	—
• 11	2		RNAME	28									
12	14		POS	29	4	$x:$	14	15	16	17	18	19	
LNAME	13	r	SIZE	30	3	ITM( $x$ ):	11	23	−2	17	31	−2	
RNAME	14		• 31	4		LOC( $x$ ):	12	26	2	18	33	—	
POS	15	2	32	9		CLR( $x$ ):	0	A	—	0	B	—	
SIZE	16	2	33	18									

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  $\text{CLR}(x)$ —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):

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

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

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

$$D_i = \{\text{SET}[i + j] \mid 0 \leq j < \text{SIZE}(i)\}. \quad (116)$$

(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.

$\text{ITEM}[k]$  is a primary item if and only if  $\text{ITEM}[k] < \text{SECOND}$ . It is currently active if and only if  $k < \text{ACTIVE}$ , where **ACTIVE** is initially the total number of items. (Thus, **ACTIVE** = 5 and **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  $\text{NODE}[x]$  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  $x$ :

$$\begin{aligned} &\text{Set } x' \leftarrow x + 1, \text{ and repeat the following while } x' \neq x: \\ &\quad \text{If } \text{ITM}(x') > 0, \text{ visit } x' \text{ and set } x' \leftarrow x' + 1; \\ &\quad \text{otherwise set } x' \leftarrow x' + \text{ITM}(x'). \end{aligned} \quad (117)$$

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  $c \neq 0$ . 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  $c = \text{A}$ , 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( $i, c$ )

If some primary item is about to lose its last remaining option, the hide routine stops what it was doing and sets  $\text{FLAG} \leftarrow 1$ , where  $\text{FLAG}$  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  $\text{LEN}(x) = 0$  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  $\text{ACTIVE}$  and  $\text{OACTIVE}$ , where  $\text{OACTIVE}$  is the value that  $\text{ACTIVE}$  had just before items of the current option were being deactivated. It uses local variables  $j$ ,  $x$ ,  $x'$ ,  $x''$ ,  $i'$ ,  $i''$ , and  $s'$ :

$$\text{hide}(i, c) = \left\{ \begin{array}{l} \text{For } 0 \leq j < \text{SIZE}(i), \text{ set } x \leftarrow \text{SET}[i + j] \text{ and} \\ \quad \text{do the following if } c = 0 \text{ or } c \neq \text{CLR}(x): \\ \quad \text{For all siblings } x' \text{ of } x, \text{ set } i' \leftarrow \text{ITM}(x') \text{ and} \\ \quad \quad \text{do the following if } \text{POS}(i') < \text{OACTIVE}: \\ \quad \quad \text{Set } s' \leftarrow \text{SIZE}(i') - 1. \\ \quad \quad \text{If } s' = 0 \text{ and } \text{FLAG} = 0 \text{ and } i' < \text{SECOND} \text{ and} \\ \quad \quad \quad \text{POS}(i') < \text{ACTIVE}, \text{ set } \text{FLAG} \leftarrow 1 \text{ and return.} \\ \quad \quad \text{Otherwise set } x'' \leftarrow \text{SET}[i' + s'], \text{ SIZE}(i') \leftarrow s', \\ \quad \quad \quad \text{SET}[i' + s'] \leftarrow x', i'' \leftarrow \text{LOC}(x'), \text{ LOC}(x') \leftarrow i' + s', \\ \quad \quad \quad \text{SET}[i''] \leftarrow x'', \text{ LOC}(x'') \leftarrow i''. \end{array} \right. \quad (118)$$

**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  $x_0x_1 \dots x_T$  and  $y_0y_1 \dots y_T$  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  $\text{TRAIL}[0], \text{TRAIL}[1], \dots$ , 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  $\text{ACTIVE}$  to the number of items,  $\text{SECOND}$  to the internal number of the smallest secondary item (or  $\infty$  if there are none), and  $l \leftarrow y_0 \leftarrow t \leftarrow 0$ .
- C2.** [Choose  $i$ .] Set  $i \leftarrow \text{ITEM}[k]$  for some  $k$  with  $0 \leq k < \text{ACTIVE}$  and  $\text{ITEM}[k] < \text{SECOND}$  and minimum  $\text{SIZE}(i)$ . 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  $k' \leftarrow \text{ACTIVE} - 1$ ,  $\text{ACTIVE} \leftarrow k'$ ,  $i' \leftarrow \text{ITEM}[k']$ ,  $k \leftarrow \text{POS}(i)$ ,  $\text{ITEM}[k'] \leftarrow i$ ,  $\text{ITEM}[k] \leftarrow i'$ ,  $\text{POS}(i') \leftarrow k$ ,  $\text{POS}(i) \leftarrow k'$ .
- C4.** [Hide  $i$ .] Set  $\text{OACTIVE} \leftarrow \text{ACTIVE}$ ,  $\text{FLAG} \leftarrow -1$ ;  $\text{hide}(i, 0)$  and set  $j \leftarrow i$ .
- C5.** [Trail the sizes.] Terminate with trail overflow if  $t + \text{ACTIVE}$  exceeds the maximum available  $\text{TRAIL}$  size. Otherwise set  $\text{TRAIL}[t + k] \leftarrow (\text{ITEM}[k], \text{SIZE}(\text{ITEM}[k]))$  for  $0 \leq k < \text{ACTIVE}$ ; then set  $y_{l+1} \leftarrow t \leftarrow t + \text{ACTIVE}$ .
- C6.** [Try  $\text{SET}[j]$ .] Set  $x_l \leftarrow \text{SET}[j]$  and  $k \leftarrow \text{OACTIVE} \leftarrow \text{ACTIVE}$ . For all siblings  $x'$  of  $x_l$ , set  $i' \leftarrow \text{ITM}(x')$ ,  $k' \leftarrow \text{POS}(i')$ , and if  $k' < k$  set  $k \leftarrow k - 1$ ,  $i'' \leftarrow \text{ITEM}[k]$ ,  $\text{ITEM}[k] \leftarrow i'$ ,  $\text{ITEM}[k'] \leftarrow i''$ ,  $\text{POS}(i'') \leftarrow k'$ ,  $\text{POS}(i') \leftarrow k$ . Then set  $\text{ACTIVE} \leftarrow k$ . (We’ve deactivated the other items of option  $x_l$ .)

**FLAG**

global variables

**OACTIVE**

secondary item

MRV: Minimum remaining values

- C7.** [Hide SET[ $j$ ].] Set  $\text{FLAG} \leftarrow 0$ . For all siblings  $x'$  of  $x_l$ , set  $i' \leftarrow \text{ITM}(x')$ ; and if  $i' < \text{SECOND}$  or  $\text{POS}(i') < \text{OACTIVE}$ , hide( $i'$ ,  $\text{CLR}(x')$ ), and go to C11 if  $\text{FLAG} = 1$ . (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  $x_0 x_1 \dots x_{l-1}$ .
- C10.** [Leave level  $l$ .] Terminate if  $l = 0$ . Otherwise set  $l \leftarrow l - 1$ ,  $i \leftarrow \text{ITM}(x_l)$ ,  $j \leftarrow \text{LOC}(x_l)$ .
- C11.** [Try again?] If  $j+1 \geq i + \text{SIZE}(i)$ , go to C10. Otherwise, for  $y_l \leq k < y_{l+1}$ , set  $\text{SIZE}(i') \leftarrow s'$  if  $\text{TRAIL}[k] = (i', s')$ . Then set  $t \leftarrow y_{l+1}$ ,  $\text{ACTIVE} \leftarrow t - y_l$ ,  $j \leftarrow j + 1$ , 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  $G\mu$  (141 mems/sol, 14  $\mu/v$ ), beating the old 19.4  $G\mu$  (179 mems/sol, 17.8  $\mu/v$ ).

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 $G\mu$	51.9 $G\mu$	0.813	
D	(2327, 77+1, 16146)	12.5 $G\mu$	20.3 $G\mu$	0.614	
H	(1416, 196+93, 5224)	623.4 $G\mu$	613.4 $G\mu$	1.016	
K	(343, 49+288, 110968)	9.6 $G\mu$	3.1 $G\mu$	3.089	
L	(352, 48+0, 326721800)	881.2 $G\mu$	1123.3 $G\mu$	0.784	
M	(1514, 49+42, 987816)	21.6 $G\mu$	25.3 $G\mu$	0.854	
O*	(6966, 180+0, 16928)	7105.4 $G\mu$	12732.0 $G\mu$	0.558	(119)
Q	(256, 32+58, 14772512)	58.9 $G\mu$	40.2 $G\mu$	1.467	
R	(121, 11+741, 401800)	4.8 $G\mu$	0.8 $G\mu$	5.816	
S	(3858, 342+90, 30258432)	211.4 $G\mu$	170.5 $G\mu$	1.240	
U	(2440, 72+0, 31520)	119.7 $G\mu$	194.5 $G\mu$	0.615	
W	(1212, 12+36, 352)	7.4 $G\mu$	10.5 $G\mu$	0.702	
Y*	(949, 205+276, 16)	26.3 $G\mu$	23.2 $G\mu$	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  $5 \times 7$  subcase of Grabarchuk's double-snake puzzle for windmill dominoes (exercise 7.2.2.1–306). Problem K colors the  $7 \times 7$  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  $3 \times 4 \times 5$  box (exercise 7.2.2.1–340(b)). Problem W makes  $6 \times 6$  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<sup>+</sup>. Algorithm C<sup>+</sup> 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	(120)
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	

**\*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 C2<sup>+</sup> of Algorithm C<sup>+</sup>, 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  $\text{SIZE}(i)$  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  $\text{SIZE}(i) = 1$ . Step C3 of Algorithm 7.2.2.1C can terminate early if it finds  $\text{SIZE}(i) = 0$ .

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

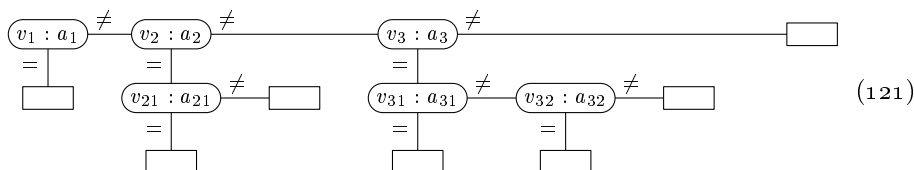
In fact, the amount of time devoted to the MRV heuristic in (120) is comparatively small; it's at most 8%, except in Problem R. In Problem L, for example, only about  $33\text{ G}\mu$  of the  $814.7\text{ G}\mu$  total running time is spent in step C2. In Problems C, O\*, U, and W the MRV time is completely 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:



(This tree has basically the same structure as the computation in Fig. 117; node  $\alpha$  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<sup>+</sup>, 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  $h(i) = \text{SIZE}(i)$  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,  $\text{WT}(i)$ , for each primary item  $i$ . We start with  $\text{WT}(i) \leftarrow 1$ ; then we set  $\text{WT}(i) \leftarrow \text{WT}(i) + 1$  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:

$$h(i) = \text{SIZE}(i) / \text{WT}(i). \quad (122)$$

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<sup>+</sup> needs 7.9 *gigamems*.

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  $\text{FR}(i)$  and  $\text{TRY}(i)$  for each primary item  $i$ , where  $\text{FR}(i)$  is initially 0.5 and  $\text{TRY}(i)$  is initially 1. After trying to cover item  $i$  with one of its options  $o$ , we set  $\text{TRY}(i) \leftarrow \text{TRY}(i) + 1$  and

$$\text{FR}(i) = \begin{cases} \text{FR}(i) - \text{FR}(i)/\text{TRY}(i), & \text{if nonfailure;} \\ \text{FR}(i) + (1.0 - \text{FR}(i))/\text{TRY}(i), & \text{if failure;} \end{cases} \quad (123)$$

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

$$h(i) = \text{SIZE}(i)/\text{FR}(i). \quad (124)$$

(Like MRV, it tends to help us "fail early" in a search, rather than later.)

By definition, we have  $\text{TRY}(i) = 2$  just after the first time we try to branch on  $i$ ; and the failure rate  $\text{FR}(i)$  is either 0.25 or 0.75. Later, after the second try, we'll have  $\text{TRY}(i) = 3$  and  $\text{FR}(i) \in \{0.1666\dots, 0.5, 0.8333\dots\}$ . In general, after the  $m$ th attempt to branch on  $i$ ,  $\text{TRY}(i)$  will be  $m + 1$  and  $\text{FR}(i)$  will be in the set  $\{\frac{1}{2m+2}, \frac{3}{2m+2}, \dots, \frac{2m+1}{2m+2}\}$ , 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<sup>+</sup>. (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 G $\mu$ ; that’s much better than the 6.6 G $\mu$  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 G $\mu$	54.3 G $\mu$	45.6 G $\mu$	MRV
D	(2327, 77+1, 16146)	12.4 G $\mu$	21.3 G $\mu$	12.8 G $\mu$	MRV
H	(1416, 196+93, 5224)	407.4 G $\mu$	19.6 G $\mu$	34.8 G $\mu$	WTD
J	(264, 144+0, 1)	50.3 G $\mu$	0.8 M $\mu$	1.9 M $\mu$	WTD
K	(343, 49+288, 110968)	4.1 G $\mu$	2.7 G $\mu$	2.8 G $\mu$	WTD
L	(352, 48+0, 326721800)	814.7 G $\mu$	1080.5 G $\mu$	1126.1 G $\mu$	MRV
M	(1514, 49+42, 987816)	20.6 G $\mu$	19.1 G $\mu$	8.1 G $\mu$	FRB
O*	(6966, 180+0, 16928)	7090.2 G $\mu$	8363.8 G $\mu$	6104.5 G $\mu$	FRB (125)
Q	(256, 32+58, 14772512)	43.9 G $\mu$	66.6 G $\mu$	65.6 G $\mu$	MRV
R	(121, 11+741, 401800)	2.9 G $\mu$	3.3 G $\mu$	3.5 G $\mu$	MRV
S	(3858, 342+90, 30258432)	125.9 G $\mu$	149.9 G $\mu$	147.5 G $\mu$	MRV
U	(2440, 72+0, 31520)	119.1 G $\mu$	189.1 G $\mu$	124.8 G $\mu$	MRV
W	(1212, 12+36, 352)	7.4 G $\mu$	10.3 G $\mu$	9.2 G $\mu$	MRV
Y*	(949, 205+276, 16)	23.6 G $\mu$	6.6 G $\mu$	2.6 G $\mu$	FRB

benchmark problems  
dancing links  
focus  
forward consistency  
incompatible  
domain consistency  
compatible

Here MRV means Algorithm C<sup>+</sup>; 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–(g2), for which MRV has “bad focus.” (The corresponding runtime for dancing links is 27.4 G $\mu$ .) 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 T $\mu$  to 7.3 T $\mu$ .

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 (g2) 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.

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  $o'$ .

support matrix  
domain  
purging

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’	(126)
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’	#	

(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,  $S[05, q] = 13$ ; and 13 is indeed a support for (05, q) because  $05 \parallel 13$  and  $q \in 13$ .

Unfortunately, (126) contains an unavoidable “hole” in position  $S[10, q]$ . 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  $S[19, p]$  has to be changed. No problem: We can set  $S[19, p] \leftarrow 00$ .

But  $S[16, p]$  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’	(127)
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’	#	

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  $(05, q) (19, q)$ ; 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  $(13, q)$  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  $(13, p) (13, r)$  won't trigger any new events, because option 13 is no longer active.) To fill the hole in 19, we implicitly set  $S[19, q] \leftarrow 00$ , by pushing  $(19, q)$  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  $c_1, \dots, c_s$  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  $c_1 \cup \dots \cup c_s$ . (And that's the same as the set  $\{\text{ITEM}[k] \mid 0 \leq k < \text{ACTIVE}\}$ .)

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  $O_0^{\text{init}}$ , which is the largest subset of  $O_{-1}$  that is domain consistent, and it will enter stage 0. And  $O_r^{\text{init}}$ , 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  $O_r^{\text{init}}$  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  $s$ :

$$O_{-1} \supseteq O_0^{\text{init}} \supseteq O_0 \supset O_1^{\text{init}} \supseteq O_1 \supset \dots \supset O_s^{\text{init}} \supseteq O_s. \quad (128)$$

Here every set  $O_r^{\text{init}}$  and  $O_r$  is domain consistent, for  $0 \leq r \leq s$ .

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  $O_{-1} \setminus O_0^{\text{init}}$ , which were purged at

trigger stack  
fixit stack  
queue  
active items and options  
domain consistent  
chosen  
blocked  
removed  
purged  
Forward consistency  
age

the beginning before entering stage 0, have age  $-1$ . Consequently

$$\begin{aligned} O_{r-1} &= \{o \mid \text{AGE}(o) \geq 2r - 1\}, \\ O_r^{\text{init}} &= \{o \mid \text{AGE}(o) \geq 2r\}, \end{aligned} \quad \text{for } 0 \leq r \leq s, \quad (129)$$

if we regard  $\text{AGE}(o)$  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  $x_0x_1 \dots x_T$ ,  $y_0y_1 \dots y_{T_0}$ ,  $d_0d_1 \dots d_T$ , **TRAIL**, and **LS** as in Algorithm B, as well as linked lists for the special data structures described in exercise 339. Also  $\text{SS}[s]$  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  $\text{TRIG}(o) \leftarrow \text{FIX}(o) \leftarrow 0$  for every option  $o$ ; also  $\text{STAMP} \leftarrow \text{SSTAMP} \leftarrow 0$ ,  $s \leftarrow l \leftarrow -1$ . 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  $\text{SS}[s] \leftarrow \text{SSTAMP}$ .
- S3.** [Enter new level.] Set  $l \leftarrow l + 1$  and  $\text{LS}[s] \leftarrow l$ . Terminate with level overflow if  $l > T$  (there's no room to store  $x_l$ ).

sort  
undoing  
 $T_0$   
first item of every option  
stamping  
domain consistency

- S4.** [Choose  $i$ .] Set  $i \leftarrow \text{ITEM}[k]$  for some  $k$  with  $0 \leq k < \text{ACTIVE}$  and  $\text{ITEM}[k] < \text{SECOND}$ . But if no such  $k$  exists, go to S8. The chosen  $i$  need not minimize  $\text{SIZE}(i)$ ; however, if  $\text{SIZE}(i) > 1$ , there must be no forced move, that is, no active primary item with  $\text{SIZE}(i) = 1$ .
- S5.** [Trail the sizes.] Set  $y_s \leftarrow t$  and  $d_l \leftarrow \text{SIZE}(i)$ . If  $d_l = 1$ , go to S6. Otherwise terminate with trail overflow if  $t + \text{ACTIVE}$  exceeds the maximum available **TRAIL** size. Otherwise set  $\text{TRAIL}[t+k] \leftarrow (\text{ITEM}[k], \text{SIZE}(\text{ITEM}[k]))$  for  $0 \leq k < \text{ACTIVE}$ ; then set  $t \leftarrow t + \text{ACTIVE}$ .
- S6.** [Try **SET**[ $i$ ].] Set  $x_l \leftarrow \text{SET}[i]$  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  $x_{\text{LS}[j]}$  for  $0 \leq j < s$ .
- S9.** [Back up.] Terminate if  $s = 0$ . Otherwise set  $s \leftarrow s - 1$ ,  $l \leftarrow \text{LS}[s]$ , and repeat step S9 if  $d_l = 1$ .
- S10.** [Untrail the sizes.] For  $y_s \leq k < t$ , set  $\text{SIZE}(i') \leftarrow s'$  if  $\text{TRAIL}[k] = (i', s')$ . Then set  $\text{ACTIVE} \leftarrow t - y_s$ ,  $t \leftarrow y_s$ .
- S11.** [Remove  $x_l$ .] Set  $A \leftarrow 2s$  and  $\text{OACTIVE} \leftarrow \text{ACTIVE}$ . 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. ■

MRV  
forced move  
stamping  
 $G\mu$   
gigamems  
 $M\nu$   
meganodes  
nodes  
heuristic functions  
forward consistency

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 $G\mu$ , 5.4 $M\nu$	109.2 $G\mu$ , 6.7 $M\nu$	126.6 $G\mu$ , 7.1 $M\nu$	130.2 $G\mu$ , 8.2 $M\nu$	
D	MRV, 12.4 $G\mu$ , 1.6 $M\nu$	19.9 $G\mu$ , 0.2 $M\nu$	23.5 $G\mu$ , 0.3 $M\nu$	27.0 $G\mu$ , 0.3 $M\nu$	
H	WTD, 19.6 $G\mu$ , 32.5 $M\nu$	137.4 $G\mu$ , 4.3 $M\nu$	312.6 $G\mu$ , 7.1 $M\nu$	397.4 $G\mu$ , 7.3 $M\nu$	
K	MRV <sup>†</sup> , 3.1 $G\mu$ , 10.6 $M\nu$	28.1 $G\mu$ , 3.1 $M\nu$	20.7 $G\mu$ , 2.2 $M\nu$	96.8 $G\mu$ , 6.6 $M\nu$	
L	MRV, 814.7 $G\mu$ , 4.0 $G\nu$	7.0 $T\mu$ , 2.8 $G\nu$	8.0 $T\mu$ , 3.0 $G\nu$	12.5 $T\mu$ , 3.7 $G\nu$	
M	FRB, 8.1 $G\mu$ , 15.7 $M\nu$	73.5 $G\mu$ , 10.4 $M\nu$	67.5 $G\mu$ , 10.3 $M\nu$	333.6 $G\mu$ , 19.3 $M\nu$	
O	FRB*, 6.1 $T\mu$ , 461.5 $M\nu$	8.2 $T\mu$ , 7.1 $M\nu$	8.5 $T\mu$ , 9.6 $M\nu$	10.5 $T\mu$ , 10.1 $M\nu$	(130)
Q	MRV <sup>†</sup> , 40.2 $G\mu$ , 193.0 $M\nu$	208.5 $G\mu$ , 121.0 $M\nu$	307.7 $G\mu$ , 137.8 $M\nu$	384.7 $G\mu$ , 158.7 $M\nu$	
R	MRV, 2.9 $G\mu$ , 1.6 $M\nu$	4.4 $G\mu$ , 1.9 $M\nu$	4.6 $G\mu$ , 2.0 $M\nu$	4.9 $G\mu$ , 2.0 $M\nu$	
S	MRV, 125.9 $G\mu$ , 548.0 $M\nu$	2.7 $T\mu$ , 568.5 $M\nu$	7.5 $T\mu$ , 691.7 $M\nu$	3.2 $T\mu$ , 667.8 $M\nu$	
U	MRV, 119.1 $G\mu$ , 17.1 $M\nu$	210.6 $G\mu$ , 1.7 $M\nu$	255.9 $G\mu$ , 2.2 $M\nu$	393.4 $G\mu$ , 2.8 $M\nu$	
W	MRV, 7.4 $G\mu$ , 1.7 $M\nu$	12.0 $G\mu$ , 0.8 $M\nu$	18.5 $G\mu$ , 1.2 $M\nu$	27.0 $G\mu$ , 1.7 $M\nu$	
Y	FRB*, 2.6 $G\mu$ , 1.3 $M\nu$	7.7 $G\mu$ , 54.9 $K\nu$	4.5 $G\mu$ , 40.1 $K\nu$	5.8 $G\mu$ , 42.0 $K\nu$	

Here ‘ $G\mu$ ’ stands for gigamems, as usual, while ‘ $M\nu$ ’ 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<sup>+</sup> 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<sup>†</sup> refers to dancing links, Algorithm 7.2.2.1C, while MRV refers to dancing cells, Algorithm C<sup>+</sup>; 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  $\mu/\nu$  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 <sup>†</sup> *, 59.1 G $\mu$	54.5 G $\mu$	13.0 G $\mu$	22.6 G $\mu$	
E	(2536, 54+14, 89328)	FRB*, 33.2 G $\mu$	28.1 G $\mu$	55.6 G $\mu$	62.3 G $\mu$	
F	(7800, 81+594, 1)	WTD*, 10.5 G $\mu$	158.3 M $\mu$	139.1 M $\mu$	149.5 M $\mu$	
G	(576, 48+506, 8388608)	FRB, 41.7 G $\mu$	96.3 G $\mu$	87.6 G $\mu$	70.3 G $\mu$	
I	(20088, 81+72, 16)	MRV*, 28.9 G $\mu$	999.5 M $\mu$	1.1 G $\mu$	1.0 G $\mu$	(131)
N	(5546, 17+668, 43)	FRB*, 77.4 G $\mu$	30.5 G $\mu$	13.5 G $\mu$	32.7 G $\mu$	
P	(14179, 200+100, 3)	FRB*, 1.4 T $\mu$	3.0 T $\mu$	531.7 G $\mu$	4.3 T $\mu$	
T	(2658, 29+338, 416)	FRB <sup>†</sup> , 4.9 T $\mu$	12.4 T $\mu$	5.8 T $\mu$	4.1 T $\mu$	
V	(22000, 9+20, 32620)	FRB <sup>†</sup> *, 112.9 G $\mu$	65.7 G $\mu$	73.8 G $\mu$	81.0 G $\mu$	
Z	(1104, 24+24, 575)	MRV, 203.7 G $\mu$	29.4 M $\mu$	29.9 M $\mu$	30.1 M $\mu$	

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  $\pi$ ” 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  $9 \times 9$  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  $10 \times 10$  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  $4 \times 5$  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  
 $\pi$   
 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



Twelve FC experiments lie behind each row of (131), namely the application of algorithms that we may call  $\text{MRV}^\dagger$ ,  $\text{MRV}$ ,  $\text{WTD}$ ,  $\text{FRB}$ ,  $\text{WTD}^\dagger$ ,  $\text{FRB}^\dagger$ ;  $\text{MRV}^{\dagger*}$ ,  $\text{MRV}^*$ ,  $\text{WTD}^*$ ,  $\text{FRB}^*$ ,  $\text{WTD}^{\dagger*}$ ,  $\text{FRB}^{\dagger*}$ . The dagger after  $\text{MRV}$  indicates dancing links, and the dagger after  $\text{WTD}$  or  $\text{FRB}$  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

$$(202.1, 168.8, 98.9, 1653.7, 77.6, 94.6; 202.0+10.2, 168.8+10.2, \\ 94.5+10.2, 1729.3+10.2, 77.6+10.2, 48.9+10.2) \text{ G}\mu,$$

where  $10.2 \text{ G}\mu$  was the preprocessing time. In this problem,  $\text{FRB}^{\dagger*}$  was a clear winner and  $\text{FRB}^*$  was a clear loser;  $\text{WTD}^\dagger$  was a close second.

The biggest surprise in (131) was the result of Problem G, whose six scores

$$(2999.1, 5405.7, 918.2, 41.7, 1129.1, 539.9) \text{ G}\mu$$

testified to a tremendous victory for the  $\text{FRB}$  heuristic, placing it ahead of all three variants of Algorithm S. Previous experiences with  $\text{MRV}$  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  $\text{FRB}^*$  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  $n \leq 8$  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  $k$ th move with  $k$  for  $1 \leq k \leq n^2$ , 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 <b>13</b> 06 09 50 <b>97</b> 00	<b>47</b> 10 49 78 <b>67</b> 08 65 76 <b>73</b> 70	<b>59</b> 64 57 68 <b>71</b> 62 91 08 <b>73</b> 96
34 <b>31</b> 14 69 10 49 98 <b>05</b> 08 51	50 <b>79</b> 46 09 04 77 68 <b>71</b> 64 75	56 <b>67</b> 60 63 06 69 72 <b>97</b> 92 09
27 16 <b>29</b> 32 <b>71</b> 12 <b>07</b> 52 99 96	45 48 <b>11</b> 80 <b>07</b> 66 <b>03</b> 74 69 72	65 58 <b>03</b> 70 <b>61</b> 90 <b>07</b> 10 95 74
38 35 70 <b>11</b> 68 <b>41</b> 48 95 04 93	54 51 44 <b>13</b> 42 <b>05</b> 84 99 40 63	36 55 66 <b>89</b> <b>02</b> <b>05</b> 44 75 98 93
<b>17</b> 26 <b>37</b> 40 45 72 <b>03</b> 92 <b>53</b> 56	<b>19</b> 12 <b>53</b> 06 81 60 <b>41</b> <b>02</b> <b>97</b> 00	<b>19</b> 22 <b>37</b> 04 45 14 <b>11</b> 94 <b>43</b> 76
36 39 44 <b>67</b> <b>02</b> <b>47</b> 42 55 94 91	52 55 20 <b>43</b> 14 <b>83</b> 98 85 62 39	54 35 20 <b>17</b> 88 01 46 15 12 99
25 18 <b>23</b> 46 <b>43</b> 66 <b>73</b> 82 57 54	21 18 <b>23</b> 82 <b>59</b> 32 <b>61</b> 94 01 96	21 18 <b>23</b> 38 <b>47</b> 16 <b>13</b> 00 77 42
22 <b>61</b> 20 01 74 85 58 <b>79</b> 90 81	56 <b>89</b> 58 15 24 93 86 <b>29</b> 38 35	34 <b>53</b> 32 87 50 39 28 <b>79</b> 84 81
<b>19</b> 24 63 86 <b>59</b> 76 65 88 <b>83</b> 78	<b>17</b> 22 91 88 <b>31</b> 26 33 36 95 28	<b>31</b> 24 51 48 <b>29</b> 26 85 82 <b>41</b> 78
62 21 60 75 64 87 84 77 80 <b>89</b>	90 57 16 25 92 87 30 27 34 <b>37</b>	52 33 30 25 86 49 40 27 80 <b>83</b>

in which a queen near the center attacks all 25 of the primes  $\leq 100$ . (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

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

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  $\dots$  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  $\pi$  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 <b>71</b> 06 75 94 <b>73</b> 82	11 34 99 96 <b>71</b> 06 85 94 <b>73</b> 78	11 34 99 96 <b>79</b> 72 77 94 <b>67</b> 70
98 <b>37</b> 10 33 <b>00</b> 95 72 <b>83</b> 76 93	98 <b>37</b> 10 33 <b>00</b> 95 72 <b>79</b> 86 93	98 <b>37</b> 10 33 <b>00</b> 95 68 <b>71</b> 74 93
35 12 <b>97</b> 70 <b>07</b> 18 <b>05</b> 74 81 84	35 12 <b>97</b> 70 <b>07</b> 18 <b>05</b> 84 77 74	35 12 <b>97</b> 80 <b>05</b> 78 <b>73</b> 76 69 66
38 09 36 <b>17 32 01</b> 80 85 92 77	38 09 36 <b>17 32 01</b> 80 75 92 87	38 09 36 <b>17 32 01</b> 06 65 92 75
<b>13 16 31 08 69 04 19 78 89 86</b>	<b>13 16 31 08 69 04 19 88 83 76</b>	<b>13 16 31 08 81 04 19 02 61 64</b>
30 39 14 <b>41 02 79</b> 58 87 20 91	30 39 14 <b>41 02 89</b> 58 81 20 91	30 39 14 <b>41 18 07</b> 60 63 20 91
15 42 <b>47</b> 68 <b>59</b> 66 <b>03</b> 90 57 88	15 42 <b>47</b> 68 <b>59</b> 66 <b>03</b> 90 57 82	15 42 <b>47</b> 82 <b>59</b> 84 <b>03</b> 90 57 62
48 <b>29</b> 40 45 <b>26</b> 63 60 <b>23</b> 54 21	48 <b>29</b> 40 45 <b>26</b> 63 60 <b>23</b> 54 21	48 <b>29</b> 40 45 <b>26</b> 87 58 <b>23</b> 54 21
<b>43</b> 46 27 50 <b>67</b> 24 65 52 <b>61</b> 56	<b>43</b> 46 27 50 <b>67</b> 24 65 52 <b>61</b> 56	<b>43</b> 46 27 50 <b>83</b> 24 85 52 <b>89</b> 56
28 49 44 25 <b>64</b> 51 62 55 22 <b>53</b>	28 49 44 25 <b>64</b> 51 62 55 22 <b>53</b>	28 49 44 25 86 51 88 55 22 <b>53</b>

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) 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*  $[u_i \dots v_i]$  of multiplicities, governing the number of times it must appear in a solution.

\* Indeed, the middle one, obtained after 6.4 T $\mu$  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–(6g), 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  $[u_i \dots v_i]$  is specified for every primary item  $i$ , where  $u_i \leq v_i$  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  $\text{SIZE}(i)$  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  $\text{SLACK}(i)$  and  $\text{BOUND}(i)$ , where

$$\text{SLACK}(i) = v_i - u_i \quad \text{and} \quad \text{BOUND}(i) = v_i. \quad (132)$$

The value of  $\text{SLACK}(i)$  remains unchanged throughout the computation; but  $\text{BOUND}(i)$  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  $v_i = \text{BOUND}(i)$  has become less than  $\text{SLACK}(i)$ . 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  $\text{SIZE}(i) = 0$  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  $\text{SIZE}(i)$  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-  
SLACK field+

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  $\text{SET}[i]$ , 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  $\text{SIZE}(i) = 5$  options  $\{o_1, o_2, o_3, o_4, o_5\}$ , and that  $\text{SLACK}(i) = 1$ ,  $\text{BOUND}(i) = 4$ ; 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

$$d_i = \text{SIZE}(i) + 1 - (\text{BOUND}(i) \dot{-} \text{SLACK}(i)) \quad (133)$$

and ' $\dot{-}$ ' is the “monus operation,”  $x \dot{-} y = \max(x - y, 0)$ . Algorithm F takes care to ensure not only that  $\text{SIZE}(i) > 0$ , 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 as small as possible.

Notice that a “forced move” arises when  $d_i = 1$ , namely when  $\text{SIZE}(i) = \text{BOUND}(i) - \text{SLACK}(i)$ , because  $\text{SIZE}(i) > 0$ . This means that all  $\text{SIZE}(i)$  of  $i$ 's current options must be included in  $S$ ; otherwise we wouldn't satisfy the lower bound  $u_i = \text{BOUND}(i) - \text{SLACK}(i)$ . (This analysis generalizes the forced-move condition of XCC problems, where  $\text{BOUND}(i) = 1$  and  $\text{SLACK}(i) = 0$ ; in Algorithms B and C, a move for  $i$  was forced if and only if  $\text{SIZE}(i) = 1$ .)

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	
$\mathcal{C}^\#$	(4068, 132+0, 5347)	4607.2 Gμ	4080.2 Gμ	6774.6 Gμ	2646.0 Gμ	
$\mathcal{D}^\#$	(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	
$\mathcal{H}^\#$	(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	
$\mathcal{N}^\#$	(256, 2700+58, 71486)	1786.8 Gμ	87.4 Gμ	134.5 Gμ	140.6 Gμ	(134)
$\mathcal{P}$	(2436, 1730+0, 112)	438.7 Gμ	354.4 Gμ	991.1 Gμ	379.9 Gμ	
$\mathcal{Q}^\#$	(3940, 65+126, 512)	284.0 Gμ	138.0 Gμ	138.7 Gμ	138.7 Gμ	
$\mathcal{R}^\#$	(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	
$\mathcal{X}^\#$	(576, 115+128, 4)	550.2 Gμ	361.3 Gμ	158.9 Gμ	411.0 Gμ	

Here Problem  $\mathcal{A}$  is the “authentic” partridge puzzle (exercise 7.2.2.1–155) with  $n = 6$ . Problem  $\mathcal{B}$  covers an  $8 \times 12$  grid with 10 two-dimensional balls of diameter 4 (see exercise 354). Problem  $\mathcal{C}$  covers the diagonals of a  $10 \times 10$  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  $7 \times 7$  board, allowing multiplicities  $[1..2]$  at the edges (see exercise 355). Problem  $\mathcal{H}$  packs

monus operation  
MRV  
forced move  
partridge puzzle  
balls  
disks  
pentominoes  
balanced  
5-queens  
domination  
piles

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  $6 \times 12$  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  $4 \times 5$  word rectangles with fewest distinct letters, 7.2.2.1–(66). Problem  $\mathcal{S}$  achieves central symmetry in the blank regions of a balanced  $10 \times 10$  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  $8 \times 8$  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, ‘ $\mathcal{C}^\neg$ ’ 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  $\{0, 1, \dots, p-1\}$ , 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.)

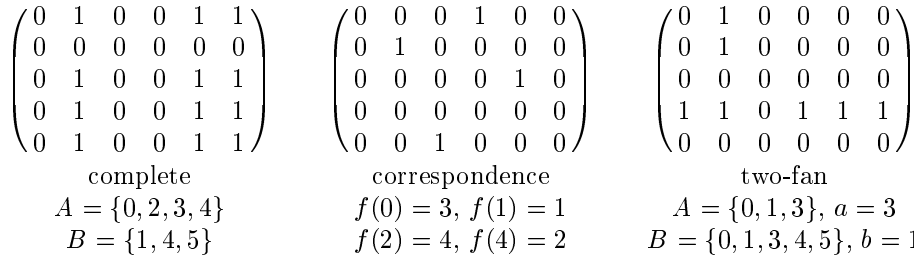
hypersolid pentominoes  
motley dissections  
16 queens problem  
no-three-in-a-line  
Perfect Packing puzzle  
word rectangles  
balanced  
pentomino  
Tullis  
tapestry  
Wainwright  
partridge puzzle  
crossword diagram  
sharp preference heuristic  
nonsharp preference heuristic  
tractable-  
treewidth  
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.

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  $A$ .



**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  $|A| \leq 1$  or  $|B| \leq 1$ , 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  $D_w \leftarrow$

implicational constraints–  
complete relation  
correspondence  
two-fan  
matrices of 0s and 1s  
matrix representation of binary relation  
0/1/all  
consistency  
unary constraint

$D_w \cap B_{vw}$ . 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  $(v, a) \rightarrow (w, b)$  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  $(v, a) \rightarrow \perp$  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

$$D_w = \{0, 2, 4\}, \quad D_x = \{0, 3\}, \quad D_y = \{1, 4\}, \quad D_z = \{1, 4, 5\}. \quad (135)$$

Therefore  $G$  has  $3 + 2 + 2 + 3 + 1$  vertices,  $\{(w, 0), (w, 2), \dots, (z, 5), \perp\}$ . The arcs for the correspondence  $C_{wy}$  (and for its converse,  $C_{yw}$ ) are

$$(w, 0) \rightarrow \perp, \quad (w, 2) \rightarrow (y, 4), \quad (w, 4) \rightarrow \perp; \quad (y, 1) \rightarrow \perp, \quad (y, 4) \rightarrow (w, 2); \quad (136)$$

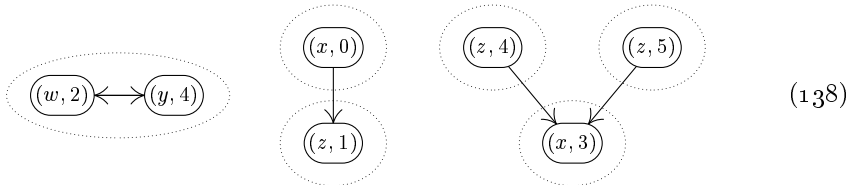
the arcs for the two fan  $C_{xz}$  (and for  $C_{zx}$ ) are

$$(x, 0) \rightarrow (z, 1); \quad (z, 4) \rightarrow (x, 3), \quad (z, 5) \rightarrow (x, 3). \quad (137)$$

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  $\hat{G}$  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  $\hat{G}$ .

In our example, with  $G$  defined by (136) and (137), the acyclic digraph  $\hat{G}$  is



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  $\hat{G}$  in reverse topological order as follows: "Choose any component  $\{(v_1, a_1), \dots, (v_t, a_t)\}$  that has no outgoing arcs in  $\hat{G}$ . If  $v_1$  has not already been assigned a value, set  $v_j \leftarrow a_j$  for  $1 \leq j \leq t$ . (Variables  $v_2, \dots, v_t$  won't have been set yet either; see exercise 370.) Delete this component, and repeat until  $\hat{G}$  is empty."

strong components  
Tarjan  
bad strong component  
topological order

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

$$(v_1, a_1) \longrightarrow (v_2, a_2) \longrightarrow \cdots \longrightarrow (v_{t-1}, a_{t-1}) \longrightarrow (v_t, a_t), \quad (139)$$

and if  $a'_t \neq a_t$  is another element of  $D_{v_t}$ , then  $G$  also contains a path

$$(v_t, a'_t) \longrightarrow (v_{t-1}, a'_{t-1}) \longrightarrow \cdots \longrightarrow (v_2, a'_2) \longrightarrow (v_1, a'_1), \quad (140)$$

for some values  $a'_1 \neq a_1$ ,  $a'_2 \neq a_2$ ,  $\dots$ ,  $a'_{t-1} \neq a_{t-1}$ .” 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  $O(\sum_v (1 + |D_v|) + \sum_{vw} (1 + |A_{vw}| + |B_{vw}|))$ , because (i) the domain reduction takes linear time using exercise 371; (ii) the construction of  $G$  and  $\hat{G}$  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  $(x_1, \dots, x_t)$  and  $(x'_1, \dots, x'_t)$  both satisfy the relation, so does  $(\max(x_1, x'_1), \dots, \max(x_t, x'_t))$ .”

For example, the relation ‘ $x < y$ ’ is obviously max-closed. A CSP in which all variables have domain  $\{0, 1, \dots, d-1\}$  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:

$$\begin{aligned} x &\text{ is prime;} \\ 3x + 5y + 8z &\geq 13w + 21; \\ xyz^2 + 3w &\geq t^3 - 7; \\ w \geq 2 \text{ or } x \geq 3 \text{ or } y \geq 4 \text{ or } z &\leq 5. \end{aligned}$$

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

$$f(x_1, \dots, x_t) \geq g(x) \quad (141)$$

is a max-closed relation on  $t + 1$  variables. For if, say,  $x \leq x'$ , we have

$$g(\max(x, x')) = g(x') \leq f(x'_1, \dots, x'_t) \leq f(\max(x_1, x'_1), \dots, \max(x_t, x'_t)).$$

Exercise 374 proves that, similarly, every relation of the form

$$f(x_1, \dots, x_t) \geq \alpha \quad \text{or} \quad g(x) \leq \beta, \quad (142)$$

where  $\alpha$  and  $\beta$  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:

$$\text{mc\_revise}(c, v) = \begin{cases} \text{Set } a \text{ to the largest element of } D_v. \\ \text{If } c \text{ contains no tuple having } v = a, \text{ and having} \\ \quad \text{all other variables in their current domains,} \\ \quad \text{set } D_v \leftarrow D_v \setminus a, \\ \quad \text{and repeat this process if } D_v \neq \emptyset. \end{cases} \quad (143)$$

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 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. (Coordinate-wise 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  $v_1, \dots, v_n$ , with domains of nonzero sizes  $d_1, \dots, d_n$ , together with  $\binom{n}{2}$  path consistent relations  $R_{ij}$  for  $1 \leq i < j \leq n$  defined by  $d_i \times d_j$  matrices that have no all-0 rows and are row convex, then it has a global solution  $a_1 \dots a_n$  that can be found in  $O(n^2)$  steps.*

*Proof.* In fact, any locally consistent partial assignment  $a_1 \dots a_k$  that satisfies  $a_i a_j \in R_{ij}$  for  $1 \leq i < j \leq k < n$  can be extended to a locally consistent partial assignment  $a_1 \dots a_k a_{k+1}$  as follows: We suppose for convenience that the domain of  $v_i$  is  $\{0, 1, \dots, d_i - 1\}$ . If  $k = 0$ , set  $a_1 \leftarrow 0$ . Otherwise, row convexity tells us that there are indices  $l_i$  and  $r_i$ , with  $0 \leq l_i \leq r_i < d_{k+1}$  for  $1 \leq i \leq k$ , such that  $a_i a_{k+1} \in R_{i(k+1)}$  if and only if  $l_i \leq a_{k+1} \leq r_i$ . Furthermore, whenever  $1 \leq i < j \leq k$ , path consistency tells us that there’s an element  $c$  such that  $l_i \leq c \leq r_i$  and  $l_j \leq c \leq r_j$ . Hence  $l_i \leq r_j$ .

Set  $a_{k+1} \leftarrow \max\{l_1, \dots, l_k\}$ . This works because it’s  $\leq \min\{r_1, \dots, r_k\}$ . ■

CRC (connected row convex) constraints—  
row convex  
column convex  
transpose  
path consistent  
domain consistent  
polynomial time  
complete relation  
partial assignment

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  $ab = a_1 b_1, \dots, a_t b_t = a' b'$  such that  $M_{a_1 b_1} = \dots = M_{a_t b_t} = 1$ , 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  $\begin{smallmatrix} 111 \\ 100 \end{smallmatrix}$  and the other is  $\begin{smallmatrix} 001 \\ 010 \end{smallmatrix}$ . 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  $\begin{smallmatrix} 001 \\ 000 \end{smallmatrix}$  for  $uv$  and  $u'v'$ ;  $\begin{smallmatrix} 001 \\ 010 \end{smallmatrix}$  for  $uu'$  and  $vv'$ ; also  $\begin{smallmatrix} 100 \\ 000 \end{smallmatrix}$  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  $R(x_1, \dots, x_m)$  is an  $m$ -ary relation on a domain  $D$ , namely a set of  $m$ -tuples  $a_1 \dots a_m$  where each  $a_j$  is in  $D$ . And let  $f(x_1, \dots, x_k)$  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:

$$a_{11} \dots a_{1m} \in R, a_{21} \dots a_{2m} \in R, \dots, \text{ and } a_{k1} \dots a_{km} \in R \text{ implies } f(a_{11}, \dots, a_{k1}) f(a_{12}, \dots, a_{k2}) \dots f(a_{1m}, \dots, a_{km}) \in R. \quad (144)$$

(The  $m$ -tuples  $a_{i1} \dots a_{im}$  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  $\Gamma$  of relations if  $f$  is a polymorphism of every relation in  $\Gamma$ . 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  $\Gamma$ ’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

$$\text{if } a_1 \vee b_1 \text{ and } a_2 \vee b_2 \text{ and } a_3 \vee b_3, \text{ then } f(a_1, a_2, a_3) \vee f(b_1, b_2, b_3). \quad (145)$$

For brevity let’s write ‘ $[a_1 a_2 a_3]$ ’ 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

$$([001] \vee [110]) \wedge ([010] \vee [101]) \wedge ([011] \vee [100]) \wedge ([011] \vee [101]) \wedge ([011] \vee [110]) \wedge ([101] \vee [110]). \quad (146)$$

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  $[a_1 \dots a_k]$

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  $[001], [010], \dots, [110]$ . Every clause in that formula is necessary. For if, say,  $([011] \vee [110])$  were absent, we could set  $[011] = [110] = 0$  and all other variables to 1, violating (145). Exercise 393 shows that, for  $k = (1, 2, 3, 4, 5, \dots)$ , the OR relation has exactly  $(2, 6, 40, 1376, 1314816, \dots)$   $k$ -ary polymorphisms—an interesting sequence!

Instead of OR, let's consider the relation ' $\leq$ ', namely  $R = \{00, 01, 11\}$ . Its  $k$ -ary polymorphisms are, by definition, characterized by the condition

$$a_1 \leq b_1 \text{ and } \dots \text{ and } a_k \leq b_k \text{ implies } f(a_1, \dots, a_k) \leq f(b_1, \dots, b_k); \quad (147)$$

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

$$a_1 \vee b_1 \vee c_1 \text{ and } a_2 \vee b_2 \vee c_2 \text{ and } a_3 \vee b_3 \vee c_3 \text{ implies } f(a_1, a_2, a_3) \vee f(b_1, b_2, b_3) \vee f(c_1, c_2, c_3), \quad (148)$$

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  $\Gamma$  of relations over the domain  $D = \{0, 1, \dots, d-1\}$ , the indicator problem  $\mathcal{I}_k(\Gamma)$  has  $d^k$  variables  $[x_1 \dots x_k]$ , one for each  $k$ -tuple with  $0 \leq x_1, \dots, x_k < d$ , representing the values  $f(x_1, \dots, x_k)$  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  $a_{11} \dots a_{1m}, \dots, a_{k1} \dots a_{km}$  all belong to  $R$  generates a constraint

$$[a_{11} \dots a_{k1}][a_{12} \dots a_{k2}] \dots [a_{1m} \dots a_{km}] \in R \quad (149)$$

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  $[x_1 \dots x_k]$ : the value of  $f(x_1, \dots, x_k)$

For example, suppose  $\Gamma = \{R_1, R_2\}$  contains the two relations  $R_1 = \{01, 10, 11\}$  (OR) and  $R_2 = \{00, 01, 11\}$  ( $\leq$ ) that we just looked at. Then  $\mathcal{I}_3(\Gamma)$  involves the eight variables  $[000], [001], \dots, [111]$ . We get  $3^3 = 27$  matrices  $A$  for  $R_1$ ,

$$\begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, \dots, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix};$$

they generate the 27 constraints  $[000][111] \in R_1, [001][110] \in R_1, [001][111] \in R_1, \dots, [111][111] \in R_1$ . In a similar way we get 27 matrices  $A$  for  $R_2$ , leading to the 27 further constraints  $[000][000] \in R_2, [000][001] \in R_2, \dots, [111][111] \in R_2$ .

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

$$[a_{11} \dots a_{k1}] = b_1 \text{ and } [a_{12} \dots a_{k2}] = b_2 \text{ and } \dots \text{ and } [a_{1m} \dots a_{km}] = b_m, \\ \text{for each } b_1 b_2 \dots b_m \in R; \quad (150)$$

we omit all cases where  $a_{1i} \dots a_{ki} = a_{1j} \dots a_{kj}$  and  $b_i \neq b_j$ .

The indicator problem  $\mathcal{I}_k(\Gamma)$  always has at least  $k$  solutions, regardless of  $\Gamma$ , because we can easily verify that the  $j$ th *projection function*

$$\pi_j(x_1, \dots, x_k) = x_j \quad (151)$$

is always a polymorphism, for  $1 \leq j \leq k$ . 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  $\Gamma$ , but also because it tells us how to make “gadgets” from  $\Gamma$ . A *gadget* is a combination of constraints that builds a new constraint, often using auxiliary variables as we did in (75). If  $\Gamma$  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,  $'x + y + z \neq 1'$ , 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  $\pi_j$  (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.)

Columns [001], [010], and [011] of (152) are somewhat remarkable because they jointly contain only the four tuples  $\hat{R} = \{000, 011, 101, 111\}$ , 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  $\hat{R}$ , 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  $\hat{R}$  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  $[000] \leq [001]$ ; in essence,  $R(x, y, y)$  is a gadget for  $'x \leq y'$ . (See exercise 402.) When all the detritus is cleared away, a very simple gadget emerges:

$$\hat{R}(x, y, z) = R(x, y, z) \wedge R(x, z, z). \quad (153)$$

Conversely,  $\mathcal{I}_3(\{\hat{R}\})$  gives us a gadget for  $R$  in terms of  $\hat{R}$  (exercise 404).

Another four-element relation,  $M = \{000, 001, 011, 111\}$ , can be seen in columns [000][001][011] of (152). So  $M$ , like  $\hat{R}$ , can be expressed in terms of  $R$ . But that fact is no surprise, because  $M$  is just the relation  $'x \leq y \leq z'$ ; 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,  $\{[0000], [0001], \dots, [1111]\}$ , and  $5^4 = 625$  constraints. It has 17 solutions, which are analogous to (152) but they include also the Boolean functions  $w$ ,  $w \vee x$ ,  $w \vee y$ ,  $w \vee x \vee y$ ,  $\dots$ ,  $w \vee x \vee y \vee z$ . Columns [0101][1010][1100] of those solutions turn out to have only seven of the eight binary possibilities, namely

$$T = \{000, 010, 011, 100, 101, 110, 111\}. \quad (154)$$

In other words,  $T$  is the Boolean relation  $'xyz \neq 001'$ , 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

$$R(x, y, z) = T(x, y, z) \wedge T(y, z, x) \wedge T(z, x, y). \quad (155)$$

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:

$$T(x, y, z) = R(v, w, z) \wedge R(v, x, x) \wedge R(w, y, y), \text{ for some } v \text{ and } w. \quad (156)$$

truth table  
max-closed  
auxiliary variables  
Goldberg  
3SAT

**NP-complete families of CSPs.** When a set  $\Gamma$  of relations on a domain  $D$  has only the projection functions  $\pi_j$  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  $\Gamma$ . For if  $R$  is an  $m$ -ary relation with  $k$  tuples, we can find  $m$  columns of  $\mathcal{I}_k(\Gamma)$  where the tuple  $\tau$  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:

$$\begin{array}{cccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array} \quad (157)$$

In this list, the values of variable  $[a_1 a_2 a_3 a_4]$  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,

$$\begin{aligned} R_0(x, y, z) &= (x \vee y \vee z), & R_1(x, y, z) &= (\bar{x} \vee y \vee z), \\ R_2(x, y, z) &= (\bar{x} \vee \bar{y} \vee z), & R_3(x, y, z) &= (\bar{x} \vee \bar{y} \vee \bar{z}), \end{aligned} \quad (158)$$

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  $\Theta$  called a ternary *shortcut* (see exercise 412). It has only four tuples:

$$\Theta = \{01, 12, 20, 02\}. \quad (159)$$

(A three-element domain is often represented more elegantly when the elements are given the “balanced” labels  $\{-, 0, +\}$ ; then  $\Theta$  becomes  $\{-0, 0+, +-, -+\}$ , and its structure becomes more clear. But we shall stick to  $\{0, 1, 2\}$  for now.)

NP-complete-  
potent  
pi as source  
magic masks  
3SAT  
one-in-three satisfiability  
shortcut



Since  $\Theta$  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

$$\begin{array}{cccccccccccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \end{array} \quad (160)$$

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  $\Theta$  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

$$\Gamma = \{\neq, \{0\}, \{1\}, \dots, \{d-1\}\}, \quad (161)$$

where  $\{a\}$  is the unary relation that’s satisfied only by the constant  $a$ , we *do* get a potent set of relations for  $\{0, 1, \dots, d-1\}$ , 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  $\Gamma$  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  $\Gamma$  be a set of relations on a domain  $D$  of size  $d \geq 3$ . Then  $\Gamma$  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  $f(x_1, \dots, x_k)$  is a  $k$ -ary function we say that  $f$  *depends on*  $x_j$  if there are  $k+1$  elements  $a_1, \dots, a_k, a'_j$  such that the substitution of  $a'_j$  for  $a_j$  gives  $f(a_1, \dots, a_j, \dots, a_k) \neq f(a_1, \dots, a'_j, \dots, a_k)$ . The “essential arity” of  $f$ , denoted by  $\text{ea}(f)$ , is the number of variables  $x_j$  on which  $f$  depends. In particular,  $f$  is constant if and only if  $\text{ea}(f) = 0$ . 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

$$f(x_1, \dots, x_k) = h(x_{j_1}, \dots, x_{j_r}), \quad \text{where } j_1 < \dots < j_r \text{ and } h \text{ is full.} \quad (162)$$

For example, consider the function

$$q(x_1, \dots, x_k) = (x_1 + 2x_2 + \dots + kx_k) \bmod d \quad (163)$$

on the domain  $\{0, 1, \dots, d-1\}$ . It clearly depends on variable  $x_j$  if and only if  $j$  is not a multiple of  $d$ ; so its essential arity is  $k - \lfloor k/d \rfloor = \lceil \frac{d-1}{d}k \rceil$ . If, say,  $k = 7$  and  $d = 3$ , we have  $q(x_1, \dots, x_7) = h(x_1, x_2, x_4, x_5, x_7)$ , where the function  $h(x_1, \dots, x_5) = (x_1 - x_2 + x_3 - x_4 + x_5) \bmod 3$  is full.

pi as source  
3-coloring a graph  
nonequality  
unary relation  
constant  
depends on  
essential arity  
notation  $\text{ea}(f)$   
full

Returning to Theorem F, one direction of the proof is obvious: If  $\mathcal{I}_d(\Gamma)$  has more than  $d$  solutions,  $\Gamma$  certainly isn't potent. Suppose, therefore, that every  $d$ -ary polymorphism of  $\Gamma$  is a projection, although  $\Gamma$  does have a  $k$ -ary polymorphism  $f(x_1, \dots, x_k)$  that *isn't* a projection, where  $k$  is as small as possible.

We can't have  $k < d$ , because  $g(x_1, \dots, x_d) = f(x_1, \dots, x_k)$  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

$$g(x_1, \dots, x_k) = f(x_1, \dots, x_{j-1}, x_i, x_{j+1}, \dots, x_k) \quad (164)$$

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  $B$ :

$$\text{If } X \subseteq A, \quad X^\nearrow = \{y \in B \mid xRy \text{ for all } x \in X\}. \quad (165)$$

$$\text{If } Y \subseteq B, \quad Y^\swarrow = \{x \in A \mid xRy \text{ for all } y \in Y\}. \quad (166)$$

[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  $3 \times 5$  matrix and the possibilities for  $R$ ,  $X$ , and  $Y$  might be

$$R = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix};$$

$X$	$X^\nearrow$	$Y$	$Y^\swarrow$	$Y$	$Y^\swarrow$	$Y$	$Y^\swarrow$	$Y$	$Y^\swarrow$
$\epsilon$	01234	$\epsilon$	012	3	01	4	1	34	1
0	13	0	12	03	1	04	1	034	1
1	034	1	0	13	0	14	$\epsilon$	134	$\epsilon$
01	3	01	$\epsilon$	013	$\epsilon$	014	$\epsilon$	0134	$\epsilon$
2	02	2	2	23	$\epsilon$	24	$\epsilon$	234	$\epsilon$
02	$\epsilon$	02	2	023	$\epsilon$	024	$\epsilon$	0234	$\epsilon$
12	0	12	$\epsilon$	123	$\epsilon$	124	$\epsilon$	1234	$\epsilon$
012	$\epsilon$	012	$\epsilon$	0123	$\epsilon$	0124	$\epsilon$	01234	$\epsilon$

(Here ' $\epsilon$ ' stands for  $\emptyset$  and '01234' stands for  $\{0, 1, 2, 3, 4\}$ , etc.)

One of the immediate consequences of (165) and (166) is that

$$X \subseteq X^{\nearrow\swarrow} \quad \text{and} \quad Y \subseteq Y^{\swarrow\searrow}; \quad (167)$$

indeed,  $X^{\nearrow\swarrow} = \{x \in X \mid xRy \text{ for all } y \in X^\nearrow\}$ , 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:

$$X_0 \subseteq X_1 \quad \text{implies} \quad X_0^\nearrow \supseteq X_1^\nearrow; \quad (168)$$

$$Y_0 \subseteq Y_1 \quad \text{implies} \quad Y_0^\swarrow \supseteq Y_1^\swarrow. \quad (169)$$

essential arity  
polarities-  
polarities  
Birkhoff

Applying (168) to (167) tells us, for example, that  $X^\nearrow \supseteq (X^{\nearrow\swarrow})^\nearrow = X^{\nearrow\swarrow\swarrow}$ . But we also know, by setting  $Y = X^\nearrow$  in (167), that  $X^\nearrow \subseteq (X^\nearrow)^{\swarrow\swarrow} = X^{\nearrow\swarrow\swarrow}$ . Hence

$$X^\nearrow = X^{\nearrow\swarrow\swarrow} \quad \text{and} \quad Y^\swarrow = Y^{\swarrow\swarrow\swarrow}. \quad (170)$$

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

$$\overline{X} = X^{\nearrow\swarrow}, \quad \overline{Y} = Y^{\swarrow\swarrow}, \quad (171)$$

calling those sets the “closures” of  $X$  and  $Y$ . According to (167) and (170),

$$X \subseteq \overline{X}, \quad \overline{\overline{X}} = \overline{X}, \quad Y \subseteq \overline{Y}, \quad \overline{\overline{Y}} = \overline{Y}. \quad (172)$$

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  $3 \times 5$  relation  $R$  above are

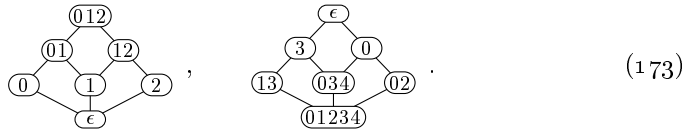
$$\epsilon, 0, 1, 2, 01, 12, 012 \subseteq A; \quad \epsilon, 0, 3, 02, 13, 034, 01234 \subseteq B.$$

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  $3 \times 5$  example it can be seen in the set-inclusion diagrams



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  $2 \times 2$  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  $f(x_1, \dots, x_k)$  is in  $\mathcal{C}$  if and only if either

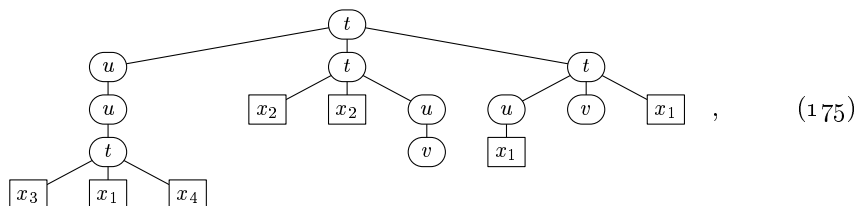
- i)  $f(x_1, \dots, x_k) = x_j$  for some  $j$  with  $1 \leq j \leq k$ ; or
- ii)  $f(x_1, \dots, x_k) = g(h_1(x_1, \dots, x_k), \dots, h_l(x_1, \dots, x_k))$ , where  $g$  is an  $l$ -ary operation in  $O$  and  $h_1, \dots, h_l$  are  $k$ -ary operations in  $\mathcal{C}$ .

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

$$f(x_1, x_2, x_3, x_4) = t(u(u(t(x_3, x_1, x_4))), t(x_2, x_2, u(v)), t(u(x_1), v, x_1)). \quad (174)$$

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



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  $a_1x_1 + \dots + a_kx_k$ , 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  $a_1x_1 + \dots + a_kx_k$ , 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

After some experimentation, a simple pattern emerges:

$$a_1x_1 + \cdots + a_kx_k \in \mathcal{C} \iff \text{exactly one of the coefficients is odd.} \quad (176)$$

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  $h_1(x_1, \dots, x_k)$  and  $h_2(x_1, \dots, x_k)$  have exactly one odd coefficient, so does  $h_1(x_1, \dots, x_k) + 2h_2(x_1, \dots, x_k)$ . Conversely, if  $f(x_1, \dots, x_k) = a_1x_1 + \cdots + a_kx_k$  has exactly one odd coefficient and is not a projection function, some coefficient  $a_j$  exceeds 1. Hence  $g(x_1, \dots, x_k) = f(x_1, \dots, x_k) - 2x_j$  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

$$m_k(x_1, \dots, x_k) = \bigvee_{1 \leq i < j \leq k} (x_i \wedge x_j) = [x_1 + \cdots + x_k > 1] \quad (177)$$

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  $l > k \geq 4$ . (See exercise 424.)

Clones are particularly interesting to us because we get them from polarities and polymorphisms. If  $\Gamma$  is any set of relations on a finite domain  $D$ , we let  $\text{Pol}(\Gamma)$  be the set of all its polymorphisms; and if  $O$  is any set of operations on  $D$ , we let  $\text{Inv}(O)$  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,  $\text{Pol}(\Gamma) = \Gamma^\nearrow$  and  $\text{Inv}(O) = O^\swarrow$ .

According to polarity theory,  $\overline{\Gamma}$  is  $\text{Inv}(\text{Pol}(\Gamma))$ , the set of all relations that are preserved by all polymorphisms of  $\Gamma$ . And  $\overline{O}$  is  $\text{Pol}(\text{Inv}(O))$ , 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  $\Gamma$  is potent, then  $\text{Pol}(\Gamma)$  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  $\text{Inv}(O)$  is the set of max-closed relations, and  $\overline{O}$  is the set of all  $f(x_1, \dots, x_k) = \max(x_{j_1}, \dots, x_{j_l})$ , for some  $1 \leq j_1 < \cdots < j_l \leq k$ .

Polarity theory also tells us that the closed sets of operators are the sets that are  $\text{Pol}(\Gamma)$  for some  $\Gamma$ . 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  $\Gamma$  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  $\Gamma$ , 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  $\text{Pol}(\text{Inv}(O))$  belongs to  $O$ . We conclude that a set of operations is “closed”,  $\overline{O} = O$ , if and only if it’s a clone.

projection functions  
median operation  
monotone self-dual Boolean functions  
self-dual Boolean functions  
Boolean functions  
Post  
 $\text{Pol}(\Gamma)$   
 $\text{Inv}(O)$   
max-closed relations

Similarly, a set of *relations* is closed if and only if it's a so-called *relational clone*. Any set of the form  $\text{Inv}(O)$  is a clone of relations, just as any set of the form  $\text{Pol}(\Gamma)$  is a clone of operations.

The closure  $\bar{\Gamma}$  of a set of relations is perhaps best understood as the set of all *gadgets* that can be made from  $\Gamma$ . According to the formula  $\bar{\Gamma} = \text{Inv}(\text{Pol}(\Gamma))$ , we start with the polymorphisms of  $\Gamma$ , 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  $\Gamma$  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  $\Gamma$ ; 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

$$\Sigma_{ij} R' = \{a_1 \dots a_m \in R' \mid a_i = a_j\}, \quad (178)$$

where  $R'$  is an  $m$ -ary relation in  $\mathcal{C}$  and  $1 \leq i < j \leq m$ .

If  $R' \subseteq D^{m'}$  and  $R'' \subseteq D^{m''}$  are  $m'$ -ary and  $m''$ -ary relations, their product

$$R' \otimes R'' = \{a_1 \dots a_{m'} b_1 \dots b_{m''} \mid a_1 \dots a_{m'} \in R' \text{ and } b_1 \dots b_{m''} \in R''\} \quad (179)$$

is the  $(m' + m'')$ -ary relation obtained by simply concatenating their tuples. And if  $\sigma = i_1 \dots i_s$  is any sequence of integers,  $1 \leq i_j \leq m'$ , the projection

$$\Pi_\sigma R' = \{a_{i_1} \dots a_{i_s} \mid a_1 \dots a_{m'} \in R'\} \quad (180)$$

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

$$\begin{aligned} R''' = R' \otimes R'' &= \{00110, 00111, 00122, 12010, 12011, 12022\}; \\ R^{\text{iv}} = \Pi_{3141413} R''' &= \{0111110, 0121210, 1010101, 1020201\}; \\ R^{\text{v}} = \Sigma_{37} R^{\text{iv}} &= \{1010101\}. \end{aligned}$$

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  $\Gamma$ . It is familiar in the theory of databases, where the “join” operation  $R' \bowtie R''$  is a combination of the  $\otimes$ ,  $\Sigma$ , and  $\Pi$  operators.

**\*The dichotomy theorem.** We've seen that some families  $\Gamma$  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  $\Gamma$  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  $\Gamma$  in that domain are considerably more complicated, but their clones  $\text{Pol}(\Gamma)$  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  $f(x_1, \dots, x_k)$  on domain  $D$  is said to be *idempotent* if we have

$$f(x, x, \dots, x) = x \quad \text{for all } x \in D. \quad (181)$$

And we call it a *near unanimity* operator, or NU for short, if  $k > 2$  and

$$f(y, x, x, \dots, x) = f(x, y, x, \dots, x) = \dots = f(x, x, \dots, x, y) = x \quad (182)$$

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

$$f(y, x, x, \dots, x) = f(x, y, x, \dots, x) = \dots = f(x, x, \dots, x, y), \quad (183)$$

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  $\Gamma$  be a finite set of relations on a finite domain  $D$ . The decision problem for every CSP whose constraints all belong to  $\Gamma$  is solvable by a known polynomial-time algorithm if  $\Gamma$  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

lax “Taylor term,” namely a  $k$ -ary polymorphism  $f$  that satisfies  $k$  identities

$$f(u_{i1}, \dots, u_{ik}) = f(v_{i1}, \dots, v_{ik}) \text{ and } u_{ii} \neq v_{ii}, \quad \text{for } 1 \leq i \leq k, \quad (184)$$

where each  $u_{ij}$  and  $v_{ij}$  is either  $x$  or  $y$ . The identities needn’t be distinct. When  $k = 4$ , for example,  $f(x_1, \dots, x_4)$  is a lax Taylor term if it satisfies

$$f(y, x, y, x) = f(x, y, x, x) = f(y, x, x, y), \quad (185)$$

because the first equality covers  $i \neq 4$  in (184) and the second covers  $i \neq 3$ .

Amazingly, an LWNUP polymorphism exists if and only if there are polymorphisms  $p(x, y, z)$  and  $q(x, y, z)$  that satisfy the three identities

$$p(x, y, x) = q(x, y, x); \quad p(x, y, y) = q(x, x, y) = q(y, x, x); \quad (186)$$

$p$  and  $q$  are called a *lax Maróti pair*. The tractability of  $\Gamma$  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  $D = \{0, 1, 2, 3\}$ :

$$G = \{12, 21, 13, 23, 30\}. \quad (187)$$

The indicator problem  $\mathcal{I}_3(\{G\})$  shows that it has 126760 ternary polymorphisms, including four that are LWNUP. One of the latter is

$$f(x, y, z) = (0 \in \{x, y, z\} ? 0 : 3 \in \{x, y, z\} ? 3 : \langle xyz \rangle). \quad (188)$$

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

$$Z = \{12, 23, 31, 20, 30\}. \quad (189)$$

This one has a huge number of  $k$ -ary polymorphisms—in fact,  $k \cdot 4^{k-1} - 3^{k-1}$ —yet *none* of them are LWNUP! 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  $\Gamma$ , 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  $\Gamma$  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  $\Gamma$  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).

Taylor term  
lax Maróti pair  
Maróti pair  
indicator problem  
Bulatov  
Zhuk  
domain  
unary relations  
idempotent



**\*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:

$$\begin{aligned}\lambda(x) &= \min\{0 \leq y < n \mid x R y\}, \text{ for } 0 \leq x < m; \\ \rho(y) &= \min\{0 \leq x < m \mid x R y\}, \text{ for } 0 \leq y < n.\end{aligned}\tag{190}$$

They suffice because  $x R y$  if and only if  $x \geq \rho(y)$  and  $y \geq \lambda(x)$ . (Here either  $\lambda(x)$  or  $\rho(y)$  might be  $\infty$ , 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  $f(x_1, \dots, x_k)$  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,  $f(x_1, \dots, x_k)$ , is  $k$ -ary, where  $k \geq 3$ . 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  $D = \{0, 1, \dots, d-1\}$ , one for each choice of  $(x_1, \dots, x_k)$  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  $i_1 \dots i_k$  where  $1 \leq i_1 < \dots < i_k \leq m$ . 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,  $\dots$ ,  $m$ -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

$$f(x, y, z) = (x, y, z \text{ distinct? } 0: \langle xyz \rangle). \quad (191)$$

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  $\pi = (3.02100333\dots)_4$ :

$$R = \langle \{302100333, 122220202, 011220300, 203103010, \dots, 103200202\} \rangle. \quad (192)$$

One of  $R$ 's more understandable seeds is the following subset of 17 tuples inspired by Lemma N,

$$R' = \{000100010, 000200330, 000210002, 011220300, 020220000, 022220002, 023200031, 100220202, 102210002, 103200202, 122220202, 203103010, 300130310, 301200000, 301212022, 302100000, 302100333\}. \quad (193)$$

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  $\Pi_{ij}$  with  $1 \leq i < j \leq 9$ . According to Lemma N, a given tuple  $\tau$  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  $\Pi_{ij}$ . For example, 000100010 is the only tuple  $\tau$  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  $\leq 1,140,000$ .

decomposable  
Baker  
Pixley  
Jeavons  
Cohen  
Cooper  
median operator  
majority operator  
pi as source  
stub  
minimal stub

Let  $R_0 = \{000100000, \dots, 023200031\}$  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?

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  $\Pi_{25}$  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

$$\Pi_\sigma \langle R \rangle_f = \langle \Pi_\sigma R \rangle_f, \quad \text{for all relations } R \text{ and operations } f. \quad (194)$$

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  $\sigma$  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  $\Pi_{ij}$  as  $R_0$  does. So we want the union, over  $1 \leq i < j \leq 9$ , of enough tuples to achieve this. Consider, for example, the case  $ij = 23$ . Rule (194) tells us that

$$\langle \Pi_{123}R' \rangle = \Pi_{123}\langle R' \rangle = \Pi_{123}R. \quad (195)$$

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  $\Pi_{23}$ . 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,

$$\begin{aligned} R'_0 = \{ & 000100010, 000100300, 000100310, 000200000, 000200330, \\ & 000210002, 000220000, 001200000, 001210002, 002100000, 002200030, \\ & 002200300, 002210002, 003100010, 003100030, 003200000, 003200002, \\ & 011220300, 020220000, 022220002, 023200031 \}. \end{aligned} \quad (196)$$

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.

witnesses  
compact stub-

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  $\tau$  to  $S$ ,” where  $\tau$  is a tuple, is a shorthand for the following operation: “If all of the  $(k-1)$ -wise projections of  $\tau$  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  $f(x_1, \dots, x_k)$  be a  $k$ -ary near unanimity operation on the domain  $D = \{0, 1, \dots, d-1\}$ , 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  $i_1 \dots i_{k-1} \in \binom{[n]}{k-1}$ , and each  $d$ -ary string  $d_1 \dots d_{k-1} \in D^{k-1}$ , contribute to  $S$  the  $n$ -tuple  $t_1 \dots t_n$  that has  $t_{i_1} = d_1, \dots, t_{i_{k-1}} = d_{k-1}$ , 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  $\tau$ , where  $\tau$  is any element of  $S$ . Otherwise input a new constraint  $R$ , where  $R \subseteq D^r$  is an  $r$ -ary relation that constrains variables  $j_1, \dots, j_r$ . (We assume that  $1 \leq r < k$  and that  $f$  is a polymorphism of  $R$ .) Let  $\rho$  be the string  $j_1 \dots j_r$ .
- 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

$$T = \langle S_0 \rangle \cap R(j_1, \dots, j_r), \quad (197)$$

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  $\Pi_\sigma$ .] If  $\sigma = i_1 \dots i_{k-1}$ , let  $\hat{\sigma}$  be the string  $\sigma\rho = i_1 \dots i_{k-1}j_1 \dots j_r$ , 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  $\tau$  of  $\langle \Pi_{\hat{\sigma}} S_0 \rangle$ , and contribute to  $S$  a tuple of  $\langle S_0 \rangle$  that witnesses  $\tau$  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  $\sigma = i_1 \dots i_{k-1}$ , when we’re imposing a new constraint  $R$  on  $r < k$  variables  $\rho = j_1 \dots j_r$ , where  $\sigma$  and  $\rho$  are given strings.

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  $l = |\hat{\sigma}| = |\{i_1, \dots, i_{k-1}, j_1, \dots, j_r\}|$ . Notice that  $k - 1 \leq l \leq k - 1 + r \leq 2k - 2$ ; the actual value of  $l$  depends on overlaps between the  $i$  and  $j$  coordinates. There's a string  $\hat{\rho}$  such that, if  $\tau = t_1 \dots t_l$  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  $\tau$  in  $H$  will in fact be the projection  $\Pi_{\hat{\sigma}} \tau^*$  of some known  $n$ -tuple  $\tau^*$  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  $\tau^*$  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  $m \leq d^l$ .

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:

$$f(x, x, y) = y = f(y, x, x). \quad (198)$$

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  $1 \leq i \leq m$  and  $a, a' \in D$ , such that  $R$  contains tuples  $\tau = t_1 \dots t_m$  and  $\tau' = t'_1 \dots t'_m$  with equal prefixes of length  $i - 1$ :

$$t_1 = t'_1, \dots, t_{i-1} = t'_{i-1}, t_i = a, \text{ and } t'_i = a'. \quad (199)$$

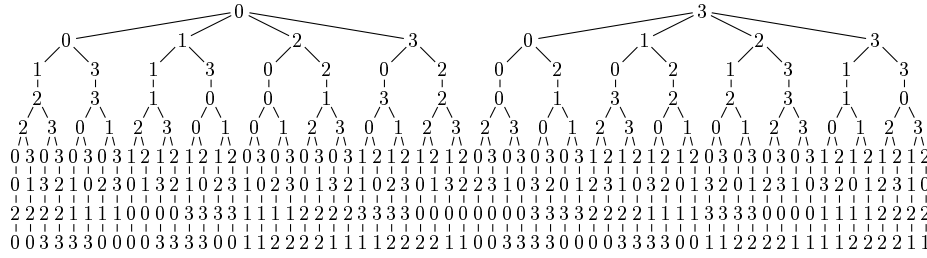
In particular,  $R$  has a forking  $(i, a, a)$  with  $\tau = \tau'$  for each  $a$  in its projection  $\Pi_i R$ .

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  $x_1 \dots x_{i-1} a x_{i+1} \dots x_m \in R$ ,  $x_1 \dots x_{i-1} a' y_{i+1} \dots y_m \in R$ , and  $z_1 \dots z_{i-1} a' z_{i+1} \dots z_m \in R$ , we have  $z_1 \dots z_{i-1} a w_{i+1} \dots w_m \in R$ , 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  $\Pi_i R$ .

polynomial time  
Maltsev  
forking  
transitive law  
equivalence relation  
set partition

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”

$$p(x, y, z) = x \oplus y \oplus z, \quad (200)$$

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  $\langle \Pi_{12\dots i} R' \rangle = \Pi_{12\dots i} R$  for  $i = 2, \dots, m$ . (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.

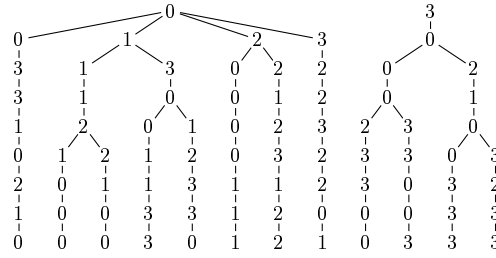
The local condition in this case needs to be slightly trickier than before, because a forking can involve two distinct tuples, while a projection involves only one. Here's the new definition: The tuples  $\tau_1, \tau_2, \dots, \tau_M$  of  $R$  are a compact Maltsev stub for  $R$  if, for  $1 < j \leq M$ ,  $\{\tau_1, \dots, \tau_j\}$  has at least two more forkings than  $\{\tau_1, \dots, \tau_{j-2}\}$ . (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  $\tau$ '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.

trie structure  
forest  
bitwise parity constraint  
parity constraint  
data compression  
Maltsev stub  
compact

For example, one of many compact stubs for the relation  $P$  of Fig. 120 is

$$P' = \{003310210, 011121000, 011122100, 013001133, \\ 013012330, 020000111, 022123122, 032232201, \\ 300023300, 300033003, 302100333, 302103233\}. \quad (201)$$

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  $\tau = t_1 \dots t_m$  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  $\tau$  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

$$P'_3 = \{300023300, 302100333, 302113130, 310331123, 310332023, \\ 312212110, 323333001, 331101311, 333021222\}. \quad (202)$$

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:

**Algorithm V** (*Visiting every solution to a CSP*). Given a CSP  $\mathcal{P}$  with variables  $\{v_1, \dots, v_n\}$  over the domain  $D = \{0, 1, \dots, d-1\}$ , let  $T_0$  be the set of all  $n$ -tuples  $a_1 \dots a_n$  such that  $(v_1 = a_1, \dots, v_n = a_n)$  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  $a_1 \dots a_n \leftarrow 0 \dots 0$ , and set  $S_j$  to a representation of

$$T_j = T_0 \cap (v_1 = a_1) \cap \dots \cap (v_j = a_j), \quad (203)$$

for  $j = 0, 1, \dots, n$ .

**V2.** [Solution found?] If  $S_n \neq \emptyset$ , visit the solution  $a_1 \dots a_n$ .

**V3.** [Done?] (At this point we have visited every solution to  $\mathcal{P}$  that is lexicographically less than or equal to  $a_1 \dots a_n$ .) Let  $j$  be maximum such that  $j \leq n$ ,  $j > 0$ ,  $S_{j-1} \neq \emptyset$ , and  $a_j \neq d-1$ . Terminate if no such  $j$  exists.

**V4.** [Advance.] Set  $a_j \leftarrow a_j + 1$  and  $a_{j+1} \dots a_n \leftarrow 0 \dots 0$ . Update the representations  $S_j, \dots, S_n$  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  $\Gamma$  of relations is tractable, provided that  $\Gamma$  is a *core* —that is, provided that every unary polymorphism of  $\Gamma$  is a permutation of  $D$ . For in that case, exercise 443 shows that  $\Gamma$  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

$$Y = \{001, 010, 100, 222\}. \quad (204)$$

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  $2 \dots 2$ .

The situation gets more interesting when we have a constraint like

$$Y^+ = \{001, 010, 100, 222, 223, 232, 233, 322, 323, 332, 333\}. \quad (205)$$

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.

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



**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.

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  $(x_1, \dots, x_n)$  such that  $x_j \in D_j$  for  $1 \leq j \leq n$ , 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:

$$\text{For } 1 \leq k \leq n, \text{ set } R_{ij} \leftarrow R_{ij} \cap R_{ik} R_{kk} R_{kj} \text{ for } 1 \leq i, j \leq n. \quad (206)$$

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

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

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  $\alpha$ -valuations,  $\beta$ -valuations,  $\sigma$ -valuations, and  $\rho$ -valuations. Every  $\alpha$ -valuation was a  $\beta$ -valuation; every  $\beta$ -valuation was a  $\sigma$ -valuation; every  $\sigma$ -valuation was a  $\rho$ -valuation; and every  $\rho$ -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  $\beta$ -valuations were exactly the same as Golomb’s graceful labelings, and interest in the subject began to take off.

Rosa’s  $\rho$ -valuations eventually became known as “rainbow graceful” —a nice coincidence, because “ $\rho$ ” 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

$$\text{revise}(R_{ij}, x_i) = \begin{cases} \text{For each } a \in D_i, \\ \text{if } D_j \& (\text{row } a \text{ of } R_{ij}) = 0, \\ \text{set } D_i \leftarrow D_i \setminus a, \end{cases} \quad (207)$$

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  $|D_j| < |D_i|$ :

$$\text{revise}(R_{ij}, x_i) = \begin{cases} \text{Set } z \leftarrow 0 \text{ and, for each } b \in D_j, \\ \text{set } z \leftarrow z \mid (\text{column } b \text{ of } R_{ij}); \\ \text{then set } D_i \leftarrow D_i \& z. \end{cases} \quad (208)$$

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  $\Gamma$  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  $\text{Пол}(\hat{R})$  and  $\text{Инв}(\mathfrak{F})$ . 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/](https://github.com/sofdem/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  $\epsilon$ . 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  $u - v$  in  $G \iff h(u) - h(v)$  in  $G'$ ?)
  - d) Does graph  $G = (V, E)$  have bandwidth  $k$ ? (Can its vertices be given distinct integer labels so that  $u - v$  implies  $|h(u) - h(v)| \leq k$ ?)
  - 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  $x_1 \dots x_k$  is said to be *unlike* the  $k$ -tuple  $x'_1 \dots x'_k$  if  $x_j \neq x'_j$  for  $1 \leq j \leq k$ . It's convenient to write ‘ $x_1 \dots x_k \parallel x'_1 \dots x'_k$ ’ 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  $(V, \neq)$  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  $\Sigma$ , as defined in (g).
  - 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  $N \rightarrow \infty$ .
  - 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

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  $\{1, \dots, 8, A, \dots, H\}$ .
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

$$f_{(l'q_k)k} + f_{(l'q_k+1)k} + \dots + f_{(l'+l''q_k-1)k} \geq r_k - (l' + l'')p_k$$

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  $\{7 \cdot A, 2 \cdot B, 5 \cdot C, 4 \cdot D, 4 \cdot E, 2 \cdot F, 4 \cdot G, 2 \cdot H\}$  have overall requirements  $(r_0, \dots, r_4) = (15, 20, 10, 12, 6)$ , 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  $1 \leq p < q$ , 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  $G_{pq}(z) = \sum_{n \geq 0} C_{pqn} z^n$  for  $1 \leq p < q \leq 5$ , 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  $i \neq j$  with  $0 \leq i, j < 7$  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  $0 \leq j < 7$ .

symmetry  
car sequencing problem  
MCC problem  
 $(p/q)$ -string  
generating functions  
extreme  
plane partitions  
Szilassi  
polyhedron



- **27.** [M28] Three-dimensional space can be discretized into little “cubies,” where cubie  $(i, j, k)$  consists of all points  $(x, y, z)$  with  $i \leq x \leq i+1$ ,  $j \leq y \leq j+1$ , and  $k \leq z \leq k+1$ . (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  $0 \leq i < m$  and  $0 \leq j < n$ , its *histoscape* is the set of cubies  $(i, j, k)$  for  $0 \leq k < a_{ij}$ . (For example, Fig. 101(d) is the histoscape for  $\begin{pmatrix} 4 & 3 \\ 1 & 2 \end{pmatrix}$ .)

How many  $2 \times 2$  matrices with  $0 \leq a_{ij} < 10$  have a histoscape that’s a 3VP?

- **28.** [M27] Continuing exercise 27, how many of the  $10^{64}$   $8 \times 8$  matrices whose entries satisfy  $0 \leq a_{ij} < 10$  for  $0 \leq i, j < 8$  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  $k$ th  $m \times n$  histoscape whose entries satisfy  $0 \leq a_{ij} < t$ , 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  $8 \times 8$  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  $2 \times 2$  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  $2 \times 2$  whirlpool permutations of  $\{1, 2, 3, 4\}$ :

$$\begin{pmatrix} 1 & 2 \\ 4 & 3 \end{pmatrix} \quad \begin{pmatrix} 1 & 4 \\ 2 & 3 \end{pmatrix} \quad \begin{pmatrix} 2 & 1 \\ 3 & 4 \end{pmatrix} \quad \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix} \quad \begin{pmatrix} 3 & 2 \\ 4 & 1 \end{pmatrix} \quad \begin{pmatrix} 3 & 4 \\ 2 & 1 \end{pmatrix} \quad \begin{pmatrix} 4 & 1 \\ 3 & 2 \end{pmatrix} \quad \begin{pmatrix} 4 & 3 \\ 1 & 2 \end{pmatrix}.$$

- a) The  $4 \times 4$  matrix at the right is not quite a whirlpool permutation. Fix the problem by interchanging two rookwise adjacent elements.  $\begin{pmatrix} 16 & 3 & 2 & 13 \\ 9 & 7 & 8 & 10 \\ 5 & 6 & 12 & 11 \\ 4 & 14 & 15 & 1 \end{pmatrix}$
- 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  $\{1, \dots, mn\}$ ?
- 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  $5 \times 5$  matrices are whirlpool permutations of  $\{0, 1, \dots, 24\}$ ? *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  $a_1 a_2 \dots a_{2n-1}$  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,  $(U_1, \dots, U_5) = (1, 2, 14, 204, 5104)$ .
- 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  $U(z) = U_1 z/1! + U_2 z^3/3! + U_3 z^5/5! + \dots$ .
- 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  $\{0, \dots, 2n-1\}$  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  $5 \times 5$  matrices can be completed to a whirlpool permutation of  $\{1, 2, \dots, 25\}$  in exactly one way?

(i)	1	3	5	7	9
	17				
			25		
	2	4	6	8	10

(ii)	3	14	15	9	2
		6		5	

(iii)	3	14	15		
		9	2	6	
	5				
		1	25	22	
			11	21	19

(iv)	3		14		15
	9		2		6
			5		
	1		21		25
	4		18		22

► **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

$$D(v_0, v_1, v_2, v_3) > 0, \quad \text{where } D(v_0, v_1, v_2, v_3) = \det \begin{pmatrix} x_0 & y_0 & z_0 & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{pmatrix}.$$

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  $(\pi, \phi, \gamma)$  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  $X_\epsilon$  be the interior of the closed set  $X \cap S_\epsilon(v_0)$ , where

$$S_\epsilon(v_0) = \{v \mid \|v - v_0\| \leq \epsilon\} = \{(x, y, z) \mid (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 \leq \epsilon^2\}.$$

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  $X_\epsilon$ .

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.

- Show that exactly 64 of those placements make a 3VP in which the center of the box is a vertex.
- Furthermore, if that 3VP is in general position, we'll be able to *see* its central vertex in exactly 32 cases.
- Draw those 32 pictures, and verify that the different possibilities for V, W, and Y junctions are precisely those shown in Table 1.
- 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.

- What is the constraint for line  $ab$  of (20) in this scheme?
- How about the lines  $np$  and  $op$ ?
- What's the answer to exercise 42, with respect to *this* model?
- 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  $H^R$ .

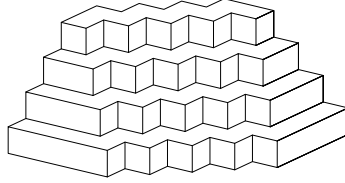
- 47. [M25] An HC network is called *realizable* if it corresponds to at least one actual HC picture. Furthermore, that HC picture must not have a T junction whose collinear lines both lie on the outer boundary. (Such a T cannot be the image of a 3VP in general position. Notice that the line labeling problem for  $H$  is well defined regardless of whether or not  $H$  can be physically realized.)

- What is the smallest unrealizable HC network? *Hint:* It has three junctions.
- Characterize all realizable HC networks whose junctions all have type V.
- Find an HC network, consisting entirely of type W junctions, that is unrealizable because it doesn't define a planar graph.
- 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.
- True or false: If the junction  $T(a, b, c)$  in a realizable network is changed to either  $W(c, b, a)$  or  $Y(a, b, c)$ , 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



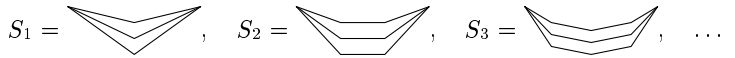
standard labeling  
smile of order  $n$   
standard  
bow tie  
biconnected  
standard  
Lucas number  
Fibonacci numbers  
twindragon fractal  
fractal

and watch the disconnected interior images as they jump in and out, before your eyes.


- Show that this picture has only one standard labeling.
  - In how many ways can the boundary junctions be labeled consistently, without regard to any of the interior junctions?
  - How many labelings are possible altogether, standard or not?
- 50. [M30] Let  $(j_0 j_1 \dots j_{q-1})$  be the boundary cycle of a realizable HC network.
- For  $0 \leq k < q$ , 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 **L**;
    - $j_k = T(j'_k, j_{k+1}, j_{k-1})$ , called case **R**;
    - $j_k = V(j_{k+1}, j_{k-1})$ , called case **V**;
    - $j_k = V(j_{k-1}, j_{k+1})$ , called case **A**;
    - $j_k = W(j_{k+1}, j'_k, j_{k-1})$ , called case **W**;
    - $j_k = Y(j_{k+1}, j'_k, j_{k-1})$ , called case **Y**.

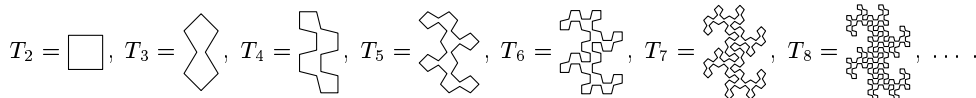
(The subscripts in ' $j_{k\pm 1}$ ' are to be understood mod  $q$ . The line  $j_k - j'_k$  in cases **L**, **R**, **W**, and **Y** is called an "inner line," although  $j'_k$  might lie on the boundary.)

- 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?
  - 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:



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  $(j_0 j_1 \dots j_{q-1})$  is **V** or **A**.
- Let  $M_k = A$  if  $j_k = \mathbf{V}$  and  $M_k = B$  if  $j_k = \mathbf{A}$ , where  $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$  and  $B = \begin{pmatrix} 1 & -1 \\ -1 & 0 \end{pmatrix}$  are  $2 \times 2$  matrices. Prove that the number of ways to label the boundary cycle  $(j_0 j_1 \dots j_{q-1})$  is  $\text{trace}(M_0 M_1 \dots M_{q-1}) + L_q$ , where  $L_q$  is a Lucas number.
  - Show that  $2F_q \leq \text{trace}(A^p B^{q-p}) + L_q \leq 2L_q$  for  $0 \leq p \leq q$ . What  $p$  gives equality?
  - 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  $n \geq 2$ , where  $T_n$  has  $2^n$  junctions:



The clockwise path  $T_n$  turns left or right at step  $k$  and at step  $k + 2^{n-1}$  according as the Jacobi symbol  $(\frac{-1}{k})$  is  $-1$  or  $+1$ , for  $1 \leq k \leq 2^{n-1}$ . (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

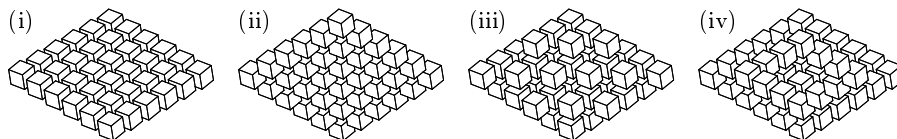


- What is the HC network for the Penrose  $n$ -gon?
- In how many ways can the Penrose  $n$ -gon be labeled consistently?
- Is the Penrose  $n$ -gon weakly realizable for any  $n \geq 3$ ?

59. [20] Explain how to obtain (32) as the projection of nine “squashed” cubes.

60. [M22] In how many ways can Reutersvård’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)?

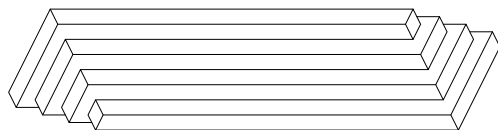
62. [23] The 36 boxes in the  $6 \times 6$  hexagonal arrays of exercise 61 involve 85 pairs  $(A, B)$  of adjacent boxes:  $30 = 6 \cdot 5$  pairs in direction  $\nearrow$ ;  $30 = 5 \cdot 6$  pairs in direction  $\nwarrow$ ; and  $25 = 5 \cdot 5$  pairs in direction  $\uparrow$ . In every case we’re 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  $C < A$ .

- In how many ways can those 85 relations be specified, without any such non-transitive triplets? *Hint:* This is a CSP.
- Generalize to  $m \times n$  hexagonal arrays of boxes, for  $1 \leq m \leq n \leq 10$ .

► 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  
Reutersvård  
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  $\{1, \dots, 18\}$  for the edge labels, 18 primary items  $\{\text{NH-MA}, \dots, \text{GA-SC}\}$  for the edges, 13 secondary items  $\{\text{NH}, \dots, \text{SC}\}$  for the colonies, and 19 secondary items  $\{h_0, \dots, h_{18}\}$  for the holders of vertex labels. These items are to be governed by  $18 \cdot 19 \cdot 18 = 6156$  options, such as

‘6 PA-DE PA:3 DE:9  $h_3$ :PA  $h_9$ :DE’,

namely one for each edge label  $d$ , each edge, and each way to assign labels  $j$  and  $k$  with  $0 \leq j < k = j + d \leq 18$  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 L0, FIRST, NEXTL, NEXTH, NAME in (35) correspond to the labeling in Fig. 105(b). What arrays L0', ..., NAME' correspond to its complement, Fig. 105(c)?

**71.** [M20] (S. Golomb, 1972.) Complete the proof that  $K_n$  is ungraceful when  $n \geq 5$ .

- **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  $p_1 \dots p_{2m}$  is a graceful permutation of  $\{1, \dots, 2m\}$  with  $p_{2m} = p_1 + m$ , then  $p_{2k} > m$  for  $1 \leq k \leq m$ .
- c) Conversely, if  $p_1 \dots p_{2m}$  is graceful and  $p_{2k} > m$  for  $1 \leq k \leq m$ , 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  $0 \text{ --- } (m - 1)$  and satisfies (40).

**77.** [16] Study Fig. 107. Why doesn't  $\begin{bmatrix} 29 \\ 80 \end{bmatrix}$  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  $3 \times 17$  matrix whose first row contains the first 34 digits of  $\pi$ ?
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  $(0, m-2, 4, m-6, 8, \dots)$ .
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"  $G_m^\pi$  can be based on  $\pi$  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  $\{0, \dots, m\}$ ; the edges are  $0 \text{ --- } m$  and  $a_k \text{ --- } a_k + m - k$ , for  $1 \leq k < m$ .

- Show that these integers  $a_k$  are unique, and compute them for  $k \leq 20$ .
  - How many isolated vertices does  $G_m^\pi$  have, for  $m \leq 20$ ? How many components?
  - Determine the chromatic numbers  $\chi(G_1^\pi), \dots, \chi(G_{20}^\pi)$ .
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  $\leq 16$  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  $\leq 14$  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  $1 \text{ --- } 3$  doesn't touch any of the longer edges  $2 \text{ --- } 5$ ,  $0 \text{ --- } 4$ ,  $0 \text{ --- } 5$ .
- Is the 13-colonies labeling in Fig. 105(b) rooted?
  - How many of the 160488 graceful labelings of that graph are rooted?
  - How many of the  $16!$  labelings in exercise 91 are rooted?
  - 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?

$\pi$   
 KC graph  
 random graceful graph  
 $G_m^\pi$   
 $\pi$   
 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.

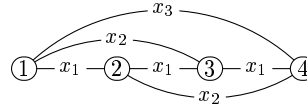
**103.** [24] Find all of the essentially distinct graceful labelings of (41).

**104.** [M30] (D. Anick, 2016.) To backtrack through all graceful labelings of free trees on the vertices  $\{0, \dots, m\}$ , we can successively choose  $L0[k]$  for  $k = m, m-1, \dots, 1$ , in such a way that the edge  $L0[k] \text{ --- } L0[k] + k$  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  $L0[k]$ , where  $t_k$  is suitably large.

At the moment we choose  $L0[k]$ , 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  $0 \leq r < k$ , 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.”

- Show that there's always at least one good residue.
- If there are  $g$  good residues, then  $t_k \geq g$ .
- If there are  $G$  good components, then  $t_k \geq G - g$ .
- If  $k < m/2$ , a bad component contains at least two different bad residues.
- Hence we may let  $t_k = \lfloor (k+4)/3 \rfloor$  when  $k < m/2$ .
- When  $k \geq m/2$  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  $\{1, \dots, n\}$ , assign the weight  $x_d$  to edge  $k \text{ --- } (k+d)$ , for  $1 \leq k \leq n-d$  and  $1 \leq d < n$ , 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  $S(x_1, \dots, x_{n-1})$  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.

- Express  $S(x_1, \dots, x_{n-1})$  as a determinant. *Hint:* See exercise 2.3.4.2–20.
- Explain how to compute  $\tau(n-1) = [x_1 \dots x_{n-1}]S(x_1, \dots, x_{n-1})$  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  $\{0, 1, \dots, 2^n - 1\}$ , rooted at 0, where the parent of node  $x \neq 0$  is node  $x \& (x-1)$ . (See 7.2.1.3–(21).) If  $x = (x_{n-1} \dots x_1 x_0)_2$ , let  $l(x) = (l_{n-1} \dots l_1 l_0)_2$ , where  $l_k = x_0 \oplus \dots \oplus x_k$ . 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  $0 = a_0 < a_1 < a_2 < \dots$ : Let  $g(2k)$  and  $g(2k+1)$  be labeled  $a_k$  and  $m-k-a_k$ , respectively, where  $m = n2^{n-1}$ . For example,

$$\begin{array}{cccccccc} v & = & 000 & 001 & 011 & 010 & 110 & 111 & 101 & 100 \\ l(v) & = & a_0 & 12-a_0 & a_1 & 11-a_1 & a_2 & 10-a_2 & a_3 & 9-a_3 \end{array}$$

when  $n=3$ . (See 7.2.1.1–(4).) Assume that  $a_{2^n+r} = a_{2^n} + a_r$  for  $0 \leq r < 2^n$ .

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  $0 \leq j \leq 1$ . (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  $a_1, a_2, a_4, a_8, \dots$  make the labeling graceful?
- **110.** [M25] A *parallomino graph* (see exercise 7.2.2.1–303) has vertices  $(x, y)$  for integers  $0 \leq x \leq r$  and  $s_x \leq y \leq t_x$ , where  $0 = s_0 \leq s_1 \leq \dots \leq s_r, t_0 \leq t_1 \leq \dots \leq t_r$ , and  $s_{k+1} \leq t_k$  for  $0 \leq k < r$ ; 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:

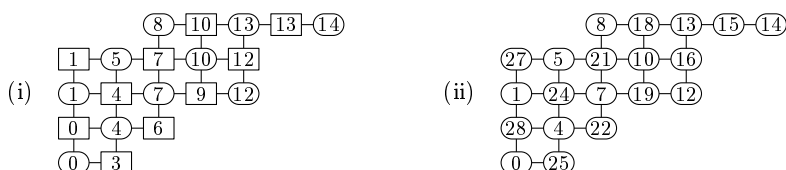


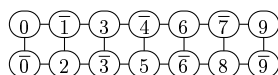
Illustration (ii) is in fact a remarkable *graceful* labeling, where the edges whose labels are 1, 2, ..., 28 appear in strict order, from right to left and top to bottom!

- 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  $\bar{l}$  denote  $m - l$ . A graph is  $\alpha$ -graceful if its edges can be written

$$u_0 \text{ --- } \overline{v_0}, u_1 \text{ --- } \overline{v_1}, \dots, u_{m-1} \text{ --- } \overline{v_{m-1}},$$

$$\text{where } u_k + v_k = k, 0 \leq u_k < l, \text{ and } 0 \leq v_k < m+1-l, \text{ for some } l.$$

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  $\alpha$ -graceful; and a similar construction works for  $K_2 \square P_r$  in general.

- a) Prove that an  $\alpha$ -graceful graph is graceful and bipartite.
- b) For which  $n$  is the cycle  $C_n$   $\alpha$ -graceful?
- c) Prove that every  $\alpha$ -graceful labeling has an “edge complement” in which edge  $k$  becomes edge  $m+1-k$ , for  $1 \leq k \leq m$ .
- d) Find a tree with seven nodes that's not  $\alpha$ -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  $u \text{ --- } v$  with  $u \in U, v \in V$ .
- a) Show that every  $\alpha$ -graceful graph has an ordered graceful labeling.
- b) Show that the non- $\alpha$ -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  $u \text{ --- } v$  between parts  $U$  and  $V$  leads naturally to a bipartite graph  $G^{(t)}$  with  $tm$  edges  $u \text{ --- } v_i$  between parts  $U$  and  $V_1 \cup \dots \cup V_t$ . If  $G$  has an ordered graceful labeling, show that  $G^{(t)}$  does too.
- 113.** [M21] Continuing exercises 111 and 112, how many (a)  $\alpha$ -graceful labelings (b) ordered graceful labelings have  $m$  edges? (Compare with Theorem S.)

parallomino graph

grid

skeleton

$\alpha$ -graceful

bipartite

cycle  $C_n$

complement

*ordered* graceful labeling

near  $\alpha$ -labeling, see ordered graceful labeling

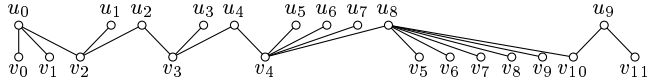
114. [M20] The direct product of bipartite graphs always has at least two components.

- Prove this, by determining the components of  $K_{a,b} \otimes K_{c,d}$ .
- 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.
- If  $G$  and  $H$  each have an ordered graceful labeling, prove that  $(G \otimes H)'$  and  $(G \otimes H)''$  do too.

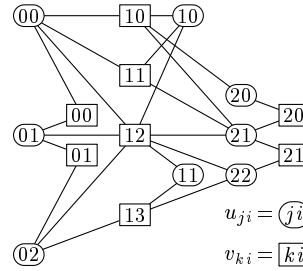
- 115. [M28] (*Caterpillar nets*.) A “caterpillar” is a graph with at least two vertices that becomes a path (or empty) when you remove all of its vertices of degree 1. More precisely, an  $(s, t)$ -caterpillar is a bipartite graph with vertices  $\{u_0, \dots, u_s; v_0, \dots, v_t\}$  and edges defined by a binary vector  $e = e_1 \dots e_{s+t}$  that has  $s$  0s and  $t$  1s:

$$u_{s_i} \text{ --- } v_{t_i} \text{ for } 0 \leq i \leq s+t, \text{ where } s_i = \bar{e}_1 + \dots + \bar{e}_i \text{ and } t_i = e_1 + \dots + e_i.$$

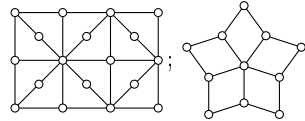
For example, here's the  $(9, 11)$ -caterpillar whose edge vector is 11001001000011111101:



- Draw the eight  $(s, t)$ -caterpillars for which  $s + t = 3$ .
- Prove that every  $(s, t)$ -caterpillar is  $\alpha$ -graceful.
- 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  $U_j = \{u_{j0}, \dots, u_{jp_j}\}$  and  $V_k = \{v_{k0}, \dots, v_{kq_k}\}$ , for  $0 \leq j \leq s$  and  $0 \leq k \leq t$ . The edges are  $(p_{s_i}, q_{t_i})$ -caterpillars between  $U_{s_i}$  and  $V_{t_i}$ , for  $0 \leq i \leq s+t$ . 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?



- Prove that every caterpillar net is  $\alpha$ -graceful.
- Prove that the complete bipartite graph  $K_{n,r}$  is a caterpillar net.
- Prove that the grid  $P_n \square P_r$  is a caterpillar net.
- 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  $\alpha$ -graceful.

117. [HM36] Exercise 115(e) proved that  $K_{n,r}$  is  $\alpha$ -graceful. Let  $A(n, r)$  be the exact number of different  $\alpha$ -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  $0 \text{ --- } \bar{0}, 1 \text{ --- } \bar{0}, 0 \text{ --- } \bar{2}, 1 \text{ --- } \bar{2}$  or  $0 \text{ --- } \bar{0}, 0 \text{ --- } \bar{1}, 2 \text{ --- } \bar{0}, 2 \text{ --- } \bar{1}$  in the notation of exercise 111.)

- Prove that  $A(n, r)$  is the number of ways to write the polynomial  $F_m(x) = 1 + x + \dots + x^{m-1}$  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  $A(p_1 p_2, q_1 q_2)$ , 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  $A(p_1^{e_1} \dots p_s^{e_s}, q_1^{f_1} \dots q_t^{f_t}) = [p_1^{e_1} \dots p_s^{e_s} q_1^{f_1} \dots q_t^{f_t}] 1 / ((1-p_1) \dots (1-p_s) + (1-q_1) \dots (1-q_t) - 1)$ .
- i) In particular,  $A(p^e, q_1^{f_1} \dots q_t^{f_t}) = \binom{e+f_1}{e} \dots \binom{e+f_t}{e}$ .

118. [M22] Show that  $K_{n,r}$  sometimes has graceful labelings that are *not*  $\alpha$ -graceful:

- a) If  $r = 2$  and  $2n+1 = pq$  with  $p, q > 1$ , use labels  $\{2n, 2n-p\}$  in the second part, with labels  $\bigcup_{k=0}^{\lfloor q/2 \rfloor - 1} [2kp \dots 2kp+p]$  and  $\lfloor p/2 \rfloor$  others in the first part.
- b) If  $n = 3k+1$ , use labels  $[0 \dots 2k] \cup [nr-k \dots nr-1]$  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.

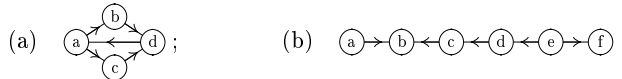
121. [13] For what  $a$  and  $b$  does  $x \mapsto (ax+b) \bmod 13$  take Fig. 109(e) into Fig. 109(f)?

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  $l'(v) = (a(l(v)-b)) \bmod q$  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  $d \not\equiv 0 \pmod q$ .
- 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?



125. [16] Design an algorithm to create the FIRST and NEXT arrays of a graceful digraph, given its L0 array.

► 126. [M25] Let  $l$  be a graceful labeling of  $D$ , and let L0, FIRST, NEXT, and NAME be the corresponding representation as in (44). A labeling  $l'$  equivalent to  $l$  will then correspond to certain arrays L0', FIRST', NEXT', and NAME'. (Compare with exercise 70.)

- a) Compute them when  $l'(v) = (a(l(v)-b)) \bmod q$ , given  $a$  and  $b$  with  $a \perp q$ .
- b) Compute them when  $l'(v) = l(v\alpha)$ , given an automorphism  $\alpha$  of  $D$ .
- c) Compute them when  $l'(v) = l(v\alpha)$ , given an antiautomorphism  $\alpha$  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  
L0  
digraph representation  
automorphism  
antiautomorphism  
set inclusion  
transitive  
Boolean lattice

the arcs  $000 \rightarrow 011$ ,  $000 \rightarrow 101$ ,  $000 \rightarrow 110$ ,  $000 \rightarrow 111$ ,  $001 \rightarrow 111$ ,  $010 \rightarrow 111$ ,  $100 \rightarrow 111$  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  $x = x_1 \dots x_m$  with  $0 \leq x_l \leq m$  for  $1 \leq l \leq m$ . If  $x \in \mathcal{D}_m$ , the digraph  $aD(x) + b$  has  $m+1$  vertices  $\{0, \dots, m\}$  and  $m$  arcs,  $(ax_l + b) \bmod q \rightarrow (ax_l + al + b) \bmod q$ , 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  $a \perp q$ .

- What are the equivalence classes of  $\mathcal{D}_2$  and  $\mathcal{D}_3$ ?
- What's a good way to visit each equivalence class of  $\mathcal{D}_m$ , when  $m$  isn't too large?
- 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  $k$ th vertex label in a path or cycle  $v_0 \rightarrow \dots \rightarrow v_m$ .

- Show that  $l_{2k} = r - 1 - k$  and  $l_{2k+1} = r + k$  gracefully label the oriented path  $P_{2r}^+$ .
- Find a somewhat similar pattern of graceful labels for  $C_{2r}^+$ . *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  $D \rightarrow \overline{K_n}$  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  $D \rightarrow \overline{K_n}$  is graceful for all  $n > 0$ .

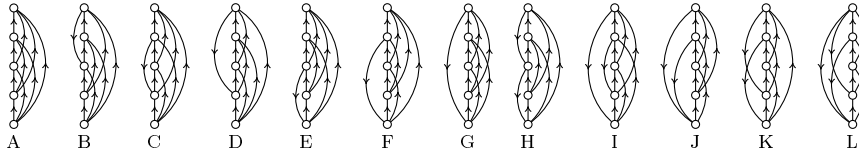
**133.** [16] Is the oriented complete bipartite graph  $K_{m,n}^+ = \overline{K_m} \rightarrow \overline{K_n}$  graceful?

**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  $u \rightarrow v$  or  $v \rightarrow u$  for every pair of vertices  $u$  and  $v$  (see exercise 7–59). There are twelve unlabeled tournaments of order 5:



- What are the *converses* of A, B, ..., L? (For example,  $A^T = A$ .)
- How many essentially distinct graceful labelings does each of them have?
- What are the graceful tournaments of orders 3 and 4?
- A *cyclic  $(v, k, \lambda)$ -difference set* is a set  $\{a_1, \dots, a_k\} \subseteq \{0, 1, \dots, v-1\}$  such that the  $k(k-1)$  differences  $(a_j - a_k) \bmod v$  for  $j \neq k$  contain each nonzero residue exactly  $\lambda$  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.

- Show that  $\{1, 7, 7^2 \bmod 37, \dots, 7^8 \bmod 37\}$  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  $(v, k, \lambda)$ -difference set

- 138.** [46] An undirected graph with  $m$  edges can be converted to a directed graph in  $3^m$  ways, because each edge  $u - v$  can become  $u \rightarrow v$  or  $u \leftarrow v$  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  $F$ .
- Explain why this construction gives  $n^2 + n + 1$  points and  $n^2 + n + 1$  lines.
  - Which points belong to the line  $[1, 0, 2]$  when  $n = 3$ ?
  - Why do two lines intersect in a unique point?
- **145.** [HM27] (J. Singer, 1938.) Suppose  $K_{n+1}$  has graceful rainbow labels  $\{l_0, \dots, l_n\}$ .
- Show that they’re a cyclic  $(n^2 + n + 1, n + 1, 1)$ -difference set (see exercise 137(d)).
  - 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–(g)). Consequently the nonzero elements of  $F$  are  $\{1, \pi, \pi^2, \dots, \pi^{p^3-2}\}$ , where  $\pi$  is a root of  $f$  in  $F$ . What are the other two roots of  $f$ ? *Hint:*  $(x + y)^p \equiv x^p + y^p \pmod{p}$ .
  - 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  $\pi^{k+1} = a'_1\pi^2 + a'_2\pi + a'_3$ .
  - 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  $a_1b_1 + a_2b_2 + a_3b_3 = a'_1b'_1 + a'_2b'_2 + a'_3b'_3$ .
  - 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  $0 \leq k < p^3 - 1$ . List those triples in the special case when  $p = 5$  and  $f(x) = x^3 - 4x^2 - 3$ . (You can stop at  $k = 31$ .)
  - 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  $0 \leq k < p^2 + p + 1$ . Furthermore,  $L = \{k \mid a_{k1} = 0 \text{ and } 0 \leq k < p^2 + p + 1\}$  is a set of graceful rainbow labels for  $K_{p+1}$ .
  - 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  $x = x_1 \dots x_m$  with  $0 \leq x_l \leq 2m$  for  $1 \leq l \leq m$ . If  $x \in \mathcal{R}_m$ , the graph  $aG(x) + b$  has  $2m + 1$  vertices  $\{0, \dots, 2m\}$  and  $m$  edges,  $(ax_l + b) \bmod q \text{ --- } (ax_l + al + b) \bmod q$ , 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  $a \perp q$ .
- What are the equivalence classes of  $\mathcal{R}_2$  and  $\mathcal{R}_3$ ? (Compare with exercise 129.)
  - What’s a good way to visit each equivalence class of  $\mathcal{R}_m$ , when  $m$  isn’t too large?
  - 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  $n_1 \dots n_r$  vertices  $x_1 x_2 \dots x_r$ , where  $0 \leq x_k < n_k$  for  $1 \leq k \leq r$ ; and we have  $x_1 x_2 \dots x_r \sim y_1 y_2 \dots y_r$  if and only if  $x_k \neq y_k$  for exactly one index  $k$ .

- How many edges does  $K_{n_1} \square K_{n_2} \square \cdots \square K_{n_r}$  have?
- How many automorphisms does  $K_{n_1} \square K_{n_2} \square \cdots \square K_{n_r}$  have?
- Compute the distance between 141421 and 271828 in a Hamming graph.
- 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  $K_{n_k}$ .
- What 4-vertex graph  $G$  can't be strictly embedded in a Hamming graph?
- 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  $\{v_0, v_1, \dots, v_{n-1}\}$ , a Hamming embedding of  $G$  is a function  $f(v_i) = x_{i1} \dots x_{ir}$  with the property that, for  $0 \leq i < j < n$ , we have  $x_{i1} \dots x_{ir} \sim x_{j1} \dots x_{jr}$  in a Hamming graph if and only if  $v_i \sim v_j$  in  $G$ .

- Assume that  $G$  is connected, and that each vertex  $v_i$  for  $i > 0$  has a “parent vertex”  $v_{i'}$  with  $i' < i$  and  $v_{i'} \sim v_i$ . Show that every Hamming embedding of  $G$  can be “normalized” so that (i)  $x_{0k} x_{1k} \dots x_{(n-1)k}$  is a restricted growth string, as defined in 7.2.1.5–(4), for  $1 \leq k \leq r$ ; 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  $x_{01} x_{02} \dots x_{0r} = 00 \dots 0$  and  $x_{11} x_{12} \dots x_{1r} = 10 \dots 0$ .)
- 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  $G' \neq G$ .

- Is  $G$  Hamming embeddable if and only if it has no induced MNH subgraph?
- Prove that an MNH subgraph is connected.
- 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  $(u \sim v) \bowtie (u' \sim v')$  be the relation  $d(u, u') - d(u, v') \neq d(v, u') - d(v, v')$ , when  $u \sim v$  and  $u' \sim v'$  are edges of a graph and  $d(u, v)$  denotes shortest distance in that graph.

- 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,  $(u - v) \bowtie (u' - v') \iff \{u, v\} \cap \{u', v'\} \neq \emptyset$ .
- 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  $1 \leq n \leq 9$ , 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  $****$ .

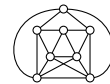
It’s easy to work with subcubes inside a computer, using the asterisks-and-bits representation of exercise 7.1.1–30. For example,  $0*1*$  is represented by the two bitstrings  $a = 0101$  and  $b = 0010$ , showing respectively the \*s and the 1s.

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  $0 - 1 - 2 - 3 - 4 - 0$  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  $2 - 3$ , 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, *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  $G \subseteq G''$ .      v)  $G \sqsubseteq G' \subseteq G''$  implies  $G \subseteq G''$ .
- ii)  $G \subseteq G' \subseteq G''$  implies  $G \sqsubseteq G''$ .      vi)  $G \sqsubseteq G' \subseteq G''$  implies  $G \sqsubseteq G''$ .
- iii)  $G \subseteq G' \sqsubseteq G''$  implies  $G \subseteq G''$ .      vii)  $G \sqsubseteq G' \sqsubseteq G''$  implies  $G \subseteq G''$ .
- iv)  $G \subseteq G' \sqsubseteq G''$  implies  $G \sqsubseteq G''$ .      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  $A = \{a_1, \dots, a_t\}$  is a multiset of positive integers. Let  $T$  be the tree with root  $r$  and  $mn+m$  additional vertices  $x_{jk}$  for  $1 \leq j \leq m$ ,  $0 \leq k \leq n$ , whose edges are  $r \text{ --- } x_{j0}$  and  $x_{jk} \text{ --- } x_{j(k+1)}$ . Prove that  $G \subseteq T$  if and only if  $A$  can be partitioned into  $m$  multisets whose sums are each  $\leq n$ . (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  $0 \leq k < 5$ , and whose edges  $(v, k) \text{ --- } (v', k')$  are of three kinds: (i)  $v = v'$  and  $\{k, k'\} \in \{\{0, 1\}, \{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 0\}, \{2, 4\}\}$ ; (ii)  $v \text{ --- } v'$  and  $\{k, k'\} = \{0, 1\}$ ; (iii)  $v \neq v'$ ,  $v \not\text{---} v'$ , 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  $v \neq w$  then  $f(v) \neq f(w)$ . (ii) If  $v \text{ --- } w$  in  $G$  then  $f(v) \text{ --- } f(w)$  in  $H$ . (iii)  $l_i(v)$  is compatible with  $l_i(f(v))$ , for all  $i$ . (iv) If  $v \text{ --- } w$  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  $\leq 3$ . *Hint:* Reduce from 3SAT.

**168.** [20] If  $G$  is a graph with  $n$  vertices and  $m$  edges, let  $\hat{G}$  be the directed acyclic graph with  $m+n$  vertices and  $2m$  arcs obtained by replacing each edge  $u \text{ --- } v$  by  $u \rightarrow uv \leftarrow v$ . Prove or disprove: (a)  $G \subseteq H \iff \hat{G} \subseteq \hat{H}$ ; (b)  $G \sqsubseteq H \iff \hat{G} \sqsubseteq \hat{H}$ .

**169.** [21] Given an integer  $M \geq 3$  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  $\mathcal{G}_{\lfloor 21g n + 2 + \delta \rfloor} \not\sqsubseteq \mathcal{G}_n$  a.s., for fixed  $\delta > 0$  as  $n \rightarrow \infty$ .
- b) Prove that  $\mathcal{G}_{\lceil 21g n - \delta \rceil} \sqsubseteq \mathcal{G}_n$  a.s., for fixed  $\delta > 0$  as  $n \rightarrow \infty$ .

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



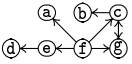
- **173.** [21] The reduced target graph  $\hat{H}$  obtained from BRAIN83(250) has 11 vertices in the left brain and 11 vertices in the right brain, with only two edges between them. Why does that make  $G \subseteq \hat{H}$  impossible, when  $G$  is Chvátal's graph (52)?
- 174.** [23] Embed Chvátal's 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  $\{0, 0+, 1-, 1, 1+, \dots, ((k/3) - 1)+, 0-\}$  and  $2k$  edges  $j - (j + 1)$ ,  $j - j+$ ,  $j - j-$ ,  $j+ - (j + 1)-$ ,  $j+ - (j + k/6)+$ ,  $j+ - (j + k/6)-$  (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  $k$ th largest element is greater than or equal to  $B$ 's  $k$ th largest element, for  $1 \leq k \leq |B| \leq |A|$ .
- Given a vertex  $v$  of a graph  $G$ , let  $s(v) = \{\deg(u) \mid u - v\}$  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  $G$ .
  - 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  $02 \mapsto \text{LA}$ , 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)  $P_2 \sqcap P_n \subseteq \text{USA}$ ? (b)  $P_3 \sqcap P_n \subseteq \text{USA}$ ?
- 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  $x_1, \dots, x_m$  are subject to an all-different constraint, the domains  $D_1, \dots, D_m \subseteq \{1, \dots, n\}$  are said to be *feasible* if there's a matching of size  $m$  in the bipartite graph on vertices  $\{x_1, \dots, x_m\}$  and  $\{y_1, \dots, y_n\}$  whose edges are  $x_i - y_j$  when  $j \in D_i$ . A value  $j \in D_i$  is said to be *removable* if  $x_i - y_j$  isn't in any feasible matching.
- Let  $x_1 - y_{j_1}, \dots, x_m - y_{j_m}$  be a matching, and construct the following tripartite digraph  $T$  on  $\{x_1, \dots, x_m\}$ ,  $\{y_1, \dots, y_n\}$ , and  $\{\perp\}$ :  $x_i \rightarrow y_{j_i}$  and  $y_{j_i} \rightarrow \perp$ , for  $1 \leq i \leq m$ ;  $x_i \leftarrow y_j$ , if  $j \in D_i$  and  $j \neq j_i$ , for  $1 \leq i \leq m$ ;  $y_j \leftarrow \perp$ , if  $j \notin \{j_1, \dots, j_m\}$ . 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.
- If  $I \subseteq \{1, \dots, m\}$ , 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).
  - 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.
  - 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

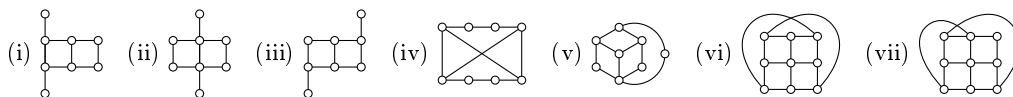
- d) If  $I$  is a Hall set, explain why we can remove  $D(I)$  from all domains  $D_j$  for  $j \notin I$ .
- e) Prove that Hall sets of feasible domains are closed under union and intersection.
- f) Prove that a feasible family of domains has no removable elements if and only if there's a partition of  $\{1, \dots, m\}$  into disjoint sets  $I_0, I_1, \dots, I_r$  with disjoint domains  $D(I_0), D(I_1), \dots, D(I_r)$  such that the Hall sets are precisely the  $2^r$  sets obtainable by unions of  $\{I_1, \dots, I_r\}$ . (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  $x_1 \dots x_n$  will be a *permutation* of  $\{1, \dots, n\}$ . 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?

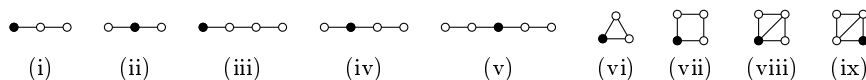


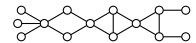
**191.** [15] True or false:  $G \sqsubseteq H$  implies  $G^{\leq 2} \sqsubseteq H^{\leq 2}$ .

**192.** [20] Compute the vertex degrees of  $G^{\leq 2}$  and  $H^{\leq 2}$  when  $G = P_4 \square P_5$  and  $H = \text{USA}$ . 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  $G^{\leq 2}$  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  $s = \bullet$ :



Compute the supplemental vertex labels  $d_G^S(v)$ , for each  $v \in G =$  .

**195.** [21] Compute supplemental edge labels for each edge  $u \rightarrow v$  of that same graph, using each of the motifs  $S = \bullet \blacksquare \circ, \bullet \blacksquare \circ \circ, \circ \bullet \blacksquare \circ$ . (Here  $\bullet = s, \blacksquare = t$ .)

**196.** [20] Draw the supplemental graphs  $G^{S,k}$ , for the graph  $G$  of exercise 194, when (i)  $S = \bullet \blacksquare \circ$  and  $k = 1$ ; (ii)  $S = \bullet \blacksquare \circ \circ$  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  $11 \mapsto MO$ .

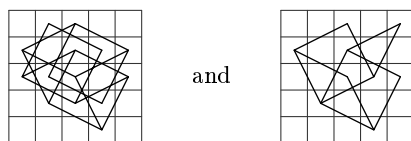
► **198.** [24] Using the supplemental graph  $G^{S,2}$ , where  $S = P_2 \square P_3$  and its vertices of degree 3 are  $s$  and  $t$ , show that the initial domains for all six interior vertices of  $P_4 \square P_5$  in the USA problem can be reduced to size 13 — less than half of what we had without 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  $G^\Sigma$  such that  $G \sqsubseteq H$  implies  $d_G(v) \leq d_H(f(v))$ ,  $\ell_G(v, w) \leq \ell_H(f(v), f(w))$ , 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*.
- Show that the pattern vertices can be prearranged into a fixed (static) order  $p_1 p_2 \dots p_m$  so that, at level  $l$  of the search, vertices  $\{p_1, \dots, p_l\}$  have been assigned and  $\{p_{l+1}, \dots, p_{r_l}\}$  are near, for some  $r_l \geq l$ . Furthermore  $r_l > l$  for  $0 < l < m$  if and only if the pattern is connected.
  - Explain how to maintain a permutation  $t_1 t_2 \dots t_n$  of the target vertices (dynamically) so that, at level  $l$  of the search, the current assignments are  $t_j = f(p_j)$  for  $1 \leq j \leq l$ , and the vertices  $\{t_{l+1}, \dots, t_{s_l}\}$  are near, for some  $s_l \geq r_l$ .
  - If  $p_{l+1}$  has  $q$  near neighbors, must  $f(p_{l+1})$  have at least  $q$  near neighbors?
  - If  $p_{l+1}$  has  $q$  far neighbors, must  $f(p_{l+1})$  have at least  $q$  far neighbors?
- 202.** [20] Let  $D_1, \dots, D_m$  be domains  $\subseteq \{1, \dots, n\}$ , with  $|D_1| \leq \dots \leq |D_m|$ . 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  $1 \leq j \leq m$ : 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  $n \leq 64$  and that each domain is represented bitwise. Process the domains in order of increasing size, *without* assuming that  $|D_1| \leq \dots \leq |D_m|$ , 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  $P_2 \sqcap P_7$  and  $P_3 \sqcap P_3$  can be seen as knight moves


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



within a  $5 \times 5$  board; in other words,  $P_2 \sqcap P_7 \subseteq N_5$  and  $P_3 \sqcap P_3 \subseteq N_5$ , where  $N_n$  is the  $n \times n$  knight graph. (This scenario generalizes the classic notion of a “knight's tour.”)

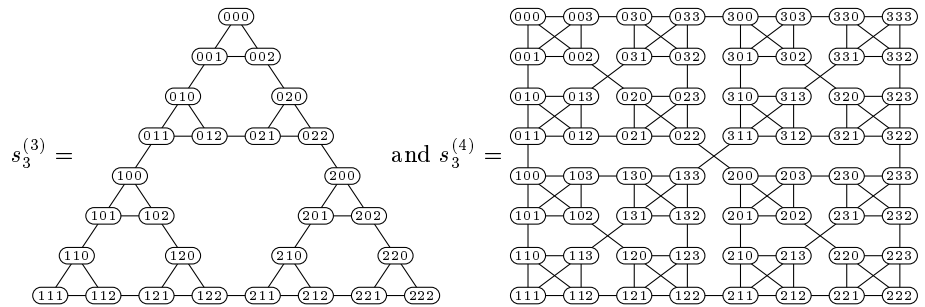
- Find the largest  $n$  with  $P_m \sqcap P_n \subseteq N_8$  when  $m = 2, 3, 4, 5, 6$ .
  - Find the largest  $n$  with  $P_m \sqcap P_n \subseteq N_8$  when  $m = 2, 3, 4, 5, 6$ .
  - Find the largest  $n$  with  $P_2 \sqcap C_n \subseteq N_8$ .
  - Find the largest  $n$  with  $P_2 \sqcap C_n \subseteq N_8$ .
  - Find the largest  $n$  with  $P_3 \sqcap C_n \subseteq N_8$ .
  - Find the largest  $n$  with  $P_3 \sqcap P_3 \sqcap P_n \subseteq N_8$ .
- 206.** [40] Continuing exercise 205, let  $f_m(t)$  be the largest  $n$  such that  $P_m \sqcap P_n \subseteq N_t$ , and let  $\bar{f}_m(t)$  be the largest  $n$  such that  $P_m \sqcap P_n \subseteq N_t$ . Compute  $f_m(t)$  and  $\bar{f}_m(t)$  for as



- d) Let  $\text{sol}[r][e] = [S_r \subseteq T_e \text{ with } r \mapsto \text{root}(T_e)]$ , for  $0 < r < m$  and  $0 \leq e < 2n - 2$ . 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  $\text{root}(T_e) = v$  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  $\text{sol}$  matrix for problem (a)?
- 214.** [29] Evaluate Matula’s algorithm (exercise 213) empirically by applying it to several classes of free trees:
- Let  $S$  run through all 551 free trees with  $m = 12$ , and let  $T$  run through all 19320 free trees with  $n = 16$ .
  - Let  $S$  and  $T$  be uniformly random free trees with  $m = 25$  and  $n = 1000$ .
  - 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  $\{0, 1, \dots, d - 1\}$ . (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  $k = (0, 1, 2)$ .
- Is this the same as Table 2’s “prefix encoding”?
  - What’s the prefix code for the extended binary tree ?
  - 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  $a_1 \dots a_{n-1}$  and its vertices, assuming that vertex  $12 \dots 2 = 21 \dots 1 \mapsto (0, 0, 0)$ . What are the coordinates of  $0 \dots 0$ ,  $1 \dots 1$ , and  $2 \dots 2$ ? *Hint:* Show that every odd number between  $-2^n$  and  $+2^n$  has a unique *binary representation*  $(b_1 \dots b_n)_2$  in which every digit  $b_j$  is  $\pm 1$ .
- 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  $\hat{S}_n^{(3)}$  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  $\hat{S}_n^{(3)}$  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  $\hat{S}_4^{(3)}$  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  $\hat{S}_n^{(3)}$  before it becomes 3-colorable?
- **228.** [21] What clique hints, analogous to (6g), 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  $\hat{S}_4^{(3)}$ ? (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  $\hat{S}_n^{(3)}$  cannot be 3-colored?
- 231.** [M46] Can an exponential lower bound be proved on the refutation length of the clauses for 3-uncolorability of  $\hat{S}_n^{(3)}$ ? (See Theorem 7.2.2.2B.)
- 232.** [24] Repeat exercise 225, but test flower snark line graphs  $L(J_q)$  instead of  $\hat{S}_n^{(3)}$ .
- 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  $d \leq d'$ .
- 237.** [16] Almost every vertex of  $S_n^{(d)}$ ,  $\hat{S}_n^{(d)}$ , 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



are different from but strongly related to the Sierpiński simplex graphs  $S_n^{(d)}$ . In general,  $s_n^{(d)}$  has  $d^n$  vertices  $a_1 \dots a_n$ , for  $0 \leq a_j < d$ , and two kinds of edges:

- clique edges  $a_1 \dots a_{n-1}j — a_1 \dots a_{n-1}k$ , for  $0 \leq j < k < d$ ;
- nonclique edges  $a_1 \dots a_i j k \dots k — a_1 \dots a_i k j \dots j$ , for all  $0 \leq i < n - 1$  and  $0 \leq j < k < d$ .

Notice that almost every vertex has degree  $d$ ; this property is akin to exercise 237.

- 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  $8 \times 8$  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  $0 \dots 00$ ,  $0 \dots 01$ ,  $0 \dots 02$ ,  $0 \dots 03$  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  $0 \leq i, j < n$ , each with domain  $\{0, 1, \dots, n-1\}$ , 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 ' $u \leq v - t$ ', when variables  $u$  and  $v$  are represented with the log encoding and  $t$  is constant. Illustrate your construction in the cases ' $u \leq v + 1$ ' and ' $u \leq v - 2$ ', assuming that  $u = (u_8 u_4 u_2 u_1)_2$  and  $v = (v_8 v_4 v_2 v_1)_2$ .

**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  $\overline{R}_{001}$ ,  $\overline{R}_{010}$ ,  $\dots$ ,  $\overline{R}_{211}$  from (80)–(82) and  $(R_{000})$  by unit propagation.

**253.** [M16] Let  $R(v_1, \dots, v_k)$  be a  $k$ -ary relation, where variable  $v_j$  has domain  $[0 \dots d_j]$  for  $1 \leq j \leq k$ . 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  $R(v_1, \dots, v_k)$ , prove:
- There will be no contradiction.
  - If no clauses remain,  $R$  is satisfied by the true literals  $v_{1a_1}, \dots, v_{ka_k}$ .
  - Otherwise the remaining clauses are the support encoding for some relation  $R'$ .
  - If literal  $v_a$  remains, there's a solution with  $v_a$  true and another with  $v_a$  false.
  - If literal  $v_a$  remains, statements (a), (b), and (c) hold also for the clauses  $S \wedge (v_a)$ .
  - 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.

- The domain of every inactive variable in a partially solved CSP has size 1.
- The domain of every active variable in a partially solved CSP has size  $> 1$ .
- The domain of every active variable in a partially solved CSP has size  $> 0$ , if we have forward consistency.
- Same as (c), but with domain consistency instead of forward consistency.
- 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  $D_w = D_x = D_y = \{1\}$ ;  $D_z = \{0, 1, 2, 3\}$ :

- If  $w$ ,  $x$ ,  $y$ , and  $z$  are active, there is forward consistency.
- If  $w$  is inactive, but  $x$ ,  $y$ , and  $z$  are active, there is forward consistency.
- If  $w$  and  $x$  are inactive, but  $y$  and  $z$  are active, there is forward consistency.
- If  $w$ ,  $x$ , and  $y$  are inactive, but  $z$  is active, there is forward consistency.
- 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  $16 \times 16$  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  $16 \times 16$  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  $k_1, \dots, k_m$ , is equivalent to a CSP  $\mathcal{P}^*$  with  $m + n$  variables and  $k_1 + \dots + k_m$  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  $\{r_1, \dots, r_5\}$ , 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  $0 \leq i < d$  do this: For  $0 \leq j < d'$ , 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  $0 \leq j < d'$  with  $s'_j = 0$  do this: For  $0 \leq i < d$  with  $m_{ij} = 0$ , if  $r_{ij} = 1$ , set  $s'_j \leftarrow 1$ , and break out of the loop on  $i$ ."

- 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?
- Improve Pass 1. *Hint:* We can often avoid looking at  $r_{ij}$  if we know that  $s'_j = 1$ .
- 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  $x_1, \dots, x_n$ , 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, ' $x_i \neq x_n$ ' for  $1 \leq i \leq m$ ; also  $n - m$  ternary constraints,

$$\text{'either } x_i = x_{j(i)} \text{ or } x_i = x_{k(i)} \text{' for } m < i \leq n,$$

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.)

- Explain why every chain CSP is unsatisfiable.
- Express any given chain CSP as an XCC problem with  $\leq 15n$  options.
- Exactly how many chain CSPs are possible, given  $m$  and  $n$  with  $1 \leq m \leq n$ ?
- Experiment with XCC solvers on uniformly random chain CSPs that have been formulated as in (b), when  $m = 24$  and  $n$  varies.
- 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.

- 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  $\{j(i+1), k(i+1), \dots, j(n), k(n)\}$ .) Find a simple formula for  $S_{m,n}$ .
- 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  $C_{24,64}$ ?
- 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  $d \geq n - 1$  and  $n \rightarrow \infty$ , using MRV and assuming that filtering is done by maintaining (a) forward consistency (only); (b) domain consistency.

**284.** [M20] Exactly how many permutations of  $\{1, 2, \dots, n\}$  have  $p_{j+1} < p_j + d$ , for  $1 \leq j < n$ , given a number  $d$  with  $1 \leq d \leq n$ ?

**285.** [M21] For every subset  $S \subseteq \{1, \dots, n - 1\}$ , prove that exactly one slow growth permutation of  $\{1, 2, \dots, n\}$  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  $\{1, 2, \dots, n\}$ . 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.

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.

$$\begin{array}{ccc} \begin{array}{c} \square 14 \square \\ 2 \square \square \square \\ 1 \square \square 2 \\ \square 3 \square \square \end{array} & ; & \begin{array}{c} 2 \ 1 \ 4 \square \\ 2 \square \square \square \\ 1 \square \square 2 \\ 3 \ 3 \square \square \end{array} & ; & \begin{array}{c} 2 \ 1 \ 4 \square \\ 2 \square \square \square \\ 1 \square \square 2 \\ 3 \ 3 \square \square \end{array} . \end{array}$$

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  $n = 1, 2, 3, \dots$ , 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:

$$\begin{array}{ccccccc} \begin{array}{c} 2 \square 1 \square 2 \square \\ 1 \square \square \square 12 \\ \square 1 \square 3 \square \square \\ \square \square \square \square \square \square \end{array} ; & \begin{array}{c} 3 \ 3 \square \square \square \square \square \square \\ 3 \ 1 \ 4 \ 1 \ 5 \ 9 \ 2 \ 6 \square \\ \square \square \square \square \square \square \square \square \\ \square \square \square \square \square \square \square \square \end{array} ; & \begin{array}{c} \square \square \square \square \square 2 \square \square \square \square \\ \square 2 \ 4 \square \square 8 \square 2 \ 4 \square \\ \square 6 \ 8 \square \square 6 \square 6 \ 8 \square \\ \square \square \square \square \square 4 \square \square \square \square \end{array} ; & \begin{array}{c} 1 \square 3 \ 4 \ 1 \ 4 \ 1 \ 2 \square \square \\ \square \square \square \square \square \square \square \square 4 \\ 2 \square \square 3 \square 2 \square \square 3 \square \\ \square 2 \square 3 \square 3 \square 1 \square \end{array} ; & \begin{array}{c} \square 2 \ 4 \square \square \square \square \square \square \\ \square 1 \ 5 \square \square \square \square \square 1 \ 3 \\ \square \square \square 3 \ 6 \square \square \square 4 \ 5 \\ \square \square \square 4 \ 6 \square \square 6 \ 3 \square \square \end{array} . \\ \text{(a)} \quad \begin{array}{c} \square \square \square \square \square \square \\ \square \square 3 \square 1 \square \\ 2 \ 1 \square \square \square 1 \\ \square 2 \square 1 \square 2 \end{array} ; & \text{(b)} \quad \begin{array}{c} 5 \ 3 \ 5 \ 8 \ 9 \ 7 \ 9 \ 3 \ 2 \\ \square \square \square \square \square \square \square \square \\ \square \square \square \square \square \square \square \square \\ \square 3 \ 8 \ 4 \ 6 \ 2 \ 6 \ 4 \ 3 \\ \square \square \square \square \square \square \square 4 \ 4 \end{array} ; & \text{(c)} \quad \begin{array}{c} 2 \ 6 \ 8 \ 4 \square \square \square \square \square \square \\ \square \square \square \square \square 6 \ 4 \ 8 \ 2 \\ \square \square \square \square 4 \square \square \square \square \\ \square 2 \ 4 \square 8 \square \square 2 \ 4 \square \\ \square 6 \ 8 \square \square 6 \ 8 \square \\ \square \square \square \square 2 \square \square \square \square \square \square \end{array} ; & \text{(d)} \quad \begin{array}{c} \square 1 \square \square \square \square \square \square 2 \\ 2 \square \square \square \square \square \square 4 \square \\ \square 4 \square 4 \square 3 \square 2 \square \\ \square 3 \square \square 1 \square 3 \square \square 1 \\ 1 \square \square \square \square \square \square \square \square \\ \square \square 3 \ 4 \ 2 \ 2 \ 4 \ 2 \square 3 \end{array} ; & \text{(e)} \quad \begin{array}{c} \square \square \square \square \square \square 3 \ 1 \square \square \\ \square \square 6 \ 5 \square \square \square \square \square \square \\ \square \square 5 \ 2 \square 4 \ 2 \square \square \square \\ 3 \ 4 \square \square \square 1 \ 3 \square \square \square \\ 2 \ 3 \square \square \square \square 4 \ 1 \square \\ \square \square \square \square \square \square \square 2 \ 4 \square \end{array} . \end{array}$$

- 293.** [24] There are 59,951  $4 \times 4$  fillomino patterns  $\Phi$  whose labels don’t exceed 5. Exhaustively study them all, finding every valid puzzle without redundant clues for which  $\Phi$  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  $\Phi$ , and let  $\delta_d = \limsup_{n \rightarrow \infty} \#_d(\Phi_n)/n^2$  be the maximum density of  $d$ ’s in any infinite sequence of  $n \times n$  patterns  $\Phi_n$ . Determine  $\delta_d$  for as many  $d$  as you can, and show that  $\delta_d = 1 - \Theta(1/\sqrt{d})$  as  $d \rightarrow \infty$ .

- 297.** [41] Construct a valid  $9 \times 9$  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  $2 \times 3$  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,  $\begin{pmatrix} x & x \\ 8 & 3 \end{pmatrix}$  has the same effect as  $\begin{pmatrix} x \\ 8 \end{pmatrix}$ . One idea for avoiding them is to set  $\sigma \leftarrow \sigma - 1$  when backtracking, instead of advancing  $\sigma$  twice per node by always setting  $\sigma \leftarrow \sigma + 1$ .

a) Demolish that idea.

b) Find a correct way to do stamping that advances  $\sigma$  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  $\text{PREV}(a)$  and  $\text{NEXT}(a)$  don't change when  $a$  is deleted). Show that if  $0 \leq a < a' < d$  and  $a' \in D_v$ , then  $\text{NEXT}(a) \leq a'$  (even when  $a \notin D_v$ ).

**301.** [18] Explain in detail the representation of the initial domain  $\{0, 1, \dots, d-1\}$ , 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  $(b_{\lceil d/e \rceil} \dots b_1 b_0)_{2^e} = \sum \{2^a \mid a \in D\}$ .

- **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  $D' \subseteq \{0, 1, \dots, d-1\}$  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  $R(v_1, \dots, v_k)$  is a relation whose variables  $v_j$  each have a  $d$ -ary domain  $D_j$ , with a sparse-set representation  $\text{DOM}_j$ ,  $\text{IDOM}_j$ ,  $\text{SIZE}_j$ , 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  $\text{OSIZE}_j$  be the value of  $\text{SIZE}_j$  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  $x_1, x_2, \dots$ , in that order. Step  $t$  of the search process begins at level  $l = l_t$ , at which time we've made certain provisional assignments  $x_1 \leftarrow a_1, \dots, x_l \leftarrow a_l$  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

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  $s = s_t \leq l$  for which the previous backward step at level  $l$  dealt with exactly the same assignments  $x_j \leftarrow a_j$  for  $1 \leq j \leq s$ . Thus we already “know” the results of all tests on  $s$ -ary relations between  $a_1, \dots, a_{s-1}$  and  $a_{t+1}$ , and we could have saved that information in an auxiliary array.

- 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  $0 \leq t < 30$  correspond to the sequence ‘ $((())((()))((()()))())((()()))$ ’? (Use  $s = 0$  before backward steps.)
- Devise a way to calculate  $s_0, s_1, \dots$  from a given level sequence  $l_0, l_1, \dots$ . *Hint:* Maintain a sequence of intervals  $[p_0 \dots q_0], [p_1 \dots q_1], \dots, [p_r \dots q_r]$ , where  $0 = p_0 < p_1 < \dots < p_r$ , such that  $s_t = p_k$  when  $k$  is maximum with  $p_k \leq l_t \leq q_k$ .
- Show that the  $s$  values can indeed be rather complicated, by constructing a level sequence  $l_0, l_1, \dots$  for which the intervals in the preceding hint are

 $[0 \dots \infty], [2 \dots 8], [4 \dots 6], [5 \dots 5], [10 \dots 15], [11 \dots 12], [14 \dots 14].$ 

- Find levels  $0 = l_0, l_1, \dots, l_{30} = 0$  for which  $s_0 + s_1 + \dots + s_{29} \geq 107$ .
- What's the average of  $s_0 + \dots + s_{29}$  over *all* level sequences  $0 = l_0, \dots, l_{30} = 0$ ?
- 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  $0 \leq t < 2000000$ . *Hint:* See Algorithm 7.2.1.6W.
- Let  $D_j = \{1, \dots, d_j\}$  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  $\text{CLR}(x)$  in Table 3.

**308.** [10] What node in Table 7.2.2.1–2 corresponds to node  $x$  in Table 3, for  $0 < x < 19$ ?

**309.** [20] True or false:  $\text{ITM}(x) < \text{SECOND}$  if and only if  $\text{LOC}(x) < \text{SECOND}$ .

► **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  $\text{SIZE}(i)$  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  $\text{hide}(i, c)$  will set  $\text{FLAG} \leftarrow 1$  if  $\text{FLAG} = 0$ .

► **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  $\text{SIZE}(i) = 1$  or no other choices exist. (This tactic is called the “sharp preference” heuristic.)

**317.** [22] Why doesn't step C7 hide  $i'$  when  $i' > \text{SECOND}$  and  $\text{POS}(i') > \text{OACTIVE}$ ?

nested parentheses  
 average  
 ACTIVE  
 minimum remaining values  
 MRV  
 hide  
 sharp preference

- **318.** [33] (C. Solmon.) Upgrade Algorithm C to Algorithm C<sup>+</sup> by treating cases with  $\text{SIZE}(i) = 1$  more efficiently. *Hints:* Maintain a list of all such active primary items. Step C5 is unnecessary when  $\text{SIZE}(i) = 1$ , 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  $o_1, \dots, o_d$ , where  $d = \text{SIZE}(i) > 1$ . 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<sup>+</sup> 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  $h(i) = \text{SIZE}(i)$  is used in Algorithm B, the running time doesn't actually match the speed of Algorithm C<sup>+</sup>. 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<sup>+</sup> (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  $\text{MARK}(i)$  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  $O_{-1}$ ,  $O_0^{\text{init}}$ , 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  $\text{opt\_out}(o)$  at a time when  $o \in Q$ ?

Solmon  
 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)  
 $\text{opt\_out}(o)$

**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  $\text{TRIG}(o) = \text{CLR}(o)$ ,  $\text{FIX}(o) = \text{XTRA}(o)$ , and  $\text{AGE}(o) = \text{XTRA}(o + 1)$ , when  $o$  is the spacer node preceding an option. The first item of an option,  $\text{ITM}(o + 1)$ , 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  $\text{TRIG}(0) = 1$ ,  $\text{TRIG}(5) = 3$ ,  $\text{TRIG}(13) = 7$ ,  $\text{TRIG}(19) = 11$ , and the following **POOL**:

$p$ :	1	2	3	4	5	6	7	8	9	10	11	12	13
$\text{INFO}(p)$ :	19	4	13	4	13	17	5	11	19	11	0	17	—
$\text{LINK}(p)$ :	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  $\text{QF} = \text{QR}$ . (The **POOL** above can be accompanied by  $\text{QF} = \text{QR} = 13$ .) The insertion operation ‘ $o \Rightarrow Q$ ’ means ‘ $\text{INFO}(\text{QR}) \leftarrow o$ ,  $\text{LINK}(\text{QR}) \leftarrow \text{AVAIL}$ ,  $\text{QR} \leftarrow \text{LINK}(\text{QR})$ ’; and the deletion operation ‘ $Q \Rightarrow o$ ’ means ‘ $p \leftarrow \text{QF}$ ,  $o \leftarrow \text{INFO}(p)$ ,  $\text{QF} \leftarrow \text{LINK}(p)$ ,  $p \Rightarrow \text{AVAIL}$ ,’ if  $\text{QF} \neq \text{QR}$ .

For example, starting with (127) represented as above, and with  $\text{FIX}(0) = \dots = \text{FIX}(19) = 0$ , a call on  $\text{opt\_out}(13)$  would have the effect of setting  $\text{SIZE}(11) \leftarrow 1$ ,  $\text{SIZE}(23) \leftarrow 2$ ,  $\text{FIX}(5) \leftarrow 7$ ,  $\text{FIX}(19) \leftarrow 9$ ,  $\text{LINK}(8) \leftarrow 0$ ,  $\text{INFO}(7) \leftarrow 13$ ,  $\text{INFO}(9) \leftarrow 13$ ,  $\text{TRIG}(13) \leftarrow 0$ ,  $\text{INFO}(13) \leftarrow 5$ ,  $\text{LINK}(13) \leftarrow 14$ ,  $\text{INFO}(14) \leftarrow 19$ ,  $\text{LINK}(14) \leftarrow 15$ ,  $\text{QR} \leftarrow 15$ .

Use these conventions to design Algorithm O, a “naïve” implementation of the ‘ $\text{opt\_out}$ ’ subroutine described in the text.

**340.** [32] Design Algorithm E, the ‘ $\text{empty\_q}$ ’ 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.

- 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.
- 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  $\text{TRIG}(o)$ , after  $\text{opt\_out}(o)$  has acted, by  $\text{AGE}(o')$ .

**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  $\text{SSTAMP} \leftarrow (\text{SSTAMP} + 1) \bmod 2^{32}$ ; 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  
 $\text{opt\_out}$   
 $\text{empty\_q}$   
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  $0 \leq i < m$ ,  $0 \leq j < n$ , and  $P \subseteq \{1, 2, \dots, mn\}$ , construct an XCC problem whose solutions assign labels  $\{1, 2, \dots, mn\}$  to the cells of an  $m \times n$  board, where the labels define steps  $1, 2, \dots, mn$  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  $10 \times 10$  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  $\leq mn$  plus all numbers  $2^e$  for  $0 \leq e \leq \lg mn$ .
- Exhibit solutions of this problem, for as many  $m \leq n$  as you can.
  - Also count the total number of solutions, for as many  $m \leq n$  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  $[u_{xy} \dots v_{xy}]$  to each cell  $(x, y)$  of the shape and the multiplicity  $[u_p \dots v_p]$  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 ‘ $x \neq a$  or  $y \neq b$  or  $z \neq c$ ’?
- 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<sup>+</sup> 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<sup>+</sup>?
- 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  $1 \leq q \leq 7$ , find the maximum number of knights that can be placed together with  $q$  queens on an  $8 \times 8$  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  $10 \times 10$  pattern in which, for  $1 \leq c \leq 4$ , (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:

$$S(x_1x_2 \dots x_k, y_1y_2 \dots y_k) \iff x_1 = y_2 \text{ and } x_2 = y_3 \text{ and } \dots \text{ and } x_{k-1} = y_k.$$

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.

- When  $k = 3$  and  $x = x_1x_2x_3$ , express the condition  $x_1 = x_3$  in terms of  $S$ .
- When  $x = x_1x_2x_3$  and  $y = y_1y_2y_3$ , express the condition  $x_1 = y_3$  in terms of  $S$ .
- When  $x = x_1 \dots x_k$ ,  $y = y_1 \dots y_k$ ,  $1 \leq i \leq j \leq k$ , express the condition  $x_i = y_j$ .
- 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  $a_1 \dots a_r$ , let  $R_\alpha$  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  $0 \leq i < t$ . For example,  $R_{13}(x, y) = \{xy \in \{01, 12, 32, 34, 45, 56, 76, 78, 89\}\}$ .
- True or false: The relation  $R_\alpha$  is min-closed and max-closed.
  - Prove that there's no homomorphism from  $R_{it}$  to  $R_{jt}$  when  $i \neq j$  (see (4)).
  - Given  $0 \leq i < t$ , 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  $\hat{G}$ , if a value has previously been assigned to any variable  $v_j$  of a sink component  $\{(v_1, a_1), \dots, (v_t, a_t)\}$ , 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  $X = \{x_1, \dots, x_t\}$  with  $x_1 < \dots < x_t$  is represented in  $t + 1$  nodes  $\{X, p_1, \dots, p_t\}$ , with

$$\text{LINK}(X) = p_1; \quad \text{INFO}(p_j) = x_j, \quad \text{LINK}(p_j) = p_{j+1}, \text{ for } 1 \leq j \leq t;$$

$X$  is the list head, and  $p_{t+1} = \perp$  is the address of a special node with  $\text{INFO}(\perp) = \infty$ .

If  $m$  sets  $X_1, \dots, X_m$  are represented in this way, explain how to compute their intersection  $Y = X_1 \cap \dots \cap X_m$ , in  $O(m + |X_1| + \dots + |X_m|)$  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:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

**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:  $x \vee x = x$ ;  $x \vee y = y \vee x$ ;  $(x \vee y) \vee z = x \vee (y \vee z)$ .

- 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).
- In that partial ordering,  $x \vee y$  is the least upper bound of  $x$  and  $y$ .
- 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  $1 \leq m, n \leq 6$ . (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  $1 \leq m, n \leq 6$ .
- **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  $1 \leq m, n \leq 6$ .
- **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  $1 \leq m, n \leq 6$ .
- 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  $3 \times 3$  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  $2 \leq l \leq m \leq n \leq 4$ .
- **391.** [M25] Find a function  $\Delta(x, y, z)$  such that a binary relation  $R(x, y)$  is implicational if and only if  $\Delta$  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  $(a_1x_1 + \dots + a_mx_m) \bmod d = b$ , for integer constants  $\{a_1, \dots, a_m, b\}$ , if and only if that constraint is closed under  $\mu_d$ . (Thus, by Theorem D, a CSP for which  $\mu_d$  is a polymorphism is solvable in polynomial time.)
- 393.** [M25] Consider the  $k$ -ary polymorphisms of  $\{01, 10, 11\}$  (OR) when  $k \geq 1$ .
  - 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  $1 \leq k \leq 5$ ?
- 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  $\{v_1, \dots, v_n\}$ . The set of all  $n$ -tuples  $a_1 \dots a_n$  such that  $\{v_1 = a_1, \dots, v_n = a_n\}$  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  $R = S_{0,2,3} = \{000, 011, 101, 110, 111\}$  ( $'x + y + z \neq 1'$ ) has exactly  $2^k + 1$   $k$ -ary polymorphisms, for all  $k \geq 1$ .

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

**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  $\mathcal{I}_3(\{'x+y+z \neq 1'\})$ ?

**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  $\hat{R}$  by adapting exercise 402 to  $\mathcal{I}_3(\{\hat{R}\})$ .

**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  $S^{(m)} = S^{(m)}(x_1, \dots, x_m)$  be the Boolean  $m$ -ary relation ' $x_1 + \dots + x_m = 1$ '. 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 ' $x, y, \text{ and } z \text{ are not all equal}$ '. 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  $\{a_{11} \dots a_{1m}, \dots, a_{k1} \dots a_{km}\}$ , and let  $\Gamma$  be a family of relations. Show that there's a gadget for  $R$  using only the relations in  $\Gamma$  if and only if we have  $[a_{11} \dots a_{k1}] \dots [a_{1m} \dots a_{km}] \in R$  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  $\pi = (3.0323221430334324112412240414023 \dots)_5$ .)

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,  $\mathcal{I}_4(\{R_1, \dots, R_5\})$  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  $\Theta_3$  of an infinite family of binary "shortcut" relations  $\Theta_d$  for all  $d \geq 3$ , defined as follows: We start with  $d$  arcs  $0 \rightarrow 1 \rightarrow 2 \rightarrow \dots \rightarrow d-1 \rightarrow 0$ , which give us the oriented  $d$ -cycle  $C_d^+$ ; then we add a "shortcut" arc  $0 \rightarrow 2$ , so that we also have an oriented  $(d-1)$ -cycle.

- Prove that  $\Theta_d$  has only one unary polymorphism.
- Construct a gadget for the relation  $C_d^+$  from  $\Theta_d$ .
- Use  $\Theta_d$  to make a gadget for some relation  $R$  that contains exactly  $d^2 - 1$  pairs.
- If  $\Phi_d$  is *any* binary relation on  $\{0, 1, \dots, d-1\}$  that satisfies the conditions proved for  $\Theta_d$  in (a), (b), and (c), prove that  $\Phi_d$  is potent.

**413.** [M26] Can a binary relation on  $\{0, 1, \dots, d-1\}$  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  $3 \times 5$  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  $(A, B, R)$ .

self-dual  
unit vector  
nonequality  
Cooper  
not all equal  
gadget  
pi, as source  
relations  
oriented  $d$ -cycle  
shortcut  
gadget  
closed set  
polarities

► 417. [HM30] Identify the closed subsets of  $A$  and  $B$  in the following cases of polarity:

- $A = B =$  all positive integers,  $xRy = 'x \text{ divides } y'$ .
- $A = B =$  all positive integers,  $xRy = 'x^2 \text{ divides } y^3'$ .
- $A = B =$  all  $2 \times 2$  matrices of integers,  $xRy = 'xy = yx'$ .
- $A = B =$  all rational numbers,  $xRy = 'x \leq y'$ .
- $A = B =$  all rational numbers,  $xRy = 'x < y'$ .
- $A =$  all real numbers,  $B =$  all polynomials over the integers,  $xRy = 'y(x) = 0'$ .

$2 \times 2$  matrices  
matrices  
rational numbers  
Galois  
field  
automorphisms  
spectrum  
essential spectrum  
full  
Boolean operations  
Willard pairing  
clone  
Maltsev term  
spectrum

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  $z_1, \dots, z_n$ . Let  $A$  be the smallest field that contains all rational numbers and all the roots  $\{z_1, \dots, z_n\}$ . Let  $B$  be the group of automorphisms of that field. And let  $xRy = 'y \text{ fixes the point } x'$ . 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  $f_0, f_1, \dots$ , where  $f_k$  is the number of  $k$ -ary operations in  $\mathcal{C}$ . The “essential spectrum”  $e_0, e_1, \dots$  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.

- Show that  $f_n = \sum_k \binom{n}{k} e_k$  and  $e_n = \sum_k (-1)^{n-k} \binom{n}{k} f_k$ .
- What's the spectrum of the *smallest* clone of operations on a set of size  $d \geq 1$ ?
- How many Boolean operations  $f(x_1, x_2, x_3, x_4)$  are full?

420. [M33] (Willard pairing.) If  $f(x_1, \dots, x_k)$  is a  $k$ -ary function and  $1 \leq i, j \leq k$ ,  $i \neq j$ , let  $f_{ij}(x_1, \dots, x_k)$  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  $\{0, 1, \dots, d-1\}$ , where  $d < k$ , has a subfunction  $f_{ij}$  whose essential arity is at least  $k-2$ .*

- Show that the assumption  $d < k$  is necessary when  $k > 2$ .
- 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  $i \neq l \neq j$ , then  $f$  depends on  $x_l$ .
- Let  $r = \max_{1 \leq i < j \leq k} \text{ea}(f_{ij})$ . If the lemma is false, show that at least one subfunction  $f_{ij}$  with  $\text{ea}(f_{ij}) = r$  is independent of  $x_j$ .
- Now assume that  $f(x_1, \dots, x_{k-2}, x, x) = h(x_1, \dots, x_r)$ , where both  $f$  and  $h$  are full. Prove that  $f(a_1, \dots, a_k) = h(a_1, \dots, a_r)$  when  $\{a_1, \dots, a_k\}$  aren't distinct.

421. [M25] Sketch an algorithm that generates all of the  $k$ -ary operations on  $D = \{0, 1, \dots, d-1\}$  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  $i_1 \dots i_l$  of nonnegative integers such that  $\max\{i_1, \dots, i_l\} = m$ , given  $l > 0$  and  $m \geq 0$ .

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  $D = \{0, 1, \dots, d-1\}$ , let  $\mathcal{C}$  be the clone of the ternary operation  $\mu_d(x, y, z) = (x - y + z) \bmod d$  in exercise 392.

- 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.
- What is the spectrum of  $\mathcal{C}$ ?
- Find a value of  $d$  such that  $\mathcal{C}$  contains no  $k$ -ary WNU operation for  $k < 1000$ .

- **426.** [M30] On the domain  $D = \{0, 1, \dots, d-1\}$ , let  $\mathcal{C}$  be the clone of the ternary operation  $t(x, y, z) = (x = y? z: x)$ . (This operation is called the “discriminator”.)
- 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  $d \geq 4$ , for which values of  $(w, x, y, z)$  does it equal (i)  $w$ ? (ii)  $x$ ? (iii)  $y$ ? (iv)  $z$ ?
  - Describe all of the  $k$ -ary operations in  $\mathcal{C}$ .
  - 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  $\{0, 1, \dots, d-1\}$ , and let  $g(x_1, \dots, x_k)$  be an element of  $\overline{O} = \text{Pol}(\text{Inv}(O))$ . Prove that  $g \in O$ . *Hint:* Consider the set of  $d^k$ -tuples  $R = \{f(0, \dots, 0) \dots f(d-1, \dots, d-1) \mid f \text{ is a } k\text{-ary operation of } O\}$ .
- **430.** [M22] On domain  $\{0, 1, 2\}$ , let  $f_k(x_1, \dots, x_k) = (x_1 + \dots + x_k = 1? 0: 2)$ ; also let  $R_m(x_1, \dots, x_m) = [x_1 + \dots + x_m = 1] \vee [2 \in \{x_1, \dots, x_m\}]$ . For which  $k \geq 1$  and  $m \geq 1$  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  $m \geq 0$ ; (b) the empty  $m$ -ary relation  $\emptyset$  for all  $m \geq 0$ ; (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  $\sigma$  such that  $\Sigma_{12}\Sigma_{34}\Sigma_{56}\Pi_\sigma(\Theta \otimes \Theta \otimes \Theta) = \{001122\}$ , where  $\Theta$  is the relation defined in (159).
- **436.** [M30] Describe the relational clone  $\text{Inv}(\{t\})$  of the discriminator (exercise 426).
- 437.** [M20] Exactly how many  $k$ -ary operations on  $\{0, 1, \dots, d-1\}$  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  $u \rightarrow v$  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  $\Gamma$  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  $(a_1, \dots, a_n)$  for variables  $(v_1, \dots, v_n)$  satisfy all constraints, so do the values  $(f(a_1), \dots, f(a_n))$ .

Let  $\Gamma$  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  $\Gamma$ , 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  $f \in O$ .

- Prove that there's always a squashing function  $e \in O$  such that  $ee = e$ .
- Is the range  $fD$  of a squashing function uniquely determined by  $O$ ?
- 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  $x = y$ .
- Consequently every  $f \in O$  has a quasi-inverse  $f^\sim \in O$  such that  $f^\sim f e = e$ .
- The set  $\Gamma$  is called a *core* if every endomorphism of  $\Gamma$  is an automorphism (a permutation of  $D$ ). Show that every CSP  $\mathcal{P}$  can be transformed in polynomial time to a CSP  $\hat{\mathcal{P}}$  whose relations  $\hat{\Gamma}$  are a core, where  $\mathcal{P}$  is satisfiable if and only if  $\hat{\mathcal{P}}$  is.

**443.** [HM30] (*Laxity.*) In Theorem D, prove that  $\Gamma$  has an LWNUP polymorphism  $w$  if and only if the associated core  $\hat{\Gamma}$  of exercise 442(e) has a WNU polymorphism  $\hat{w}$ .

► **444.** [M35] Let  $\Gamma = \{U, V, W\}$  be the following three binary relations on  $D = \{0, 1, 2\}$ :

$$U = \{01, 10\}; \quad V = \{00, 10, 11, 12, 20, 22\}; \quad W = \{00, 02, 10, 11, 12, 22\}.$$

Prove that  $f(x, y, z) = (xy + yz + zx) \bmod 2$  is an LWNUP polymorphism for  $\Gamma$ , but  $\text{Pol}(\Gamma)$  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

$$f(y, x, \dots, x) = y = f(x, \dots, x, y)$$

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  $\mu_d$  of exercise 392 satisfies the minority law.

Given a GMM op  $f(x_1, \dots, x_k)$ , construct a Taylor term  $t(x_1, \dots, x_{2k-2})$ .

**447.** [M22] Deep theorems of universal algebra prove that a clone of operations contains an LWNUP if and only if it contains a “lax Siggers term,” namely a quaternary operation  $s(w, x, y, z)$  that satisfies the identity  $s(r, a, r, e) = s(a, r, e, a)$ .

- Prove that such a clone also contains a lax Maróti pair (see (186)).
- What lax Maróti pairs  $(p, q)$  arise when  $p$  or  $q$  is a projection?

**448.** [M17] Let  $p$  be a ternary operation on  $\{0, 1, \dots, d-1\}$ . 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  $\Gamma$ ?

► **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  
LWNUP polymorphism  
WNU polymorphism  
LWNUP polymorphism  
WNU  
Taylor term  
generalized majority-minority operation  
near unanimity  
Maltsev  
Taylor term  
universal algebra  
LWNUP  
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  $k \leq r \leq m$ . 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  $\tau$  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,  $f(x_1, \dots, x_k) \in \{x_1, \dots, x_k\}$ ). Modify step N1 for the conservative case, by initializing  $S$  to  $D_1 \times \dots \times D_n$  instead of  $D^n$ , where  $D_i$  is the initial domain of the  $i$ th variable.

**462.** [HM21] Given  $d, k$ , and  $n$ , step N1 constructs a stub of size  $\Omega((nd)^{k-1})$  for the complete relation  $[0 \dots d]^n$ . 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  $y = x \setminus z$ ; that is,  $x = (x \cdot y)/y$  and  $y = x \setminus (x \cdot y)$ . The identities  $x = (x/y) \cdot y$ ,  $y = x \cdot (x \setminus y)$ , and  $x = y/(x \setminus y) = (y/x) \setminus y$  are also valid.

Using the operators ‘ $\cdot$ ’, ‘ $/$ ’, and ‘ $\setminus$ ’, 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  $\langle \Pi_{1 \dots (i-1)} R' \rangle_f = \Pi_{1 \dots (i-1)} R$ , for some  $i$  with  $1 < i \leq m$ . Assume furthermore that  $R'$  and  $R$  have the same level- $i$  forkings. Prove that  $\langle \Pi_{12 \dots i} R' \rangle_f = \Pi_{12 \dots i} R$ .

**468.** [20] In (201), for which tuples  $\tau$  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  $\tau$  to  $S$ ,” where  $\tau = t_1 \dots t_m$ , leaves  $S$  unchanged if the monoforkings  $(1, t_1, t_1), \dots, (m, t_m, t_m)$  are already present in  $S$ . The other, “contribute  $\tau$  and  $\tau'$  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  $[0 \dots d]^n$   
 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  $D = \{0, 1, \dots, d-1\}$ . 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  $\Gamma$  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.)

- $n = 5$ ,  $R_{12} = R_{23} = R_{34} = R_{45} = R_{51} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ .
- $n = 5$ ,  $R_{12} = R_{23} = R_{34} = R_{45} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$  and  $R_{51} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$ .
- $n = 5$ ,  $R_{12} = R_{23} = R_{34} = R_{45} = R_{51} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$ .
- $n = 3$ ,  $R_{12} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ ,  $R_{13} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$ , and  $R_{23} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$ .

**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  $k_0 k_1 \dots k_r$  of indices with  $k_0 = i$ ,  $1 \leq k_l \leq n$  for  $0 < l < r$ , and  $k_r = j$ : There's a sequence of values  $x_0 x_1 \dots x_r$  such that  $x_0 = s$ ,  $(x_l, x_{l+1}) \in R_{k_l k_{l+1}}$  for  $0 \leq l < r$ , 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

$$q \mapsto v_1 \setminus a_1, \dots, v_t \setminus a_t, (v \leftarrow a? q': q''), \quad \text{for some } t \geq 0,$$

where the  $v$ 's are variables, the  $a$ 's are domain elements, and the  $q$ 's are states. The meaning is, “Begin deterministically: For  $1 \leq j \leq t$ , 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 ' $q \mapsto (* \leftarrow *? q: q)$ ' 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

$$q_0 \mapsto (v \leftarrow a? q_1: q_2); \quad q_1 \mapsto w \setminus b, (* \leftarrow *? q_2: q_2); \quad q_2 \mapsto (* \leftarrow *? q_2: q_2)$$

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

$$q \mapsto (v \leftarrow a*? q_a: q_{-a})$$

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  $\Omega$ ." 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  $\{v_1, \dots, v_n\}$ , each with domain  $[0 \dots d] = \{0, 1, \dots, d-1\}$ , and subject to any number of further constraints.

- Define a CSA whose solutions  $v_1 \dots v_n$  are those with  $(v_1 + \dots + v_n) \bmod 5 \in \{1, 3\}$ .
- Define a CSA for the solutions where each value occurs at most twice.
- Define a CSA for the solutions where each value occurs either twice or not at all.
- Similarly, design a CSA for all solutions  $v_1 \dots v_n$  that are *restricted growth strings*. (See Section 7.2.1.5; in particular,  $v_1 = 0$  and  $v_2$  is 0 or 1.)
- Let  $d = 2$ , and restrict the solutions to binary strings  $v_1 \dots v_n$  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  $v_n \dots v_2 v_1$  solves a certain CSP whenever  $v_1 v_2 \dots v_n$  does. All domains are  $[0 \dots d]$ . Design a CSA that yields only one of those solutions.

**482.** [23] Suppose the cyclically shifted tuple  $v_2 \dots v_n v_1$  solves a certain CSP whenever  $v_1 v_2 \dots v_n$  does. Design a CSA that yields just one solution in each equivalence class under cyclic shifts. All domains are  $[0 \dots d]$ . *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

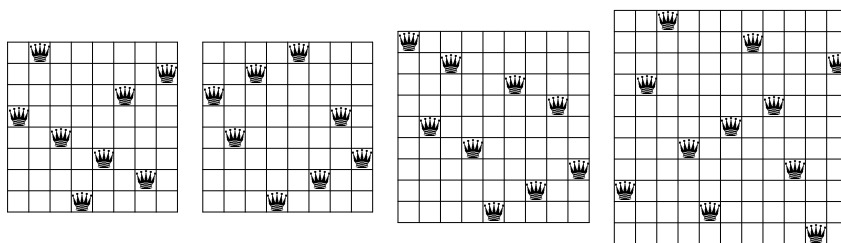
- **483.** [28] Solutions to the  $n$  queens problem belong to the same equivalence class if they differ only by a reflection and/or rotation of the board. The purpose of this exercise is to define *canonical solutions*, of which there's exactly one in each class.

Denote the cells by  $(i, j)$  for  $0 \leq i, j < n$ . 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  $90^\circ$  changes  $R_i$  to  $C_{\bar{i}}$  and  $C_j$  to  $R_{\bar{j}}$ .

- Let  $(a_i, b_i, c_i, d_i) = (R_i, C_{\bar{i}}, \bar{R}_{\bar{i}}, \bar{C}_{\bar{i}})$ . Can we have  $\{a_i, b_i, c_i, d_i\} \cap \{\bar{a}_i, \bar{b}_i, \bar{c}_i, \bar{d}_i\} \neq \emptyset$ ?
- How does reflection of the board change the numbers  $(a_i, b_i, c_i, d_i)$  of a solution?
- Let  $n' = \lfloor n/2 \rfloor$ . Write out the eight values of the  $4 \lceil n/2 \rceil$ -tuple

$$(a_{n'}, b_{n'}, c_{n'}, d_{n'}; a_{n'+1}, b_{n'+1}, c_{n'+1}, d_{n'+1}; \dots; a_{n-1}, b_{n-1}, c_{n-1}, d_{n-1})$$

that occur when the following solutions are rotated and/or reflected:



- Explain why the lexicographically least of eight such tuples is a canonical solution.
- True or false: If  $n = 2n'$ , the canonical tuple begins with  $a_{n'} \leq n' - 2$ .
- 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)

## ANSWERS TO EXERCISES

*For oure exercisce, With sharpe scourges of adversitee Ful ofte to be bete.*

— GEOFFREY CHAUCEUR, *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  $c_1 \in \{1, 2, \bar{3}\}, \dots, c_7 \in \{\bar{3}, \bar{4}, \bar{1}\}$ ;  $\binom{7}{2} = 21$  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  $x_{1B} = [x_1 = B]$ , etc. Then the clauses  $(x_{1B} \vee x_{1S}), (\bar{x}_{1B} \vee \bar{x}_{1S}), (x_{2C} \vee x_{2L}), (\bar{x}_{2C} \vee \bar{x}_{2L}), (x_{3A} \vee x_{3I} \vee x_{3U}), (\bar{x}_{3A} \vee \bar{x}_{3I}), (\bar{x}_{3A} \vee \bar{x}_{3U}), (\bar{x}_{3I} \vee \bar{x}_{3U}), (x_{4E} \vee x_{4O}), (\bar{x}_{4E} \vee \bar{x}_{4O}), (x_{5D} \vee x_{5N}), (\bar{x}_{5D} \vee \bar{x}_{5N})$  establish the domains. And the clauses  $(R_{11} \vee R_{12} \vee R_{13}), (\bar{R}_{11} \vee \bar{x}_{1B}), (\bar{R}_{11} \vee x_{3A}), (\bar{R}_{11} \vee x_{5N}), (\bar{R}_{12} \vee x_{1B}), (\bar{R}_{12} \vee x_{3U}), (\bar{R}_{12} \vee x_{5D}), (\bar{R}_{13} \vee x_{1S}), (\bar{R}_{13} \vee x_{3I}), (\bar{R}_{13} \vee x_{5N}), \dots, (\bar{R}_{33} \vee x_{2L}), (\bar{R}_{33} \vee x_{4E}), (\bar{R}_{33} \vee x_{5D})$  establish the relations. (Many other encodings are possible; this one is systematic and avoids trickery.)
4. Primary  $R_1, R_2, R_3$ ; secondary  $x_1, \dots, x_5$ . Options ' $R_1 x_1:B x_3:A x_5:N$ ', ' $R_1 x_1:B x_3:U x_5:D$ ', ' $R_1 x_1:S x_3:I x_5:N$ ',  $\dots$ , ' $R_3 x_2:L x_4:E x_5:D$ '. (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  $2^{d_1 \dots d_k}$   $k$ -ary relations on  $(D_1, \dots, D_k)$ , 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  $C_{\text{odd}}$  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  $u - v$  of  $G$  by a path  $u - uv - vu - v$ , 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  $\{1, \dots, |V|\}$  for which  $i - j$  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  $(\{0, \dots, m-1\}, O)$  be the oriented cycle  $C_m$ , where  $m = |A|$  and  $O = \{ij \mid j = (i+1) \bmod m\}$ . “Is there a homomorphism from  $(\{0, \dots, m-1\}, O, \neq)$  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  $k$ DM problem ( $k$ -dimensional matching).

11. Given similar relational structures  $S = (U, R_1, \dots, R_t)$  and  $S' = (U', R'_1, \dots, R'_t)$ , the corresponding CSP has variables  $U$ , each with domain  $U'$ . Suppose  $U = \{1, \dots, n\}$ . The values  $x_{i_1} \dots x_{i_k}$  of every  $k$ -tuple  $i_1 \dots i_k \in R_j$ , 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  $\begin{pmatrix} w z^- & w^- z^- \\ w z & w^- z^- \end{pmatrix}$ , where  $z^-$  denotes  $1/z$ . By induction we have  $G_N(z) = \begin{pmatrix} w & w^- \end{pmatrix} T^{N-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ . 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  $T = S \begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix} S^-$ , where  $S = \begin{pmatrix} \lambda - w^- z^- & \mu - w^- z^- \\ w z & w^- z^- \end{pmatrix}$ . 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  $\mu$ . 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  $G_k(z) = \begin{pmatrix} w & w^- \end{pmatrix} T^{k-1} X T^{N-k} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ , where  $X = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$ . To put this in closed form, let  $Y = S^- X S$ , so that  $T^{k-1} X T^{N-k} = S \begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix}^{k-1} Y \begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix}^{N-k} S^-$ . 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  
 $C_m$   
 $k$ DM 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  $(A, B, \dots, H)$  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.

**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  $A \leq j < H$  and an edge  $u - v$ , where  $u < v$ . There are 64 options  $(v, k)$ , where  $1 \leq v \leq 8$  and  $A \leq k \leq H$ ; 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  $(2, E)$  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  $(2, A)$ ,  $(5, A)$ ,  $(6, A)$ ,  $(8, A)$ ; then also remove options  $(2, H)$  and  $(8, B)$ , and use the pairwise ordering trick of exercise 7.2.2.1–20 with  $m = 6$ ,  $\alpha_i = (2, B + i)$ ,  $\beta_i = (8, C + i)$  to ensure that the label of 2 is less than the label of 8. (This introduces secondary items  $y_1, \dots, y_5$ ; it also puts  $y_2$  and  $y_3$  into option  $(2, E)$ .) 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 K $\mu$ .]

**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  $(AB, BC), \dots, (FG, GH)$  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  $(AB, CD), (AB, DE), \dots, (EF, GH)$ , to be one of the other 62 ordered pairs of edges. (The all-different constraint would be redundant.)

**17.** FABABACDCE (and its mirror image ECD CABABAF).

**18.** The mirror image of a solution with  $f \geq 5$  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  $f_{0k} + f_{1k} + \dots + f_{(l'q_k-1)k} \leq l'p_k$  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  $0 \leq i < t$ . Also a primary item for the name of each model type, with its given multiplicity.

Cohoon  
no-touch puzzle  
Hamiltonian path  
O’Beirne  
Koplowitz  
Gardner  
symmetry  
pairwise ordering trick  
XC: exact cover  
sharp preference heuristic  
K $\mu$ : kilomems  
symmetry  
Dull  
slot

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  $0 \leq j \leq t - q_k$  and  $0 \leq k < m$ , having multiplicity  $[0 \dots p_k]$ . (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  $j \leq i < j + q_k$ . (Thus, one option for Fig. 100 is '2 B  $u_{10}$   $u_{20}$   $u_{03}$   $u_{13}$   $u_{23}$ '.)

**21.** Notice that  $f_{0k} + \dots + f_{(t-lq_k-1)k} \geq r_k - lp_k$  if and only if  $\bar{f}_{0k} + \dots + \bar{f}_{(t-lq_k-1)k} \leq s_{lk} = t - lq_k - r_k + lp_k$ . Therefore introduce primary items  $v_{lk}$  for  $0 < l < \lceil r_k/p_k \rceil$  and  $0 \leq k < m$ , having multiplicity  $[0 \dots s_{lk}]$ . Include  $v_{lk}$  in the option for every model that does *not* use feature  $k$ , for every slot  $i$  in the range  $0 \leq i < t - lq_k$ . (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 G $\mu$ , 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 G $\mu$  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  $\alpha$  with  $\nu_\alpha \leq p$ , having arcs  $\alpha \rightarrow \beta$  when the last  $q-1$  bits of  $\alpha$  match the first  $q-1$  bits of  $\beta$ . (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  $\sigma$  in a given digraph is  $\sum_\alpha G^\alpha(z)$ , where  $G^\alpha(z) = [\alpha = \sigma] + z \sum_{\beta \rightarrow \alpha} G^\beta(z)$  for each vertex  $\alpha$ . 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,  $G_{1q}(z) = (1+z+\dots+z^{q-1})/(1-z-z^q)$  and  $G_{(q-1)q}(z) = (1+z+\dots+z^{q-1})/(1-z-z^2-\dots-z^q)$ . 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  $0 \leq P_{ij} \leq m$ ,  $P_{ij} \geq P_{i(j+1)}$ ,  $P_{ij} \geq P_{(i+1)j}$ , 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  $k = 1, 2, \dots, m$ , 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  $\begin{smallmatrix} 441 \\ 210 \end{smallmatrix}$ . 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  $((-140, 0, 0), (75, 75, -100), (0, 252, -280), (40, -100, -200), (90, -50, 0), (140, 50, 0), (-240, 0, 200), (140, 0, 0), (240, 0, 200), (-140, -50, 0), (-90, 50, 0), (-40, 100, -200), (0, -252, -280), (-75, -75, -100))$ . Then the seven hexagons  $023 \text{ --- } 310 \text{ --- } 501 \text{ --- } 054 \text{ --- } 460 \text{ --- } 206, 134 \text{ --- } 421 \text{ --- } 612 \text{ --- } 165 \text{ --- } 501 \text{ --- } 310, \dots, 612 \text{ --- } 206 \text{ --- } 460 \text{ --- } 643 \text{ --- } 356 \text{ --- } 165$  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  $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ , 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  $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$  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  $\binom{10}{1} + 4\binom{10}{2}, 4\binom{10}{2} + 8\binom{10}{3}, 4\binom{10}{2}, 8\binom{10}{3} + 8\binom{10}{4}$  valid 3VPs, a total of 4150.

(And the  $B^4$  histoscapes of  $2 \times 2$  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  $2 \times 2$  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  $c(x_0, \dots, x_n)$ , all initially 1.

**Q1.** [Iterate on rows.] Do step Q2 for  $i = 1, \dots, m-1$ ; then go to Q3.

**Q2.** [Iterate on columns.] Do subroutine  $(i, j)$  below for  $j = 1, \dots, n-[i=m-1]$ .

**Q3.** [Sum.] The answer is  $\sum \{c(x_0, \dots, x_n) \mid 0 \leq x_0, \dots, x_n < t\}$ . ■

Subroutine  $(i, j)$  is the following: Set  $q \leftarrow (j-i) \bmod (n+1)$ . For all  $t^n$  choices of  $(x_0, \dots, x_n)$  such that  $x_q = 0$ , compute  $t$  sums for  $0 \leq d < t$ , namely

$$s_d \leftarrow \sum_{0 \leq k < t} [r_{ij}(k, x_{(q+1) \bmod (n+1)}, x_{(q-1) \bmod (n+1)}, d)] c(x_0, \dots, x_{q-1}, k, x_{q+1}, \dots, x_n);$$

tableau shape  
rim representation  
Gessel  
coordinates  
matrix coordinates  
Cartesian coordinates  
structured  
dynamic programming  
quaternary relation



then set  $c(x_0, \dots, x_{q-1}, d, x_{q+1}, \dots, x_n) \leftarrow s_d$  for  $0 \leq d < t$ . (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  $\begin{pmatrix} a_{00}a_{01}\dots a_{0(n-1)}a_{10}\dots a_{1(n-1)}a_{20}\dots \\ a_{10}a_{11}\dots a_{1(n-1)}a_{20}\dots a_{2(n-1)}a_{30}\dots \end{pmatrix}!$ )

The method works because, when subroutine  $(i, j)$  begins,  $c(x_0, \dots, x_n)$  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

$$(a_{(i-1)(j-1)}, \dots, a_{(i-1)(n-1)}, a_{i0}, \dots, a_{i(j-1)}) = (x_q, x_{q+1}, \dots, x_n, x_0, \dots, x_{q-1}).$$

About 1.8 teramems of computation suffice to show that the desired number of  $8 \times 8$  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.)

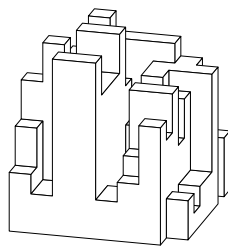
**29.** We essentially want to run that algorithm in reverse. To reverse step Q3, let the counts  $c(x_0, \dots, x_n)$  be renamed  $c_j$  for  $0 \leq j < t^{n+1}$ , in any convenient way. Then for  $j = 0, 1, \dots$ , set  $k \leftarrow k - c_j$  if  $k \geq c_j$ ; but stop when  $k < c_j$ . That gives us suitable values of  $(x_0, x_1, \dots, x_n)$ , which will be  $(a_{(m-2)(n-1)}, a_{(m-1)0}, \dots, a_{(m-1)(n-1)})$ . And we'll want the  $k$ th 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  $k$ th solution when the final  $(m-i+1)n-j-1$  elements are given.

We must rerun the algorithm for each  $(i, j) = (m-1, n-1), (m-1, n-2), \dots, (1, 1)$ , 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”  $8 \times 8$  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  $0 \leq k < a_{ij}$ .

Second, mark all the “visible” faces of cubies, by doing the following for all  $(i, j, k)$  with  $\bar{ijk} = 111$  and  $b_{ijk} = 1$ : If  $b_{(i\pm 2)jk} = 0$ , set  $b_{(i\pm 1)jk} \leftarrow 1$ ; if  $b_{i(j\pm 2)k} = 0$ , set  $b_{i(j\pm 1)k} \leftarrow 1$ ; if  $b_{ij(k\pm 2)} = 0$ , set  $b_{ij(k\pm 1)} \leftarrow 1$ . (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  $\bar{ijk} = 011$  and  $b_{ijk} = 1$ : If  $b_{i(j\pm 2)k} = 0$ , set  $b_{i(j\pm 1)k} \leftarrow 1$ ; if  $b_{ij(k\pm 2)} = 0$ , set  $b_{ij(k\pm 1)} \leftarrow 1$ . Also do this, for all  $(i, j, k)$  with  $\bar{ijk} = 101$  and  $b_{ijk} = 1$ : If  $b_{(i\pm 2)jk} = 0$ , set  $b_{(i\pm 1)jk} \leftarrow 1$ .

Fourth, mark all the vertices, by doing the following for all  $(i, j, k)$  with  $\bar{ijk} = 001$  and  $b_{ijk} = 1$ : If  $b_{ij(k\pm 2)} = 0$ , set  $b_{ij(k\pm 1)} \leftarrow 1$ .

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  $\bar{ijk} = 000$ : “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  $\begin{pmatrix} 100 \\ 111 \end{pmatrix}$ . 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  $2 \times 2$  matrix  $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$  is a vortex if and only if  $[a < b] + [b < d] + [d < c] + [c < a]$  is odd.)

(c) First row  $(1, \dots, n)$ , second row  $(2n, \dots, n+1)$ , and so on.

(d) True, by answer 27 (case 4).

(e) It suffices to verify this for  $2 \times 2$  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  $\{0, 1, \dots, mn-1\}$  and whose  $2 \times 2$  submatrices satisfy  $r_{ij}(a_{(i-1)(j-1)}, a_{(i-1)j}, a_{ij}, a_{i(j-1)})$ , with a dynamic programming algorithm structured as the method of answer 28. But this time we need counts  $c(x_{n-t}, \dots, x_t)$  for each of the  $t^{n+1}$  choices of *distinct* elements with  $0 \leq x_{n-t}, \dots, x_t < t$ , 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  $13^6 = 1235520$  when  $(i, j) = (2, 2)$ , but it rises to  $25^6 = 127512000$  during the last round when  $(i, j) = (4, 4)$ .)

Two ideas make it possible to represent these numerous counts efficiently in memory. Count  $c(x_{n-t}, \dots, x_t)$  is the number of partial solutions  $x_0 \dots x_t$  whose final  $n+1$  elements are  $x_{n-t} \dots x_t$ . Those counts can be represented by  $y_{n-t} \dots y_t$ , 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  $x_0 \dots x_8$  might be  $x_5 x_6 x_7 x_8 = 3142$ ; we represent them

square torus  
torus  
vortex  
dynamic programming  
all different  
inversions  
pi, as random example

by  $y_5 y_6 y_7 y_8 = 1132$ . Or, going the other way, if  $y_5 y_6 y_7 y_8 = 3141$ , then  $x_5 x_6 x_7 x_8$  must have been 6251. This representation has the nice property that  $0 \leq y_j \leq j$  for  $n - t \leq j \leq t$ , so there clearly are  $t^{n+1}$  possibilities.

Every permutation  $x_0 \dots x_t$  of  $\{0, \dots, t\}$  yields  $t + 2$  permutations  $x'_0 \dots x'_{t+1}$  of  $\{0, \dots, t + 1\}$ , if we choose  $x'_{t+1}$  arbitrarily and then set  $x'_j \leftarrow x_j + [x_j \geq x'_{t+1}]$ . For example, if  $t = 8$  and  $x_5 x_6 x_7 x_8 = 3142$ , the ten permutations obtained from  $x_0 \dots x_8$  will have  $x'_5 x'_6 x'_7 x'_8 = 42530, 42531, 41532, 41523, 31524, 31425, 31426, 31427, 31428$ , or  $31429$ . And the representations  $y'_5 y'_6 y'_7 y'_8$  of those last five elements will simply be respectively  $31420, 31421, \dots, 31429$ . In general, we'll have  $y'_j = y_j$  for  $0 \leq j \leq t$ , 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  $a_0 \dots a_{2n-1}$  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  $a_1 \dots a_{2k}(2n+1)a_{2k+1} \dots a_{2n}$ , for some  $k$ , where  $a_1 \dots a_{2k}$  and  $a_{2k+1} \dots a_{2n}$  are independently counted by  $Q_k$  and  $Q_{n-k}$ .

(b) Hence  $U'(z) = Q(z)^2$ , where  $Q(z) = 1 + U_1 z^2/2! + 2U_2 z^4/4! + \dots = 1 + zU(z)/2$ . 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  $\{-n, \dots, 0, \dots, n\}$  that begin with  $k$ . For example, the values  $(p_n(-n), \dots, p_n(n))$  for  $1 \leq n \leq 3$  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

$$\sum_{m,n} p_{(m+n)/2} \binom{m-n}{2} \frac{w^m z^n}{m! n!} = \frac{\cosh((w-z)/\sqrt{2})}{\cosh((w+z)/\sqrt{2}) - ((w+z)/\sqrt{2}) \sinh((w+z)/\sqrt{2})},$$

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  $V(z) = 1/(1 - z \tanh z) = 1 + V_1 z^2/1! + V_2 z^4/3! + \dots$ , where  $V_n = 2^{n-1} U_n$ , and let  $\mu$  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  $\mu$ , 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  $|z| \leq 2\mu$  is  $z = -\mu$ . Hence the function  $V(z) - 2/(\mu^2(\mu^2 - z^2))$  is analytic in  $|z| \leq 2\mu$ ; 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  $\mu$  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  $\lambda$  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  $\mu$  and  $\mu^2$ .]

To get further accuracy, Philippe Jacquet observes that there are constants  $\mu_k$  with  $\mu_k \tan \mu_k = -1$  and  $(k-.5)\pi < \mu_k < k\pi$ , for all  $k \geq 1$ ; 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  $0 \leq i < m$  and  $1 \leq j < n$ , 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  $1 \leq i < m$  and  $0 \leq j < n$ . Finally we constrain  $(r_{ij}, c_{ij}, r_{(i-1)j}, c_{i(j-1)})$  to yield a vortex, for  $1 \leq i < m$  and  $1 \leq j < n$ .

(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?

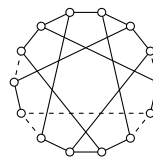
(iii)	3	14	15	12	13
	7	9	2	6	16
	5	20	24	23	17
	4	1	25	22	18
	8	10	11	21	19


(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 ( $\blacktriangle$ ) or convex ( $\blacktriangle$ ). 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  $(v_1 v_2 v_3)$  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  $(o_1 o_2 o_3)_8$ , where  $o_1 = [v \text{ is opposite } v_1 \text{ with respect to } p_{23}]$ ,  $\dots$ ,  $o_3 = [v \text{ is opposite } v_3 \text{ with respect to } p_{12}]$ . (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  $\theta$  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

**39.** (a) A careful case analysis shows that edge  $v_0 - v_1$  is concave if and only if  $X_\epsilon$  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  $\theta$  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  $z \neq z'$ .)

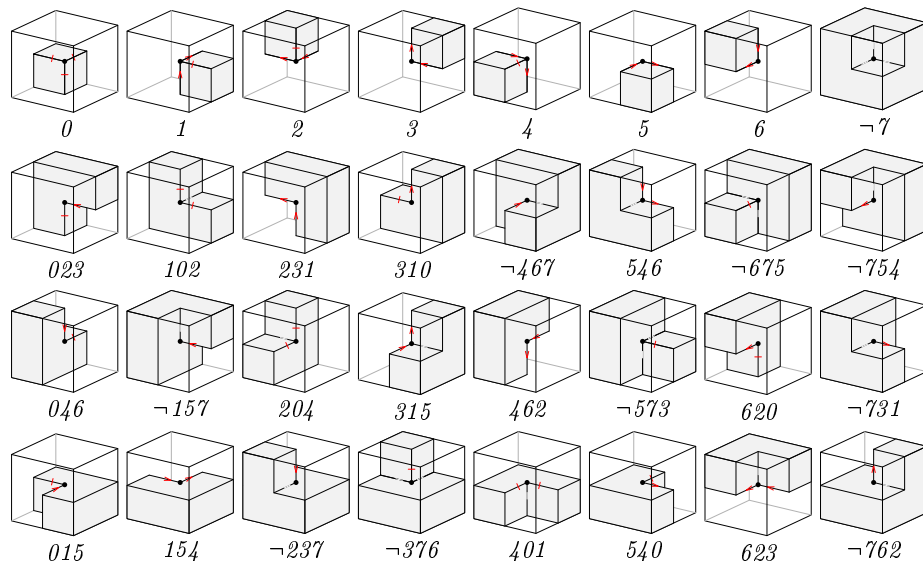
Second,  $X$  mustn't contain noncollinear edges whose projections are collinear. For example, if the line segment  $\{(t, 0, 0) \mid 0 \leq t \leq 1\}$  is an edge of  $X$ , there shouldn't also be an edge of the form  $(u, 0, u)$ . Quantitatively, each edge has the form  $\{(x_0 + \alpha t, y_0 + \beta t, z_0 + \gamma t) \mid t_0 \leq t \leq t_1\}$  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.

plane in three dimensions  
ell

43. (a)  $(a, b) \in \{41, 51, 33, 62\}$ , where ‘11’ abbreviates  $(1, 1)$ , etc.  
 (b)  $(n, p) \in \{12, 13, 22, 23, 32, 33, 42, 43\}$ ;  $(o, p) \in \{13, 23, 36\}$ .  
 (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

van Beek  
unrealizable  
Gardner  
 $K_{3,3}$   
interior angles  
complexity

immediately (see (24) and (25)). Although we can deduce  $p = 3$  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  $\{a, b, \dots, s\}$  and 26 secondary items  $\{ab, ac, \dots, rs\}$  (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  $x_1 \dots x_n$  solve  $H$  if and only if  $x_1^R \dots x_n^R$  solves  $H^R$ , where  $+^R = +$ ,  $-^R = -$ ,  $<^R = >$ ,  $>^R = <$ . (For example, in the reflection of (20) we have  $a = V(c, b)$ ; the corresponding constraint is  $(ac, ab) \in \{<+, <>, +>, >-, ><, -<\}$ , which is the same as  $(ab, ac) \in \{+<, ><, >+, ->, <>, <- \}$ , which is the same as  $(ab, ac) \in \{>+, ><, +<, <- , <>, ->\}$ .)

[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,  $a = V(b, c)$ ,  $b = V(c, a)$ ,  $c = V(b, a)$ .

(b)  $H$  is realizable if and only if each of its connected components is realizable. If  $H$  is connected and its junctions  $\{j_0, j_1, \dots, j_{p-1}\}$  all have type V, we can assume that  $j_k = V(j_{k+\sigma_k}, j_{k-\sigma_k})$  for  $0 \leq k < p$ , with subscripts treated mod  $p$ , where each  $\sigma_k$  is  $\pm 1$ . When  $p = 3$ , we must have  $\sigma_0 = \sigma_1 = \sigma_2$ . When  $p = 4$ , we must not have  $\sigma_0 \neq \sigma_1 \neq \sigma_2 \neq \sigma_3$ . 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  $a = b = c = W(d, e, f)$ ,  $d = e = f = W(a, b, c)$  is  $K_{3,3}$ .

(d) The interior angles of a polygon with  $m$  vertices sum to  $(m-2)180^\circ$ ; hence at most  $m-3$  of them are greater than  $180^\circ$ .

(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  $j_0 j_1, j_1 j_2, \dots, j_9 j_{10}$  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



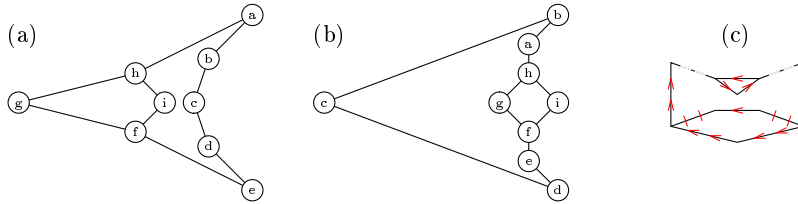


Fig. A-13. Unusual examples of HC pictures.

bridge  
boundary cycle  
trace  
organ pipe order, inverse  
Kane  
operator norm  
eigenvalues

**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  $\text{trace}(\mathbf{V}\mathbf{Y}\mathbf{Y}\mathbf{V}\mathbf{W}\mathbf{V}\mathbf{W}) = 4mmp + glpp$ ; 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  $4 \times 4$  matrix product  $j_0 j_1 \dots j_{q-1}$ , and let  $M = M_0 M_1 \dots M_{q-1}$ . By induction we can verify that  $P_{ij} = P_{(i \oplus 1)(j \oplus 1)}$  for  $0 \leq i, j < 4$ ;  $P_{00} + P_{01} = F_{q+1}$ ;  $P_{02} + P_{03} = P_{20} + P_{21} = F_q$ ;  $P_{22} + P_{23} = F_{q-1}$ ; and  $M_{ij} = P_{(2i)(2j)} - P_{(2i)(2j+1)}$  for  $0 \leq i, j < 2$ . Hence  $\text{trace } P = P_{00} + P_{11} + P_{22} + P_{33} = P_{00} + P_{00} + P_{22} + P_{22}$  and  $\text{trace } M = M_{00} + M_{11} = P_{00} - P_{01} + P_{22} - P_{23} = P_{00} - (F_{q+1} - P_{00}) + P_{22} - (F_{q-1} - P_{22})$ .

(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  $\Delta_{a,b} = F_{a+1}F_{b+1} - F_aF_b = \frac{1}{5}(L_{a+b+1} + 2(-1)^b L_{a-b})$ . Hence  $\Delta_{a,b} - \Delta_{a-1,b-1} = \frac{1}{5}(L_{a+b} + 4(-1)^b L_{a-b})$ , and the values of  $t_p = \text{trace}(A^p B^{q-p})$  occur in a peculiar order:

$$t_1 < t_3 < \dots < t_{\lfloor q/2 \rfloor} = t_{\lceil q/2 \rceil} < \dots < t_2 < t_0, \quad \text{with } t_p = t_{q-p}.$$

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  $X = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ . Thus  $M$  is a product of  $q$   $A$ s, but with  $m$   $X$ s inserted somehow, where  $m$  is the number of switches between  $\mathbf{V}$  and  $\mathbf{A}$  in the cycle. Our goal is to prove that  $\text{trace } M \geq 2F_q - L_q = -F_{q-3}$ .

We can assume that  $M = (AXA)A^{p_1}(AXA)A^{p_2} \dots (AXA)A^{p_m}$ , where  $p_k \geq -1$  for  $1 \leq k \leq m$ . If all  $p_k$  are nonnegative,  $\text{trace } M \geq 0$ , because  $AXA = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ .


If  $p_1 = p_m = -1$ , we have  $M = ABA^{p_2+2}B^{p_3+2} \dots B^{p_{m-1}+2}$ . And  $ABA = BAB$  implies that  $\text{trace } M = \text{trace}(BAB A^{p_2+1} B^{p_3+2} \dots B^{p_{m-1}+2}) = \text{trace}(ABA^{p_2+1} B^{p_3+2} \dots B^{p_{m-1}+3}) = \dots = \text{trace}(AB^{p_3+1} \dots B^{p_{m-1}+p_2+4})$ , thus reducing  $m$  by 2. Therefore we can assume that at least one  $p_k$  is  $-1$ , but no two  $-1$ s 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  $\|M\| \leq \phi^{n-5}$  when  $m \geq 4$ . (We save a factor of  $\phi^2$  when  $p_k = -1$ ,  $\phi$  when  $p_k \geq 0$ .)

Finally, let  $M$  have eigenvalues  $\lambda$  and  $\hat{\lambda}$ , where  $|\lambda| \geq |\hat{\lambda}|$ . Then  $\text{trace } M = \lambda + \hat{\lambda}$ , and  $\lambda\hat{\lambda} = \det M = (-1)^q$ . So  $|\text{trace } M| \leq |\lambda| + 1/|\lambda| \leq \phi^{n-5} + \phi^{5-n} \leq F_{n-3}$ , 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,  $D_{n+1} = D_n A X D_n^T X$ ; and the surprising formula  $D_{n+3} = \begin{pmatrix} 1 & n-1 \\ -1-n & -n^2 \end{pmatrix}$  arises by induction for  $n \geq 0$ . Consequently  $T_n$  has  $\text{trace}(D_{n-1} A D_{n-1} A) + L_{2^n} = L_{2^n} - 1$  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 Reutersvård'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  $6 \times 6$  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  
Reutersvård  
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  $\pm 1$ , 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  $[A < B] + [B < C] + [C < A] \in \{1, 2\}$ .

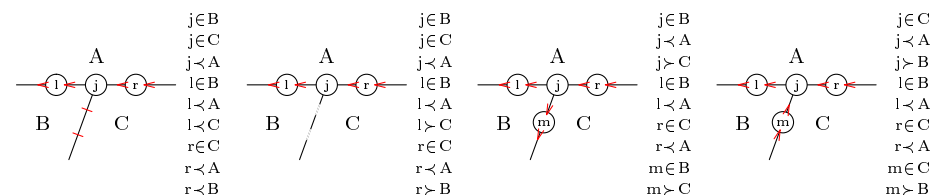
Dynamic programming works well, as in exercise 28, and this problem is considerably easier than that one: Let box  $(i, j)$  in row  $i$ , column  $j$  for  $0 \leq i < m$  and  $0 \leq j < n$  be adjacent to boxes  $(i, j+1)$ ,  $(i+1, j)$ , and  $(i+1, j+1)$ ; and consider the number  $c_n(x_1, \dots, x_{m-1})$  of  $m \times n$  solutions with  $x_j = [(i-1, n-1) < (i, n-1)]$ . After setting  $c_1(x_1, \dots, x_{m-1}) \leftarrow 1$ , we can readily compute the  $2^{m-1}$  counts  $c_{n+1}(x_1, \dots, x_{m-1})$  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  $t_{m,n} = \sum \{c_n(x_1, \dots, x_{m-1}) \mid 0 \leq x_k \leq 1\}$ . For example,  $(t_{3,1}, t_{3,2}, t_{3,3}, \dots, t_{3,n}, \dots) = (4, 162, 6570, \dots, \lceil cr^n \rceil, \dots)$ , where  $r = (41 + \sqrt{1609})/2 \approx 40.556$  and  $c = (1609 + 31\sqrt{1609})/28962 \approx .0985$ . We also have  $t_{6,6} = 22406540276117433798 \approx 2^{64.28}$ ;  $t_{10,10} = 2333623171515704644702 \dots 99558 \approx 2^{193.89}$ .

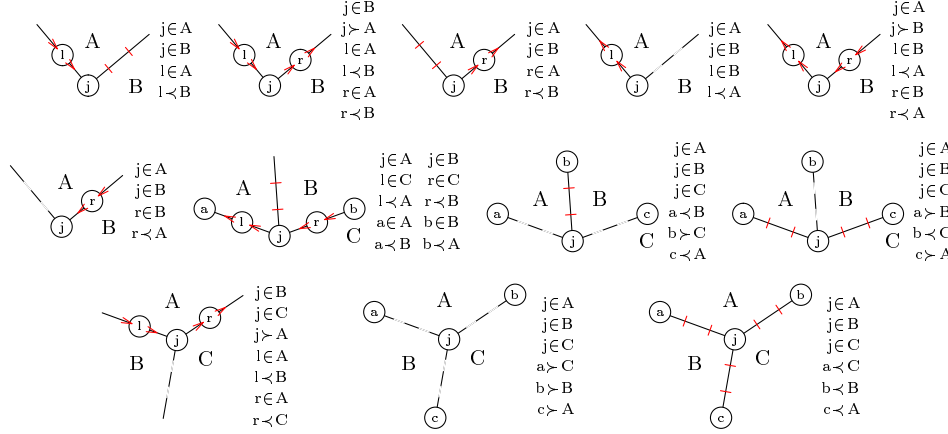
**63.** Regarding  $H$  as an embedded planar graph, let  $F$  be the set of its faces. Each  $f \in F$  will correspond to part (or all) of some face  $\hat{f}$  of  $X$ —except that  $H$ 's exterior face  $f_0$  will correspond to a “background plane”  $\hat{f}_0$ , which is sufficiently distant that 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  $\hat{j}$  of  $X$ . Every remaining junction  $j$  will correspond to an *artificial* vertex  $\hat{j}$ , 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



strict inequality  
linear programming  
Sugihara  
strongly realizable  
Lichtenstein  
3SAT  
planar  
orthohedral  
histoscape  
complexity  
Kirosis  
Papadimitriou  
Sugihara

Here ‘( $j \in A$ ,  $j < A$ ,  $j > A$ )’ means ‘ $j$  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  $\hat{j} = (x_j, y_j, z_j)$ , 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  $j \in A \iff a_A x_j + b_A y_j + z_j + c_A = 0$ ;  $j < A \iff a_A x_j + b_A y_j + z_j + c_A \geq 1$ ;  $j > A \iff a_A x_j + b_A y_j + z_j + c_A \leq -1$ .

(Actually ‘ $> 0$ ’ and ‘ $< 0$ ’ were expected here instead of ‘ $\geq 1$ ’ and ‘ $\leq -1$ ’; 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  $60^\circ$ . Furthermore, the two entries of Table 1 for which a V junction has a + label can arise only for  $60^\circ$  angles; the other four possibilities for V can arise only for  $120^\circ$  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).)

**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  $\alpha$  runs through  $G$ 's automorphisms. If those labelings aren't distinct, there are automorphisms  $\alpha$  and  $\beta$  for which  $l(v\alpha) = m - l(v\beta)$  for all  $v$ . But then  $\beta^{-1}\alpha$  would be an automorphism satisfying  $l(v\beta^{-1}\alpha) = m - l(v)$ ; 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  $\{0, \dots, m\}$  and that  $l(v) = v$  for  $0 \leq v \leq m$ . 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  $l(\text{NY}) > l(\text{MA})$  or  $l(\text{GA}) > l(\text{SC})$ . (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.**  $\text{LO}'[l] = m - l - \text{LO}[l]$ ,  $\text{NAME}'[l] = \text{NAME}[m - l]$ ,  $\text{FIRST}'[l] = m - \text{FIRST}[m - l]$ , for  $0 \leq l \leq m$ ;  $\text{NEXTL}'[l] = m - \text{NEXTH}[l]$ ,  $\text{NEXTH}'[l] = m - \text{NEXTL}[l]$ , for  $1 \leq l \leq m$ ; but change  $m + 1$  to  $-1$ . (Other settings of  $\text{FIRST}'$ ,  $\text{NEXTL}'$ ,  $\text{NEXTH}'$  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  $\text{LO}[m-3] = 2$ . That forces  $\text{LO}[m-4] = 0$ , leaving no choices for  $\text{LO}[m-5]$ .

**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  $\text{MATE}[l] = l$ ; if  $l$  is chosen and the endpoint of a partial subpath, let  $\text{MATE}[l]$  be the other endpoint; otherwise let  $\text{MATE}[l]$  be the bitwise complement of the value it had when it was most recently an endpoint, during the backtracking. For example, the  $\text{MATE}$  table ( $\text{MATE}[0], \dots, \text{MATE}[5]$ ) 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  $\text{MATE}[l] \leftarrow l$  for  $0 \leq l < n$ , 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  $\text{LO}[n - l] = v$ .] Set  $w \leftarrow v + n - l$ ,  $v' \leftarrow \text{MATE}[v]$ ,  $w' \leftarrow \text{MATE}[w]$ . Go to P4 if  $v' < 0$  or  $w' < 0$  or  $v' = w$ . Otherwise set  $\text{LO}[n - l] \leftarrow v$ ,  $\text{MATE}[v] \leftarrow \sim v'$ ,  $\text{MATE}[w] \leftarrow \sim w'$ ,  $\text{MATE}[v'] \leftarrow w'$ ,  $\text{MATE}[w'] \leftarrow v'$ ,  $l \leftarrow l + 1$ , 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  $v \leftarrow \text{LO}[n - l]$ ,  $w \leftarrow v + n - l$ ,  $v' \leftarrow \text{MATE}[v]$ ,  $w' \leftarrow \text{MATE}[w]$ . If  $v' \geq 0$  set  $\text{MATE}[v] \leftarrow v$ ; otherwise set  $\text{MATE}[v] \leftarrow \sim v'$  and  $\text{MATE}[\sim v'] \leftarrow v$ . If  $w' \geq 0$  set  $\text{MATE}[w] \leftarrow w$ ; otherwise set  $\text{MATE}[w] \leftarrow \sim w'$  and  $\text{MATE}[\sim w'] \leftarrow w$ . Return to P4. ■

**73.** A "blurred state" is obtained from  $\text{MATE}$  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  $(v_p, \beta_p, o_p)$  for  $p = 1, 2, \dots$ ; here  $v_p$  is a value of  $\text{LO}$ ;  $\beta_p$  is the blurred state if  $v_p$  is chosen; and  $o_p$  is the branch when  $v_p$  isn't. If  $\alpha$  represents a blurred state,  $\text{FIRST}(\alpha)$  represents its first branch and  $\text{LOC}(\alpha)$  represents the corresponding output node. Both  $\text{FIRST}$  and  $\text{LOC}$  are 0 unless changed.

In step P1, set  $p \leftarrow 0$  and  $\alpha_1$  to the initial blurred state.

In step P2, "visit" a solution by setting  $\text{LOC}(\alpha_l) \leftarrow 1$ .

isolated vertices  
Domain consistency  
bitwise complement  
blurred state

At the end of step P3, do the following just before returning to P2: Set  $\alpha_l$  to the current blurred state, and set  $p \leftarrow p + 1$ ,  $v_p \leftarrow v$ ,  $\beta_p \leftarrow \alpha_l$ ,  $o_p \leftarrow \text{FIRST}(\alpha_{l-1})$ , and  $\text{FIRST}(\alpha_{l-1}) \leftarrow p$ . If  $\alpha_l$  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  $z \leftarrow 2$ ,  $s \leftarrow 1$ ,  $\beta_1 \leftarrow \alpha_1$ ,  $o_1 \leftarrow 0$ ,  $\text{LOC}(\alpha_1) \leftarrow z$ . While  $s \neq 0$  do the following: “Set  $p \leftarrow \text{LOC}(\beta_s)$ ,  $q \leftarrow \text{FIRST}(\beta_s)$ ,  $s \leftarrow o_s$ . While  $q \neq 0$  do the following: “Set  $q' \leftarrow o_q$ ,  $\alpha \leftarrow \beta_q$ . If  $\text{LOC}(\alpha) = 0$  and  $\text{FIRST}(\alpha) \neq 0$ , set  $o_q \leftarrow s$ ,  $s \leftarrow q$ ,  $\text{LOC}(\alpha) \leftarrow z$ ,  $z \leftarrow z + 1$ . If  $q' \neq 0$ , output  $I_p = (\bar{v}_q? z: \text{LOC}(\alpha))$  and set  $p \leftarrow z$ ,  $z \leftarrow z + 1$ ; otherwise output  $I_p = (\bar{v}_q? 0: \text{LOC}(\alpha))$ . 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  $I_p = (\bar{v}? 0: 0)$ . 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  $p_1 \dots p_n = (x_1, 0) \dots (x_n, 0)$  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  $1 < i < j < n$ :  $p_1 \dots p_{i-1} p_j \dots p_i p_{j+1} \dots p_2 p_n$  would be longer.

Let  $S$  be the  $\lfloor n/2 \rfloor$  points with *smallest*  $x$ 's, and let  $T$  be the other  $\lfloor n/2 \rfloor$  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  $S = \{x_2, x_4, \dots, x_{2\lfloor n/2 \rfloor}\}$ .

The maximum path length is therefore  $(x_1 - x_2) + (x_3 - x_2) + (x_3 - x_4) + \dots = 2\sum T - 2\sum S - x_1 + x_n$  [ $n$  even], where  $x_1$  is the smallest  $x$  in  $T$  and  $x_n$  is the largest  $x$  in  $S$ .

Similarly, the longest *cycle*  $(p_1 \dots p_n)$  has length  $2\sum T - 2\sum S - 2x_1$  [ $n$  odd].

(b) A graceful permutation with  $p_n = p_1 + m$  yields a cycle  $(p_1 \dots p_n)$  of length  $1 + \dots + (2m-1) + (p_n - p_1) = 2m^2$ , which is maximum because  $\sum T - \sum S = m^2$ .

(c) The path length is  $2\sum T - p_n - 2\sum S + p_1 = 1 + \dots + (2m-1) = 2m^2 - m$ .

**75.** There are  $m = 2n + 1$  edges. Call  $S$  a  $(d, m)$ -set if  $S \cup \{|k - d| \mid k \in S\} \cup \{d\} = \{1, \dots, m\}$ . 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  $m-2$ ,  $\dots$ ,  $\lfloor m/2 \rfloor$  or  $\lceil m/2 \rceil$ .

There's no  $(d, m)$ -set when  $\lceil m/2 \rceil < d < m$ . For  $S$  would have to contain  $m$ ,  $m-1$ ,  $\dots$ ,  $d+1$ , and then there'd be no way to get edge  $d-1$ .

There's a unique  $(\lceil m/2 \rceil, m)$ -set, namely  $S = \{m, m-1, \dots, \lceil m/2 \rceil + 1\}$ .

breadth-first vs depth-first  
depth-first vs breadth-first  
Adamaszek  
Historical notes  
Abrham  
Kotzig  
Kløve  
McGill  
Ollis  
longest *cycle*

Finally, if  $d < \lceil m/2 \rceil$ , a  $(d, m)$ -set  $S$  must be  $\{m, m-1, \dots, m-d+1\} \cup S'$ , where  $S'$  is a  $(d, m-2d+2)$ -set. (An interesting recursion!)

So the total number of solutions is  $\sum_{d \setminus m} 2^{(d-1)/2} + \sum_{d \setminus n+1} 1 - 2^{n-1} - 1 - 2[n=1]$ .

**76.** If  $1 \text{ --- } m$ , change each  $x_{ij}$  to  $m - x_{ij}$ . Then if  $\min\{x_{11}, \dots, x_{n1}\} > \min\{x_{1r}, \dots, x_{nr}\}$ , change each  $x_{ij}$  to  $x_{i(r+1-j)}$ . Finally, sort the rows so that  $x_{11} < \dots < x_{n1}$ .

**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.]

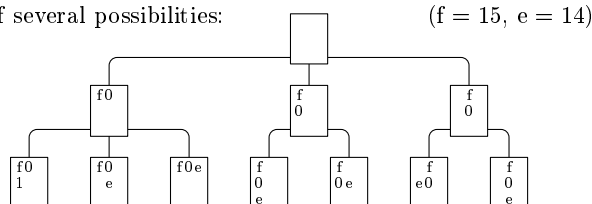
*Historical notes:* K. E. Petrie and B. M. Smith studied  $K_n \square P_2$  for  $n \leq 5$ , in order to test several strategies that exploit symmetry in instances of CSP [Lecture Notes in Computer Science **2833** (2003), 930–934]. Their methods were significantly improved by B. M. Smith and J.-F. 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  $\{0, 1, \dots, l\}$  and  $\{m, m-1, \dots, m-l\}$ , 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:



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

$$\begin{pmatrix} 31 & 41 & 59 & 26 & 53 & 58 & 97 & 93 & 23 & 84 & 62 & 64 & 33 & 83 & 27 & 95 & 02 \\ 25 & 71 & 19 & 77 & 17 & 86 & 08 & 81 & 65 & 91 & 37 & 99 & 01 & 98 & 04 & 87 & 22 \\ 68 & 24 & 10 & 29 & 74 & 45 & 07 & 18 & 89 & 05 & 96 & 00 & 88 & 03 & 80 & 13 & 94 \end{pmatrix}.$$

recursion  
breaks reflection symmetry  
Historical notes  
Petrie  
Smith  
symmetry  
Smith  
Puget  
KP  
KC  
unique  
KC graphs

[Smaller examples are also of interest. Consider, for example,

$$\begin{pmatrix} 3 & 14 & 15 & 9 & 26 \\ 22 & 21 & 0 & 27 & 13 \\ 7 & 2 & 25 & 1 & 4 \end{pmatrix}, \begin{pmatrix} 3 & 14 & 15 & 9 & 26 & 5 \\ 32 & 0 & 33 & 2 & 11 & 7 \\ 19 & 22 & 10 & 29 & 1 & 31 \end{pmatrix}, \begin{pmatrix} 3 & 14 & 15 & 9 & 26 & 5 & 35 \\ 12 & 0 & 39 & 1 & 30 & 20 & 4 \\ 39 & 36 & 2 & 34 & 7 & 25 & 32 \end{pmatrix},$$

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  $0 \leq k < \lceil r/2 \rceil$ ;  $m - x_{1(2k)} = x_{3(2k)} = 4k - 2$  and  $x_{2(2k)} = 8k - 3$  for  $1 \leq k \leq \lfloor r/2 \rfloor$ ; here  $m = 6r - 3$  is always odd. (These values are distinct; for example, the even numbers among them are  $\{0, 2, \dots, 2r - 2\}$  together with about  $1/4$  of the larger even numbers  $\leq 6r - 2$ .)

The differences between rows 2 and 3 give the odd edges  $\{1, 3, \dots, 2r - 1\}$ . 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  $1 \leq r \leq 14$  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  $\leftrightarrow$  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  $3 \text{ G}\mu$  to discover this, with a search tree of 12 million nodes. It needs 0.76 and 190 T $\mu$  to prove that  $K_5 \square K_4$  and  $K_5 \square K_5$  are *not* graceful.)

**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  $\leq 5$  vertices *is* graceful; see the list following Theorem S.)

**90.** (a) This is the mixed-radix representation  $\pi = [{}^3; \begin{smallmatrix} a_1 & a_2 & a_3 & \dots \\ 1 & 2 & 3 & 4 & \dots \end{smallmatrix}]$ ; 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  $(a_1, \dots, a_{20}) = (0, 0, 3, 1, 5, 6, 5, 0, 1, 4, 7, 8, 0, 6, 7, 10, 7, 10, 4, 10)$  [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)  $\chi(G_m^\pi) = 2$  for  $m \in \{1, 2, 3, 9, 10, 12, 15, 17\}$ ;  $\chi(G_m^\pi) = 3$  for the other  $m \leq 20$ .

[From this data we might be tempted to conjecture that a “random graceful labeling,” with  $m \rightarrow \infty$  edges, is a.s. planar, and 3-colorable. But F. Stappers has studied  $G_m^\pi$  for  $m \leq 10000$ , and found them *nonplanar* for  $m = 33, 38, 41, 44, 46$ –49, 51–52, 54–56, 58–61, and all cases  $\geq 63$ . 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  $\frac{\text{connected}}{\text{total}} = (\frac{864}{864}, \frac{1141312}{1141312}, \frac{159551124}{159601936}, \frac{6537511962}{6562523206}, \frac{106698003000}{108536168696},$

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  $7 \leq n \leq 17$ . (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  $\frac{8}{8}, \frac{22242}{8}, \frac{6317382}{6318302}, \frac{427805408}{428781978}, \frac{10110694366}{10233657368}, \frac{99592576642}{103635506314}, \frac{432843270752}{479912612982}, \frac{796114433250}{1009922060716}, \frac{516439259812}{876211145722}, \frac{67540932632}{234013536424}$ , for  $8 \leq n \leq 17$ ; 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.

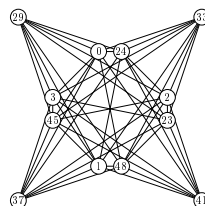
[When the labelings are also restricted to be  $\alpha$ -graceful, in the sense of exercise 111, the results become  $\frac{6}{6}, \frac{6840}{6840}, \frac{1855942}{1856280}, \frac{124467512}{124746754}, \frac{2945525928}{2980811422}, \frac{29277794448}{30452911120}, \frac{128904318498}{142798046522}, \frac{240333763962}{304499321272}, \frac{157722174046}{267381496426}, \frac{20772768256}{72154842584}$ ; 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](http://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  $m = 3, 4, \dots, 16$ , with  $0 \text{ --- } (m-1)$ , is 1, 1, 0, 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  $[z^m] 1 / \prod_{n \geq 3} (1 - z^n)$  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  $t \geq 1$ . 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  $C_3 \oplus C_5 \oplus \dots \oplus C_{2t+1}$  is graceful, for all  $t \geq 1$ . 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  
 $\alpha$ -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  $2 \leq t \leq 7$ . The method of exercise 93 rapidly gives us graceful labelings galore, respectively (1, 5, 222, 22806, 2988280, 641731574) of them.

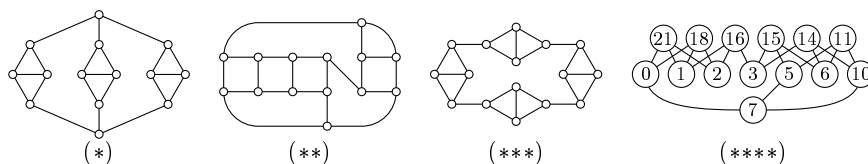
The main difficulty is to group them efficiently into equivalence classes of isomorphic graphs. One good way is to compute a “hash code”  $h(G)$  for each graph  $G$ . Let  $r_1, r_2, \dots$  be pseudorandom integers in the range  $0 \leq r_j < 2^{30}$ , and let  $r_0 = 0$ . For each vertex  $v$ , compute  $h(v)$  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  $V_k(v) \neq \emptyset$ . Set  $t_w \leftarrow 2r_k + 1$  for each  $w \in V_k(v)$ . Then, for  $k = d(v), d(v) - 1, \dots, 0$ , compute  $t'_w = t_w \prod_{u \sim w} (2r_{2d(v)+1-k} - t_u)$  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  $h(G) = (\sum_v [h(v)/2]) \bmod 2^{32}$ , 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  $h(G) \neq h(H)$  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](http://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  $n \leq 4$ ; 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  $0 = l_0 < l_1 < \dots < l_{n-1} = m$  that are to be used. Then a cubic graceful labeling is the solution to the MCC problem whose primary items are  $\#1, \dots, \#m$  and  $l_0, \dots, l_{n-1}$ , where the  $l$ ’s have multiplicity 3; the options are simply ‘ $\#k l_i l_j$ ’ for  $0 \leq i < j < n$ , 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  $2K_4, 3K_4, \dots$

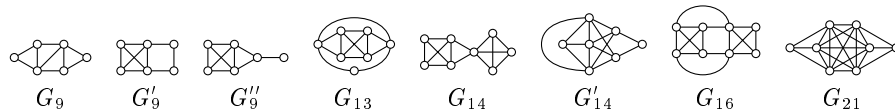
**97.** Backtracking via Theorem S, as in exercise 91, we can avoid most of the  $m!/2$  cases by allowing at most 8 of the vertices  $\{0, 1, \dots, m\}$  to touch an edge. Thus we readily discover that the (1, 2, 7, 23, 122, 888, 11302) distinct graphs with  $n = (2, 3, \dots, 8)$  nonisolated vertices 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.

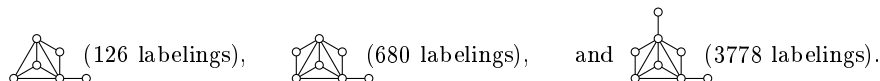
One of the seven graphs with  $n = 4$  nonisolated vertices,  $2K_2$ , doesn't have enough edges to be graceful. But the text points out that the other six work out fine (indeed, uniquely for  $K_{1,3}$ ,  $P_4$ ,  $C_4$ , and  $K_4$ , and up to 5 ways with the paw .

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  $K_1 \text{---} 2K_2$ . The other 18 are graceful:  $K_{1,4}$  uniquely, and the “dart”  $K_1 \text{---} (K_1 \oplus P_3)$  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  $K_1 \text{---} (2K_1 \text{---} 2K_2)$ . (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  $n = 8$ :  $K_2 \oplus C_6$ ,  $2K_2 \oplus K_{1,1,2}$ ,  $P_3 \oplus C_5$ ,  $C_3 \oplus P_5$ ,  $K_{1,3} \oplus L_{3,1}$ ,  $2K_2 \oplus K_4$ ,  $K_3 \oplus L_{4,1}$ ,  $K_{1,3} \oplus K_4$ ,  $P_3 \oplus K_5$ ,  $K_3 \oplus (P_2 \text{---} P_3)$ . And the 19 connected ones likewise have lots of symmetry:  $K_{1,7}$ ,  $G_{14}$ ,  $2K_1 \text{---} 3K_2$ ,  $G_{16}$ ,  $4K_1 \text{---} 2K_2$ ,  $2K_1 \text{---} (K_2 \oplus K_4)$ ,  $K_2 \text{---} 2K_3$ ,  $K_1 \text{---} G_{13}$ ,  $K_1 \text{---} (2K_1 \text{---} (K_2 \oplus K_3))$ ,  $2K_1 \text{---} K_{3,3}$ ,  $K_3 \text{---} (K_1 \oplus 2K_2)$ ,  $K_3 \text{---} (P_2 \oplus P_3)$ ,  $K_3 \text{---} (2K_1 \oplus K_3)$ ,  $G_{21}$ ,  $K_1 \text{---} G'_{14}$ ,  $2K_1 \text{---} G_9$ ,  $K_2 \text{---} G'_9$ ,  $K_2 \text{---} G''_9$ ,  $K_3 \text{---} (K_1 \oplus C_4)$ , where  $G_m$  or  $G'_m$  or  $G''_m$  denotes a special graph with  $m$  edges:



The champions for gracefulness with 6, 7, and 8 vertices are



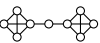

**98.** (a) No, because edge 11 ( $3 \text{---} 14$ ,  $\text{NC} \text{---} \text{SC}$ ) doesn't touch edges 12–18 (see (33)).  
(b) 11067 (including the solution to Fig. 105(d)).

(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 ( $K_1 \text{---} 2K_2$ )  
dart  
 $K_n$   
 $K_{m,n}$   
 $P_n$   
 $C_n$   
 $L_{m,n}$   
lollipop graph  
bridge  
paw

1132397252122, 1296227076156, 605872421102, 94984144008, 2895168460) cases, for  $7 \leq n \leq 17$ . 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  $4K_1 \text{ --- } 2K_2$ . (And the only examples with 9 vertices are , ,  $K_1 \text{ --- } \text{graph}$ , and  $K_1 \text{ --- } (2K_1 \text{ --- } (K_2 \oplus (2K_1 \text{ --- } K_2)))$ ; these are just four of the 259614 connected graceful graphs. The first of these is the only example with at most 14 edges.)

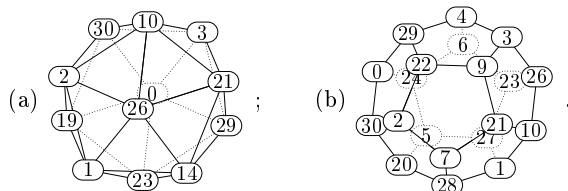
**100.** After numerous experiments, the author's most successful attempt is a backtrack program called BACK-GRACEFUL, based on Algorithm 7.2.2W (and available online). It uses a sparse-set structure to keep a list of all vertices, with labeled vertices at the left. To enable efficient bitwise tests, 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  $(j, k) = (0, q), (1, q+1), \dots, (m-q, m)$  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  $v \leftarrow lw[j]$ , and for all  $v \text{ --- } w$  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  $0 \text{ --- } 30 \text{ --- } 29 \text{ --- } 0$  and 19 have the induced path  $30 \text{ --- } 0 \text{ --- } 29$ . All are rooted except for the solution shown.



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  $[0 \dots 10] \cup [20 \dots 30]$ . 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  $L0[k] \leftarrow x$ . 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  $\leq g$  components. The largest good vertices in the other good components are OK.

(d) The vertices  $\{r, r+k, \dots, r+pk\}$  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  $g \leq k-2(k+1-G)$  and we have  $G-g \geq k+2-G$ . If  $G \leq \frac{2}{3}(k+2)$  we have  $G-g \geq (k+2)/3$ ; otherwise either  $g$  or  $G-g$  is  $\geq G/2 > (k+2)/3$ . 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  $m \leq 20$  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  $k \geq m/2$ , all edges connect small to big. The hint follows because the cycle containing  $x$  and  $x+k$  includes the edges  $y \text{---} (x+k) \text{---} x \text{---} z$ .

Let there be  $c$  unusable vertices, in  $C$  components. A component that contains  $q > 0$  unusable vertices  $x_1 < \dots < x_q$  therefore contains at least the  $2q+2$  vertices  $y_1 < x_1 < \dots < x_q < x_1+k < \dots < x_q+k < z_q$ , and it contains at least  $2q+1$  of the  $m-k$  edges. Consequently  $m-k \geq 2c+C$ ; and the number of usable vertices is  $m+1-k-c \geq (m-k)/2+1+C/2 \geq 2+\lfloor (m-k)/2 \rfloor$ , unless  $C=c=0$ .

[Altogether we get the superexponential lower bound  $t_1 \dots t_m = \Omega(m!/24^{m/2})$ .]

**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  $s_2 \dots s_{n-1} S(1, s_2, \dots, s_{n-1})$  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  $[x_1 x_2 x_3] S(x_1, x_2, x_3) = (S(1, 1, 1) - S(1, 1, -1) - S(1, -1, 1) + S(1, -1, -1))/4$ . [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,  $\tau(n) = \exp(n \ln n - n \ln(e/b) + O(\log n))$ . 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  $n \leq 6$  in 1984.)

**107.** Suppose  $1 \leq e < 2^n$ , where  $2^n + e = (e_n \dots e_1 e_0)_2$ . Then the edge labeled  $e$  is between  $x = (x_n \dots x_1 x_0)_2$  and  $x \& (x-1)$ , if  $e_k = 1$  and  $e_{k-1} = \dots = e_0 = 0$  and  $x_j = e_j \oplus [j > k]e_{j+1}$  for  $0 \leq j < n$ . (This is in fact an  $\alpha$ -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  
 $\alpha$ -labeling  
Gray code  
automorphisms

**109.** (a) Suppose  $\alpha$  has even parity and  $\beta$  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  $(a_0, a_1, \dots) = (0, 1, 3, 4, 8, 9, \dots)$ , and we have  $a_n = \sum_{k=1}^n 2^{\rho_k}$  for all  $n$ . (It can be shown that  $a_n = n + (e_1 2^{e_1} + \dots + e_t 2^{e_t})/2$  when  $n = 2^{e_1} + \dots + 2^{e_t}$  with  $e_1 > \dots > e_t \geq 0$ .) By part (a),  $L_0 = L_1 + 2^{n-2} + 2a_{2^{n-2}} = L_1 + (n+1)2^{n-2}$ . The other edges  $0\alpha - 1\alpha$  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  $L_1 = \{1, 2, \dots, (n-1)2^{n-2}\}$  by induction; and it all works,  $\alpha$ -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  $n-r-1$  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,  $\dots$  (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  $\alpha$ -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} - 3 - \overline{4} - 4 - \overline{5} - 5 - \overline{6} - 6 - \overline{7} - 7 - \overline{8} - 8 - \overline{9} - 9 - \overline{10} - 10 - \overline{11} - 11 - \overline{12} - 12 - \overline{13} - 13 - \overline{14} - 14 - \overline{15} - 15 - \overline{16} - 16 - \overline{17} - 17 - \overline{18} - 18 - \overline{19} - 19 - \overline{20} - 20 - \overline{21} - 21 - \overline{22} - 22 - \overline{23} - 23 - \overline{24} - 24 - \overline{25} - 25 - \overline{26} - 26 - \overline{27} - 27 - \overline{28} - 28 - \overline{29} - 29 - \overline{30} - 30 - \overline{31} - 31 - \overline{32} - 32 - \overline{33} - 33 - \overline{34} - 34 - \overline{35} - 35 - \overline{36} - 36 - \overline{37} - 37 - \overline{38} - 38 - \overline{39} - 39 - \overline{40} - 40 - \overline{41} - 41 - \overline{42} - 42 - \overline{43} - 43 - \overline{44} - 44 - \overline{45} - 45 - \overline{46} - 46 - \overline{47} - 47 - \overline{48} - 48 - \overline{49} - 49 - \overline{50} - 50 - \overline{51} - 51 - \overline{52} - 52 - \overline{53} - 53 - \overline{54} - 54 - \overline{55} - 55 - \overline{56} - 56 - \overline{57} - 57 - \overline{58} - 58 - \overline{59} - 59 - \overline{60} - 60 - \overline{61} - 61 - \overline{62} - 62 - \overline{63} - 63 - \overline{64} - 64 - \overline{65} - 65 - \overline{66} - 66 - \overline{67} - 67 - \overline{68} - 68 - \overline{69} - 69 - \overline{70} - 70 - \overline{71} - 71 - \overline{72} - 72 - \overline{73} - 73 - \overline{74} - 74 - \overline{75} - 75 - \overline{76} - 76 - \overline{77} - 77 - \overline{78} - 78 - \overline{79} - 79 - \overline{80} - 80 - \overline{81} - 81 - \overline{82} - 82 - \overline{83} - 83 - \overline{84} - 84 - \overline{85} - 85 - \overline{86} - 86 - \overline{87} - 87 - \overline{88} - 88 - \overline{89} - 89 - \overline{90} - 90 - \overline{91} - 91 - \overline{92} - 92 - \overline{93} - 93 - \overline{94} - 94 - \overline{95} - 95 - \overline{96} - 96 - \overline{97} - 97 - \overline{98} - 98 - \overline{99} - 99 - \overline{100} - 100 - \overline{101} - 101 - \overline{102} - 102 - \overline{103} - 103 - \overline{104} - 104 - \overline{105} - 105 - \overline{106} - 106 - \overline{107} - 107 - \overline{108} - 108 - \overline{109} - 109 - \overline{110} - 110 - \overline{111} - 111 - \overline{112} - 112 - \overline{113} - 113 - \overline{114} - 114 - \overline{115} - 115 - \overline{116} - 116 - \overline{117} - 117 - \overline{118} - 118 - \overline{119} - 119 - \overline{120} - 120 - \overline{121} - 121 - \overline{122} - 122 - \overline{123} - 123 - \overline{124} - 124 - \overline{125} - 125 - \overline{126} - 126 - \overline{127} - 127 - \overline{128} - 128 - \overline{129} - 129 - \overline{130} - 130 - \overline{131} - 131 - \overline{132} - 132 - \overline{133} - 133 - \overline{134} - 134 - \overline{135} - 135 - \overline{136} - 136 - \overline{137} - 137 - \overline{138} - 138 - \overline{139} - 139 - \overline{140} - 140 - \overline{141} - 141 - \overline{142} - 142 - \overline{143} - 143 - \overline{144} - 144 - \overline{145} - 145 - \overline{146} - 146 - \overline{147} - 147 - \overline{148} - 148 - \overline{149} - 149 - \overline{150} - 150 - \overline{151} - 151 - \overline{152} - 152 - \overline{153} - 153 - \overline{154} - 154 - \overline{155} - 155 - \overline{156} - 156 - \overline{157} - 157 - \overline{158} - 158 - \overline{159} - 159 - \overline{160} - 160 - \overline{161} - 161 - \overline{162} - 162 - \overline{163} - 163 - \overline{164} - 164 - \overline{165} - 165 - \overline{166} - 166 - \overline{167} - 167 - \overline{168} - 168 - \overline{169} - 169 - \overline{170} - 170 - \overline{171} - 171 - \overline{172} - 172 - \overline{173} - 173 - \overline{174} - 174 - \overline{175} - 175 - \overline{176} - 176 - \overline{177} - 177 - \overline{178} - 178 - \overline{179} - 179 - \overline{180} - 180 - \overline{181} - 181 - \overline{182} - 182 - \overline{183} - 183 - \overline{184} - 184 - \overline{185} - 185 - \overline{186} - 186 - \overline{187} - 187 - \overline{188} - 188 - \overline{189} - 189 - \overline{190} - 190 - \overline{191} - 191 - \overline{192} - 192 - \overline{193} - 193 - \overline{194} - 194 - \overline{195} - 195 - \overline{196} - 196 - \overline{197} - 197 - \overline{198} - 198 - \overline{199} - 199 - \overline{200} - 200 - \overline{201} - 201 - \overline{202} - 202 - \overline{203} - 203 - \overline{204} - 204 - \overline{205} - 205 - \overline{206} - 206 - \overline{207} - 207 - \overline{208} - 208 - \overline{209} - 209 - \overline{210} - 210 - \overline{211} - 211 - \overline{212} - 212 - \overline{213} - 213 - \overline{214} - 214 - \overline{215} - 215 - \overline{216} - 216 - \overline{217} - 217 - \overline{218} - 218 - \overline{219} - 219 - \overline{220} - 220 - \overline{221} - 221 - \overline{222} - 222 - \overline{223} - 223 - \overline{224} - 224 - \overline{225} - 225 - \overline{226} - 226 - \overline{227} - 227 - \overline{228} - 228 - \overline{229} - 229 - \overline{230} - 230 - \overline{231} - 231 - \overline{232} - 232 - \overline{233} - 233 - \overline{234} - 234 - \overline{235} - 235 - \overline{236} - 236 - \overline{237} - 237 - \overline{238} - 238 - \overline{239} - 239 - \overline{240} - 240 - \overline{241} - 241 - \overline{242} - 242 - \overline{243} - 243 - \overline{244} - 244 - \overline{245} - 245 - \overline{246} - 246 - \overline{247} - 247 - \overline{248} - 248 - \overline{249} - 249 - \overline{250} - 250 - \overline{251} - 251 - \overline{252} - 252 - \overline{253} - 253 - \overline{254} - 254 - \overline{255} - 255 - \overline{256} - 256 - \overline{257} - 257 - \overline{258} - 258 - \overline{259} - 259 - \overline{260} - 260 - \overline{261} - 261 - \overline{262} - 262 - \overline{263} - 263 - \overline{264} - 264 - \overline{265} - 265 - \overline{266} - 266 - \overline{267} - 267 - \overline{268} - 268 - \overline{269} - 269 - \overline{270} - 270 - \overline{271} - 271 - \overline{272} - 272 - \overline{273} - 273 - \overline{274} - 274 - \overline{275} - 275 - \overline{276} - 276 - \overline{277} - 277 - \overline{278} - 278 - \overline{279} - 279 - \overline{280} - 280 - \overline{281} - 281 - \overline{282} - 282 - \overline{283} - 283 - \overline{284} - 284 - \overline{285} - 285 - \overline{286} - 286 - \overline{287} - 287 - \overline{288} - 288 - \overline{289} - 289 - \overline{290} - 290 - \overline{291} - 291 - \overline{292} - 292 - \overline{293} - 293 - \overline{294} - 294 - \overline{295} - 295 - \overline{296} - 296 - \overline{297} - 297 - \overline{298} - 298 - \overline{299} - 299 - \overline{300} - 300 - \overline{301} - 301 - \overline{302} - 302 - \overline{303} - 303 - \overline{304} - 304 - \overline{305} - 305 - \overline{306} - 306 - \overline{307} - 307 - \overline{308} - 308 - \overline{309} - 309 - \overline{310} - 310 - \overline{311} - 311 - \overline{312} - 312 - \overline{313} - 313 - \overline{314} - 314 - \overline{315} - 315 - \overline{316} - 316 - \overline{317} - 317 - \overline{318} - 318 - \overline{319} - 319 - \overline{320} - 320 - \overline{321} - 321 - \overline{322} - 322 - \overline{323} - 323 - \overline{324} - 324 - \overline{325} - 325 - \overline{326} - 326 - \overline{327} - 327 - \overline{328} - 328 - \overline{329} - 329 - \overline{330} - 330 - \overline{331} - 331 - \overline{332} - 332 - \overline{333} - 333 - \overline{334} - 334 - \overline{335} - 335 - \overline{336} - 336 - \overline{337} - 337 - \overline{338} - 338 - \overline{339} - 339 - \overline{340} - 340 - \overline{341} - 341 - \overline{342} - 342 - \overline{343} - 343 - \overline{344} - 344 - \overline{345} - 345 - \overline{346} - 346 - \overline{347} - 347 - \overline{348} - 348 - \overline{349} - 349 - \overline{350} - 350 - \overline{351} - 351 - \overline{352} - 352 - \overline{353} - 353 - \overline{354} - 354 - \overline{355} - 355 - \overline{356} - 356 - \overline{357} - 357 - \overline{358} - 358 - \overline{359} - 359 - \overline{360} - 360 - \overline{361} - 361 - \overline{362} - 362 - \overline{363} - 363 - \overline{364} - 364 - \overline{365} - 365 - \overline{366} - 366 - \overline{367} - 367 - \overline{368} - 368 - \overline{369} - 369 - \overline{370} - 370 - \overline{371} - 371 - \overline{372} - 372 - \overline{373} - 373 - \overline{374} - 374 - \overline{375} - 375 - \overline{376} - 376 - \overline{377} - 377 - \overline{378} - 378 - \overline{379} - 379 - \overline{380} - 380 - \overline{381} - 381 - \overline{382} - 382 - \overline{383} - 383 - \overline{384} - 384 - \overline{385} - 385 - \overline{386} - 386 - \overline{387} - 387 - \overline{388} - 388 - \overline{389} - 389 - \overline{390} - 390 - \overline{391} - 391 - \overline{392} - 392 - \overline{393} - 393 - \overline{394} - 394 - \overline{395} - 395 - \overline{396} - 396 - \overline{397} - 397 - \overline{398} - 398 - \overline{399} - 399 - \overline{400} - 400 - \overline{401} - 401 - \overline{402} - 402 - \overline{403} - 403 - \overline{404} - 404 - \overline{405} - 405 - \overline{406} - 406 - \overline{407} - 407 - \overline{408} - 408 - \overline{409} - 409 - \overline{410} - 410 - \overline{411} - 411 - \overline{412} - 412 - \overline{413} - 413 - \overline{414} - 414 - \overline{415} - 415 - \overline{416} - 416 - \overline{417} - 417 - \overline{418} - 418 - \overline{419} - 419 - \overline{420} - 420 - \overline{421} - 421 - \overline{422} - 422 - \overline{423} - 423 - \overline{424} - 424 - \overline{425} - 425 - \overline{426} - 426 - \overline{427} - 427 - \overline{428} - 428 - \overline{429} - 429 - \overline{430} - 430 - \overline{431} - 431 - \overline{432} - 432 - \overline{433} - 433 - \overline{434} - 434 - \overline{435} - 435 - \overline{436} - 436 - \overline{437} - 437 - \overline{438} - 438 - \overline{439} - 439 - \overline{440} - 440 - \overline{441} - 441 - \overline{442} - 442 - \overline{443} - 443 - \overline{444} - 444 - \overline{445} - 445 - \overline{446} - 446 - \overline{447} - 447 - \overline{448} - 448 - \overline{449} - 449 - \overline{450} - 450 - \overline{451} - 451 - \overline{452} - 452 - \overline{453} - 453 - \overline{454} - 454 - \overline{455} - 455 - \overline{456} - 456 - \overline{457} - 457 - \overline{458} - 458 - \overline{459} - 459 - \overline{460} - 460 - \overline{461} - 461 - \overline{462} - 462 - \overline{463} - 463 - \overline{464} - 464 - \overline{465} - 465 - \overline{466} - 466 - \overline{467} - 467 - \overline{468} - 468 - \overline{469} - 469 - \overline{470} - 470 - \overline{471} - 471 - \overline{472} - 472 - \overline{473} - 473 - \overline{474} - 474 - \overline{475} - 475 - \overline{476} - 476 - \overline{477} - 477 - \overline{478} - 478 - \overline{479} - 479 - \overline{480} - 480 - \overline{481} - 481 - \overline{482} - 482 - \overline{483} - 483 - \overline{484} - 484 - \overline{485} - 485 - \overline{486} - 486 - \overline{487} - 487 - \overline{488} - 488 - \overline{489} - 489 - \overline{490} - 490 - \overline{491} - 491 - \overline{492} - 492 - \overline{493} - 493 - \overline{494} - 494 - \overline{495} - 495 - \overline{496} - 496 - \overline{497} - 497 - \overline{498} - 498 - \overline{499} - 499 - \overline{500} - 500 - \overline{501} - 501 - \overline{502} - 502 - \overline{503} - 503 - \overline{504} - 504 - \overline{505} - 505 - \overline{506} - 506 - \overline{507} - 507 - \overline{508} - 508 - \overline{509} - 509 - \overline{510} - 510 - \overline{511} - 511 - \overline{512} - 512 - \overline{513} - 513 - \overline{514} - 514 - \overline{515} - 515 - \overline{516} - 516 - \overline{517} - 517 - \overline{518} - 518 - \overline{519} - 519 - \overline{520} - 520 - \overline{521} - 521 - \overline{522} - 522 - \overline{523} - 523 - \overline{524} - 524 - \overline{525} - 525 - \overline{526} - 526 - \overline{527} - 527 - \overline{528} - 528 - \overline{529} - 529 - \overline{530} - 530 - \overline{531} - 531 - \overline{532} - 532 - \overline{533} - 533 - \overline{534} - 534 - \overline{535} - 535 - \overline{536} - 536 - \overline{537} - 537 - \overline{538} - 538 - \overline{539} - 539 - \overline{540} - 540 - \overline{541} - 541 - \overline{542} - 542 - \overline{543} - 543 - \overline{544} - 544 - \overline{545} - 545 - \overline{546} - 546 - \overline{547} - 547 - \overline{548} - 548 - \overline{549} - 549 - \overline{550} - 550 - \overline{551} - 551 - \overline{552} - 552 - \overline{553} - 553 - \overline{554} - 554 - \overline{555} - 555 - \overline{556} - 556 - \overline{557} - 557 - \overline{558} - 558 - \overline{559} - 559 - \overline{560} - 560 - \overline{561} - 561 - \overline{562} - 562 - \overline{563} - 563 - \overline{564} - 564 - \overline{565} - 565 - \overline{566} - 566 - \overline{567} - 567 - \overline{568} - 568 - \overline{569} - 569 - \overline{570} - 570 - \overline{571} - 571 - \overline{572} - 572 - \overline{573} - 573 - \overline{574} - 574 - \overline{575} - 575 - \overline{576} - 576 - \overline{577} - 577 - \overline{578} - 578 - \overline{579} - 579 - \overline{580} - 580 - \overline{581} - 581 - \overline{582} - 582 - \overline{583} - 583 - \overline{584} - 584 - \overline{585} - 585 - \overline{586} - 586 - \overline{587} - 587 - \overline{588} - 588 - \overline{589} - 589 - \overline{590} - 590 - \overline{591} - 591 - \overline{592} - 592 - \overline{593} - 593 - \overline{594} - 594 - \overline{595} - 595 - \overline{596} - 596 - \overline{597} - 597 - \overline{598} - 598 - \overline{599} - 599 - \overline{600} - 600 - \overline{601} - 601 - \overline{602} - 602 - \overline{603} - 603 - \overline{604} - 604 - \overline{605} - 605 - \overline{606} - 606 - \overline{607} - 607 - \overline{608} - 608 - \overline{609} - 609 - \overline{610} - 610 - \overline{611} - 611 - \overline{612} - 612 - \overline{613} - 613 - \overline{614} - 614 - \overline{615} - 615 - \overline{616} - 616 - \overline{617} - 617 - \overline{618} - 618 - \overline{619} - 619 - \overline{620} - 620 - \overline{621} - 621 - \overline{622} - 622 - \overline{623} - 623 - \overline{624} - 624 - \overline{625} - 625 - \overline{626} - 626 - \overline{627} - 627 - \overline{628} - 628 - \overline{629} - 629 - \overline{630} - 630 - \overline{631} - 631 - \overline{632} - 632 - \overline{633} - 633 - \overline{634} - 634 - \overline{635} - 635 - \overline{636} - 636 - \overline{637} - 637 - \overline{638} - 638 - \overline{639} - 639 - \overline{640} - 640 - \overline{641} - 641 - \overline{642} - 642 - \overline{643} - 643 - \overline{644} - 644 - \overline{645} - 645 - \overline{646} - 646 - \overline{647} - 647 - \overline{648} - 648 - \overline{649} - 649 - \overline{650} - 650 - \overline{651} - 651 - \overline{652} - 652 - \overline{653} - 653 - \overline{654} - 654 - \overline{655} - 655 - \overline{656} - 656 - \overline{657} - 657 - \overline{658} - 658 - \overline{659} - 659 - \overline{660} - 660 - \overline{661} - 661 - \overline{662} - 662 - \overline{663} - 663 - \overline{664} - 664 - \overline{665} - 665 - \overline{666} - 666 - \overline{667} - 667 - \overline{668} - 668 - \overline{669} - 669 - \overline{670} - 670 - \overline{671} - 671 - \overline{672} - 672 - \overline{673} - 673 - \overline{674} - 674 - \overline{675} - 675 - \overline{676} - 676 - \overline{677} - 677 - \overline{678} - 678 - \overline{679} - 679 - \overline{680} - 680 - \overline{681} - 681 - \overline{682} - 682 - \overline{683} - 683 - \overline{684} - 684 - \overline{685} - 685 - \overline{686} - 686 - \overline{687} - 687 - \overline{688} - 688 - \overline{689} - 689 - \overline{690} - 690 - \overline{691} - 691 - \overline{692} - 692 - \overline{693} - 693 - \overline{694} - 694 - \overline{695} - 695 - \overline{696} - 696 - \overline{697} - 697 - \overline{698} - 698 - \overline{699} - 699 - \overline{700} - 700 - \overline{701} - 701 - \overline{702} - 702 - \overline{703} - 703 - \overline{704} - 704 - \overline{705} - 705 - \overline{706} - 706 - \overline{707} - 707 - \overline{708} - 708 - \overline{709} - 709 - \overline{710} - 710 - \overline{711} - 711 - \overline{712} - 712 - \overline{713} - 713 - \overline{714} - 714 - \overline{715} - 715 - \overline{716} - 716 - \overline{717} - 717 - \overline{718} - 718 - \overline{719} - 719 - \overline{720} - 720 - \overline{721} - 721 - \overline{722} - 722 - \overline{723} - 723 - \overline{724} - 724 - \overline{725} - 725 - \overline{726} - 726 - \overline{727} - 727 - \overline{728} - 728 - \overline{729} - 729 - \overline{730} - 730 - \overline{731} - 731 - \overline{732} - 732 - \overline{733} - 733 - \overline{734} - 734 - \overline{735} - 735 - \overline{736} - 736 - \overline{737} - 737 - \overline{738} - 738 - \overline{739} - 739 - \overline{740} - 740 - \overline{741} - 741 - \overline{742} - 742 - \overline{743} - 743 - \overline{744} - 744 - \overline{745} - 745 - \overline{746} - 746 - \overline{747} - 747 - \overline{748} - 748 - \overline{749} - 749 - \overline{750} - 750 - \overline{751} - 751 - \overline{752} - 752 - \overline{753} - 753 - \overline{754} - 754 - \overline{755} - 755 - \overline{756} - 756 - \overline{757} - 757 - \overline{758} - 758 - \overline{759} - 759 - \overline{760} - 760 - \overline{761} - 761 - \overline{762} - 762 - \overline{763} - 763 - \overline{764} - 764 - \overline{765} - 765 - \overline{766} - 766 - \overline{767} - 767 - \overline{768} - 768 - \overline{769} - 769 - \overline{770} - 770 - \overline{771} - 771 - \overline{772} - 772 - \overline{773} - 773 - \overline{774} - 774 - \overline{775} - 775 - \overline{776} - 776 -$

also showed that the graph  $S_{n,2}$  with  $2n$  edges  $0 \text{ --- } a_j \text{ --- } b_j$  for  $1 \leq j \leq n$  has an ordered graceful labeling; that graph isn't  $\alpha$ -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  $2 \leq m \leq 8$  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  $1 \leq i \leq a, 1 \leq j \leq b, 1 \leq k \leq c, 1 \leq l \leq d$ . The edges are  $u_i x_k \text{ --- } v_j y_l, u_i y_l \text{ --- } v_j x_k$ , 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  $90^\circ$  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.

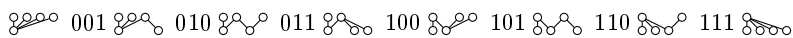
(ii) Now the board is  $(m+n) \times (m+n)$ , with  $n$  (not  $n-1$ ) rows/columns affected at the upper left or lower right. Again the two graphs are isomorphic by transposition.

(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  $x_1 \dots x_m y_1 \dots y_n$  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) 

(b) Use labels  $0, 1, \dots, s$  for  $u_0$  through  $u_s$  and  $\bar{0}, \bar{1}, \dots, \bar{t}$  for  $v_0$  through  $v_t$ .

(c) There are  $m_{s+t+1}$  edges, where  $m_0 = 0$  and  $m_{i+1} = m_i + p_{s_i} + q_{t_i} + 1$ .

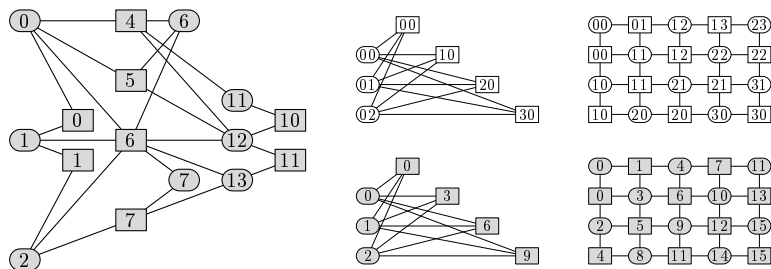
(d) Assign the label  $a_j + i$  to each element  $u_{ji}$  of  $U_j$ , and the label  $\overline{b_k + i}$  to each element  $v_{ki}$  of  $V_k$ , where  $a_0 = b_0 = 0$  and  $a_{j+1} = a_j + p_j + q_{j'} + 1, b_{k+1} = b_k + p_{k''} + q_k + 1$ ; here  $j' = \max\{i \mid u_j \text{ --- } v_i\}$  and  $k'' = \max\{i \mid u_i \text{ --- } v_k\}$ . 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  $m - m_i, m - m_i - 1, \dots, m - m_{i+1} + 1$ . 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  $u_0 \text{ --- } v_0, \dots, u_0 \text{ --- } v_{r-1}$ . Then set  $p_0 = n-1$  and  $q_i = 0$  for  $0 \leq i < r$ . (See the case (3, 4) below.)

(f) Denote the grid points by  $(x, y)$  for  $0 \leq x < r$  and  $0 \leq y < n$ . 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  $U_0 \text{ --- } V_0 \text{ --- } U_1 \text{ --- } V_1 \text{ --- } \dots$  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.)

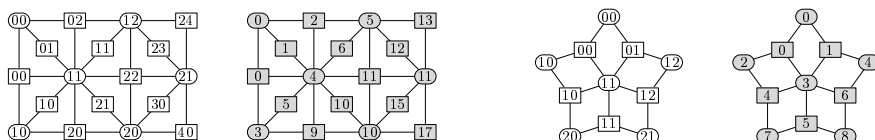
OEIS  
Sheppard  
chessboard  
Aztec diamonds  
generalized toruses  
parity  
Snevily  
Acharya  
Gill

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:



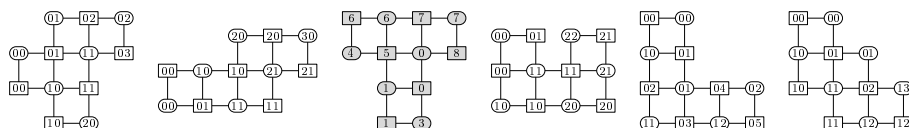
Acharya  
roots of unity  
factorization  
intervals

(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  $\alpha$ -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  $\{a_1, \dots, a_n\}$  and  $\{b_1, \dots, b_r\}$  are nonnegative integers whose  $nr$  sums  $a_i + b_j$  yield  $\{0, \dots, m-1\}$ .

(b) Because the  $m$ th roots of unity are  $e^{2\pi i k/m}$ , the complete factorization of  $(1-x^m)/(1-x)$  over the real numbers is  $(1+x)^{[m \text{ even}]} \cdot \prod_{k=1}^{[m/2]-1} (1 - 2x \cos \frac{2\pi k}{m} + x^2)$ . And any product of palindromials is a palindromial.

(c) Let  $G(x) = g_0 + \dots + g_c x^c$  and  $H(x) = h_0 + \dots + h_d x^d$ . 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  $g_{c-k}h_k + g_c h_0 \leq 1$ . But  $g_k h_0 + g_{k-1} h_1 + \dots + g_0 h_k = 1$ , 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  $[a_0 \dots a_0 + k_0) \cup [a_1 \dots a_1 + k_1) \cup \dots \cup [a_t \dots a_t + k_t)$ , 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  $[0 \dots m)$ .

Let  $k = k_0$ . Clearly  $h_k = 1$ . And we must have  $k_i \leq k$  for  $0 \leq i \leq t$ , 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  $T(x) = x^{a_0} + \dots + x^{a_t}$ .

(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  $k \mid n$  or some  $l \mid r$ . 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  $\pi$  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:  $p_1p_2q_1q_2 \equiv p_1p_2q_2q_1 \equiv p_2p_1q_1q_2 \equiv p_2p_1q_2q_1$ ,  $p_1q_1p_2q_2 \equiv p_1q_1q_2p_2 \equiv p_1q_2q_1p_2 \equiv p_2q_1p_1q_2$ ,  $p_2q_1q_2p_1 \equiv p_2q_2q_1p_1$ ,  $p_2q_2p_1q_1$ , and seven more with  $p \leftrightarrow q$ . So  $A(p_1p_2, q_1q_2) = 14$ .

(h) The Möbius polynomial for variables  $\{p_1, \dots, p_s, q_1, \dots, q_t\}$ , when the  $p$ 's and  $q$ 's commute pairwise, is  $(1 - p_1) \dots (1 - p_s) + (1 - q_1) \dots (1 - q_t) - 1$ .

(i)  $1/((1 - q_1) \dots (1 - q_t) - p) = \sum_{e \geq 0} p^e (1 - q_1)^{-1-e} \dots (1 - q_t)^{-1-e}$ .

[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  $A(p_1^e p_2^e, q_1^e q_2^e) = \sum_k (-1)^{e+k} \binom{2e}{k} (!)$ ; 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  $1 \leq j \leq \lfloor p/2 \rfloor$ ; 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  $\{jn + k \mid 1 \leq j < r\} \cup \{nr\}$ . 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  $n, r \leq 23$ , according to calculations by F. Stappers. (Is  $K_{n,n}$  uniquely graceful when  $n = 3k + 2$  is prime?)

**120.** Primary items  $\{1, \dots, m\}$  for the arc labels, and  $m$  primary items  $vw$  for the arcs  $v \rightarrow w$ . Also  $n$  secondary items  $v$  for the vertices, and  $q = m + 1$  secondary items  $\{h_0, \dots, h_m\}$  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  $v \rightarrow w$  and each  $x \neq y$  with  $0 \leq x, y \leq m$ .

(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  $(l(000), \dots, l(111)) = (0, 1, 2, 5, 12, 6, 8, 3)$ ,  $(0, 1, 2, 6, 12, 8, 5, 3)$ ,  $(0, 1, 2, 9, 6, 4, 11, 8)$ ,  $(0, 1, 3, 10, 11, 6, 2, 12)$ .

**123.** (a) Let  $d = \gcd(l(w) - l(v), q)$  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  $cc' \equiv 1 \pmod{q'}$ .

permutation  
Möbius polynomial  
Krasner  
Ranulac  
Senderov  
Spivak  
Beluhov  
amazing identity  
Stappers  
uniquely



There are  $d$  solutions to the simultaneous equations  $(a \cdot l(v) + b) \bmod q = 0$  and  $(a \cdot l(w) + b) \bmod q = d$ , namely  $a = a_k$  and  $b = (-a_k \cdot l(v)) \bmod q$ , where  $a_k = c' + kq'$  and  $0 \leq k < d$ . 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  $p \nmid q$  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  $0 \leq k < d$  are  $d'$  copies of  $\{0, 1, \dots, r-1\}$ , because  $q' \perp r$ .

(b) Exactly  $d' \varphi(r) = d \prod_{p \mid d} (1 - \frac{1}{p})$  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  $u \rightarrow u'$  and  $(l(u') - l(u)) \bmod q = 1$ . Then  $(l'_k(u') - l'_k(u)) \bmod q = a_k$ .

(c) It suffices to find the essentially different cases that are *normalized*, in the sense that  $l(v) = 0$  and  $l(w) \nmid q$ . Begin with the set of all normalized solutions (a), grouping the solutions for divisor  $d$  into equivalence classes of size  $d \prod_{p \mid d} (1 - \frac{1}{p})$  as in (b). Then, for each automorphism or antiautomorphism  $\alpha$ , apply  $\alpha$  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  $\alpha$ 's to generate them all under composition.)

**124.** (a) Denote a labeling by the tuple  $l(a)l(b)l(c)l(d)$ . If we choose  $v = b$  and  $w = c$ , 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  $\alpha = (a d)$ . 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  $\alpha = (b c)$ . 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  $\alpha = (a d)(b c)$ , which is generated by the others.)

(Alternatively, we could have chosen  $v = a$  and  $w = b$ , 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  $v = a$  and  $w = d$ , 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  $\text{FIRST}[l] \leftarrow -1$  for  $0 \leq l < q$ . Then do the following steps for  $l = 1, 2, \dots, m$ : Set  $v \leftarrow \text{LO}[l]$ ,  $w \leftarrow (v + l) \bmod q$ ,  $t \leftarrow \text{FIRST}[v]$ ,  $\text{FIRST}[v] \leftarrow w$ ,  $\text{NEXT}[l] \leftarrow t$ .

(A similar algorithm will create arrays  $\text{FIRSTP}$  and  $\text{NEXTP}$  with which all *predecessors* of any given vertex can be visited efficiently. We can also readily create  $\text{FIRST}$ ,  $\text{NEXTL}$ , and  $\text{NEXTH}$  from the  $\text{LO}$  array of a graceful *undirected* graph.)

**126.** (a) Let  $f(-1) = -1$ , otherwise  $f(x) = (a(x - b)) \bmod q$ . Then  $\text{LO}'[l \bmod q] = f(\text{LO}[l])$ ;  $\text{FIRST}'[(a(l - b)) \bmod q] = f(\text{FIRST}[l])$ ;  $\text{NEXT}'[(a(l - b)) \bmod q] = f(\text{NEXT}[l])$ ;  $\text{NAME}'[(a(l - b)) \bmod q] = \text{NAME}[l]$ .

(b)  $\text{LO}$ ,  $\text{FIRST}$ , and  $\text{NEXT}$  are unchanged;  $\text{NAME}'[l] \alpha = \text{NAME}[l]$ .

(c)  $\text{LO}'[q - l] = (\text{LO}[l] + l) \bmod q$ ;  $\text{NAME}'[l] \alpha = \text{NAME}[l]$ ;  $\text{FIRST}'$  and  $\text{NEXT}'$  must be computed from  $\text{LO}'$  using exercise 125.

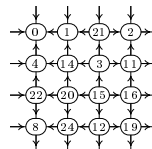
totient function  
automorphisms  
self-converse  
antiautomorphisms  
essentially distinct  
**NEXTL**  
graph representation

**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:  $\{012acj5g, 013cif28, 016e745g\}$ ,  $\{01649ehg, 018aid54, 0196edg4\}$ ,  $\{0165icf8, 01bec9ag, 01becajg\}$ ,  $\{0169ecjg, 01bac568, 01bac6f8\}$ ,  $\{016acfb8, 016j9ceg, 0198e6b4\}$ ,  $\{01358ife, 014eb976, 014hbje6, 017fg56i\}$ ,  $\{0135i8fe, 01657fgi, 01b9e476, 01bje46\}$ . (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  $4 \times 4$  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  $K_1 \rightarrow \overline{K_3}$ ,  $\overline{K_3} \rightarrow K_1$ .)

(b) Use arithmetic mod  $q$ . If  $a \perp q$  and  $aa' = 1$ , define  $ax = y_1 \dots y_m$  and  $-ax^T = z_1 \dots z_m$ , where  $y_i = a(x_{a'i} - x_{a'})$  and  $z_i = 1 - l - y_i$ . Reject  $x$  if  $x > ax$  or  $x > -ax^T$  lexicographically, for some  $a \perp q$ . The accepted tuples are inequivalent.

(c) The answer is  $\sum_{a=1}^m [a \perp q] \sum_{b=0}^m (f(a, b, q) + g(a, b, q)) / (2q\varphi(q))$  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  $t(\alpha, \beta, q) = \gcd(\alpha, q)[\gcd(\alpha, q) \setminus \beta]$ ; this is the number of  $x \in [0 \dots q)$  such that  $\alpha x \equiv \beta$  (modulo  $q$ ), when  $\alpha, \beta \in [0 \dots q)$ .

Let  $f(l, a, b, q) = (a^s l < l? 1: t(a^s - 1, -b(a^{s-1} + \dots + a + 1), q))$ , where  $s > 0$  is minimum with  $a^s l \leq l$ . (All arithmetic is mod  $q$ .) Then  $f(a, b, q) = \prod_{i=1}^{q-1} f(l, a, b, q)$ .

Let  $g(l, a, b, q) = ((-a)^s l < l? 1: t(a^s - 1, -b(a^{s-1} + \dots + a + 1) - l(s \bmod 2), q))$ , where  $s > 0$  is minimum with  $(-a)^s l \leq l$ . 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,  $\liminf_{n \rightarrow \infty} (\ln \ln n) \varphi(n) / n = e^{-\gamma}$ ; 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  $2, 4, \dots, n - 2$  become arc labels  $n - 2, n - 4, \dots, 2$ .

(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  $\{w_1, \dots, w_n\}$ . [*SIAM Journal on Algebraic and Discrete Methods* **6** (1985), 519–536.]

**132.**  $D = P_3$ ;  $l(v_0) = 2n + 1$ ,  $l(v_1) = 0$ ,  $l(v_2) = n + 1$ , and  $l(w_k) = k$  for  $1 \leq k \leq n$ .

**133.** Yes, because  $K_{m,n}$  is  $\alpha$ -graceful with labels  $\{0, 1, \dots, m - 1\}$  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  $m \rightarrow \infty$ . Many results have been reported by F. Stappers at [archive.org/details/graceful\\_digraphs\\_6](http://archive.org/details/graceful_digraphs_6).)

digits, extended hexadecimal  
Karnaugh map  
magical  $4 \times 4$  torus  
self-converse  
Burnside’s lemma  
congruence enumeration  
OEIS  
totient function  
 $\gamma$   
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  $1 \leq k \leq m$ , 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  $K_5^-$ .)

(b) C, G, H, I, J, K are not graceful; the other six are uniquely graceful. (The lexicographically smallest L0[1]L0[2]...L0[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 L0 tables 002102 and 001042 for  $A'$  and  $B'$ ;  $C'^T = B$ .

When  $n = 3$ ,  $A''$  is uniquely graceful but  $B''$  is the ungraceful  $C_3^-$ .

(d) Let  $v = q = \binom{n}{2} + 1$ . Given a graceful tournament on vertices  $\{1, \dots, n\}$ , with labels  $a_j = l(j)$ , suppose arc  $j \rightarrow k$  is labeled  $l$  and arc  $k' \rightarrow j'$  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  $(j \rightarrow k \text{ and } k' \rightarrow j')$  or  $(k \rightarrow j \text{ and } j' \rightarrow k')$  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  $u \rightarrow v$  whenever  $v$  is one of the next four elements after  $u$ . (But the transitive tournament  $K_9^-$  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  $n \leq 10^{10}$ ; 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  $u \rightarrow v$  going from  $\min(u, v)$  to  $\max(u, v)$  *except* that  $2 \rightarrow 1$ ,  $4 \rightarrow 2$ ,  $7 \rightarrow 4$ ,  $19 \rightarrow 15$ ,  $15 \rightarrow 0$ ; the “tolerant” arcs  $15 \rightarrow 7$  and  $15 \rightarrow 1$  also pair up with their reversals  $7 \rightarrow 15$  and  $1 \rightarrow 15$ .)

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  $\pm 1$ , and  $k = n$ , when they are  $\pm(n - (-1)^n)$ . Altogether, they are therefore  $\Delta(n) = \{\pm 1, \pm 3, \dots, \pm(2n-1)\}$ , 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  $\{\pm 1, \pm 2, \dots, \pm n\}$  (modulo  $q$ ). [*Discrete Mathematics* **261** (2003), 116.]


**140.** Regard  $T$  as rooted at  $v$ , with subtrees  $T_1, \dots, T_d$  where  $|T_1| \geq \dots \geq |T_d|$ , and number the vertices  $v_0, v_1, \dots, v_m$  in preorder. Let  $l(v_0) = 0$ ; and for  $k = 1, \dots, \lceil m/3 \rceil$  let  $l(v_k)$  be the least positive integer such that  $l(v_k) \neq l(v_j)$  and  $|l(v_k) - l(\text{parent}(v_k))| \neq$

historical notes  
transitive tournament  
self-converse  
Kumudakshi  
Carmichael  
Gordon  
tolerance  
Stappers  
lollipop  $L_{m,n}$   
preorder

$|l(v_j) - l(\text{parent}(v_j))|$  for  $1 \leq j < k$ . At most  $3(k-1)$  values are excluded, hence  $l(v_k) \leq m$ . Let  $C = \{|l(v_k) - l(\text{parent}(v_k))| \mid 1 \leq k \leq \lceil m/3 \rceil\}$  be the “colors” used.

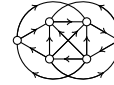
The remaining  $m - \lceil m/3 \rceil \leq 2m/3$  vertices are leaves adjacent to  $v$ , by hypothesis. So we can label them with the *negatives* of the unused colors,  $-(\{1, \dots, m\} \setminus C)$ .

**141.** The labels  $\{0, 1, 2, 5, 12, 23, 29\}$  give all differences  $\{\pm 1, \pm 2, \dots, \pm 18\}$  (modulo 37), with  $\pm 1, \pm 10, \pm 11$  occurring twice. For (i), let  $0 \not\sim 1, 2 \not\sim 29, 12 \not\sim 23$ ; for (ii), let  $1 \not\sim 2 \not\sim 12 \not\sim 1$ . For (iii), work modulo 41 and let  $0 \not\sim 36 \not\sim 18 \not\sim 31 \not\sim 0, 1 \not\sim 2 \not\sim 22 \not\sim 28 \not\sim 1$ . [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)  $a_1 + 2a_3 \equiv 0 \pmod{3} \iff a_1 \equiv a_3$ . 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  $\pi^p$  and  $\pi^{p^2}$ . [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  $\{1, \pi^v, \pi^{2v}, \dots, \pi^{(p-2)v}\} = \{1, 2, \dots, p-1\}$ , so the triples for  $\{1, \pi, \dots, \pi^{v-1}\}$  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  $\pi$  be a root of  $f$  in  $F$ . Then the subfield  $F_0$  of  $p^e$  elements is  $\{0, 1, \pi^v, \dots,$

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  $\theta = 0, 1 = 1, 2 = \omega, \dots, 7 = \omega^2 + \omega + 1$ , the points  $1, \pi, \pi^2, \dots, \pi^{72}$  are  $001, 010, 100, 512, 777, 603, 451, 655, 131, 602, 441, 755, 423, 175, 242, 304, 276, 044, \dots, 151$ , 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  $n \leq 2 \cdot 10^{10}$ ; 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  $ax = y_1 \dots y_m$  is defined by first setting  $z_{al} \leftarrow ax_l$  if  $al \leq m$ , 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  $f(l, a, b, q) = (a^s l = l? t(a^s - 1, -b(a^{s-1} + \dots + a + 1), q): q - a^s l = l? t(a^s - 1, l - b(a^{s-1} + \dots + a + 1), q): 1)$ , where  $s > 0$  is minimum with  $a^s l \leq l$  or  $q - a^s l \leq l$ . (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  $d(\text{cords}, \text{costs}) = 3$  and  $d(\text{colts}, \text{carts}) = 3$  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)  $\binom{n_1}{2} + \binom{n_2}{2} + \dots + \binom{n_r}{2}$ .

(b)  $n_1! n_2! \dots n_r! t_2! t_3! t_4! \dots$ , 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  $x_1 \dots x_r \sim y_1 \dots y_r$  because  $x_j \neq y_j$ , and  $x_1 \dots x_r \sim z_1 \dots z_r$  because  $x_k \neq z_k$ . Then  $y_1 \dots y_r \sim z_1 \dots z_r$  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  $k_0, k_1, \dots, k_4$  as we go around the cycle. Then  $k_0 \neq k_1 \neq \dots \neq k_4 \neq k_0$ , by (d). And each  $k_i$  must equal some  $k_j$  for  $j \neq i$ .

**152.** Every induced  $C_7$  of a Hamming graph is equivalent to  $000 \sim 100 \sim 110 \sim 111 \sim 121 \sim 021 \sim 001 \sim 000$ . 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.)

**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  $0 \leq j < i$  and  $j \neq i'$ , the Hamming distance  $d_H$  satisfies  $[v_j \not\sim v_i] < d_H(f(v_i), f(v_j)) \leq d(v_i, v_j)$ .

**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  $4 \leq n \leq 9$ , 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  $(|\mathcal{H}_4|/|\mathcal{C}_4|, \dots, |\mathcal{H}_9|/|\mathcal{C}_9|)$  turn out to be  $(5/6, 11/21, 36/112, 117/853, 469/11117, 2023/261080)$ ; and  $(|\mathcal{M}_4|, \dots, |\mathcal{M}_9|) = (1, 2, 0, 1, 1, 6)$ .

The MNH graphs for  $n \leq 8$  all turn out to be “tied-path graphs,” namely the graphs  $P(n_1, \dots, n_k)$  with  $2+n_1+\dots+n_k$  vertices and  $k+n_1+\dots+n_k$  edges that are obtained by tying together the endpoints of paths  $P_{n_1+2}, \dots, P_{n_k+2}$ :  $\mathcal{M}_4 = \{P(0, 1, 1)\}$ ;  $\mathcal{M}_5 = \{P(1, 2), P(1, 1, 1)\}$ ;  $\mathcal{M}_6 = \emptyset$ ;  $\mathcal{M}_7 = \{P(1, 1, 3)\}$ ;  $\mathcal{M}_8 = \{P(2, 2, 2)\}$ .

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

$$\mathcal{M}_9 = \left\{ \begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \\ \text{Diagram 3} \\ \text{Diagram 4} \\ \text{Diagram 5} \\ \text{Diagram 6} \end{array} \right\};$$

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  $l(v_0), \dots, l(v_{i-1})$ . Suppose we could legitimately set

unsupported  
backtrack  
strict embedding  
tied-path graphs

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  $\Delta$ ; 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  $d(l(v_i), l(v_j))$  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,  $(0 \text{---} 1) \bowtie (1 \text{---} 2) \bowtie (2 \text{---} 3) \nbowtie (0 \text{---} 1)$ .

(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  $k$ th 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  $l(w) = w_1 \dots w_r$  to vertex  $w$ , where  $w_k = (d(w, u^{(k)}) - d(w, v^{(k)})) \bmod 3$ .

**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)  $\nu((b \oplus b') \& \sim(a \mid a'))$ , the number of non- $*$  bits that differ.

(b) There are essentially only two other possibilities:

$$\begin{aligned} l(0) &= 0000, \quad l(1) = 1000, \quad l(2) = 110*, \quad l(3) = **11, \quad l(4) = 0001; \\ l(0) &= 000*, \quad l(1) = 100*, \quad l(2) = 1*10, \quad l(3) = *111, \quad l(4) = 010*. \end{aligned}$$

(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  $\nu(l(u) \oplus l(v)) = d(u, v)$  for all  $u$  and  $v$ . If  $w$  is their nearest common ancestor, coordinates  $(u, u', \dots, u^{(d(u, w)-1)}, v^{(d(w, v)-1)}, \dots, v', v)$  of  $l(u)$  and  $l(v)$  are respectively  $(1, 1, \dots, 1, 0, \dots, 0, 0)$  and  $(0, 0, \dots, 0, 1, \dots, 1, 1)$ ; 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

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  $0 \leq i < j < 5$  and root  $01$ , we have

23 04 14 24 03 13 34 02 12		23 04 14 24 03 13 34 02 12
$l(01) = 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$	$\Rightarrow$	$l(01) = 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$
$l(23) = 1 \ 0 \ 0 \ 0 \ ? \ ? \ 0 \ ? \ ?$		$l(23) = 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$
$l(04) = 1 \ 1 \ 0 \ ? \ ? \ * \ ? \ ? \ *$		$l(04) = 1 \ 1 \ 0 \ 0 \ * \ * \ 0 \ * \ *$
$l(14) = 1 \ 0 \ 1 \ ? \ * \ ? \ ? \ * \ ?$		$l(14) = 1 \ 0 \ 1 \ 0 \ * \ * \ 0 \ * \ *$
$l(24) = 0 \ ? \ ? \ 1 \ 0 \ 0 \ 0 \ ? \ ?$		$l(24) = 0 \ * \ * \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ .$
$l(03) = ? \ ? \ * \ 1 \ 1 \ 0 \ ? \ ? \ *$		$l(03) = * \ 0 \ * \ 1 \ 1 \ 0 \ 0 \ * \ *$
$l(13) = ? \ * \ ? \ 1 \ 0 \ 1 \ ? \ * \ ?$		$l(13) = * \ * \ 0 \ 1 \ 0 \ 1 \ 0 \ * \ *$
$l(34) = 0 \ ? \ ? \ 0 \ ? \ ? \ 1 \ 0 \ 0$		$l(34) = 0 \ * \ * \ 0 \ * \ * \ 1 \ 0 \ 0$
$l(02) = ? \ ? \ * \ ? \ ? \ * \ 1 \ 1 \ 0$		$l(02) = * \ 0 \ * \ * \ 0 \ * \ 1 \ 1 \ 0$
$l(12) = ? \ * \ ? \ ? \ * \ ? \ 1 \ 0 \ 1$		$l(12) = * \ * \ 0 \ * \ * \ 0 \ 1 \ 0 \ 1$

squashed cube  
isometric embedding  
Graham  
Pollak  
Winkler  
3-PARTITION  
strongly NP-complete  
Garey  
Johnson

(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  $0 \leq k < p$ ; similarly,  $u < v^{(k)}$  for  $0 \leq k < q$ . Define  $x_k = f(u^{(k)}, v) \bmod 2$  for  $0 \leq k \leq p$ , and  $x_{p+q-k} = f(u, v^{(k)}) \bmod 2$  for  $0 \leq k \leq q$ . 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  $x_k \neq x_{k+1}$ , for  $0 \leq k < p$ ; similarly  $u_{v^{(k)}} = ?$  and  $v_{v^{(k)}} = 1$  if and only if  $x_{p+q-k} \neq x_{p+q-k-1}$ , for  $0 \leq k < q$ . 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  $G \subseteq G'$  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  $H$ ,  $u \not\sim v$ ,  $f(u) \sim f(v)$ , and  $u = u_0 \sim u_1 \sim \dots \sim u_k = v$ . Then  $k > 1$ , and  $f(u_0) \sim f(u_1) \sim \dots \sim f(u_k) \sim f(u)$  is a cycle.

**163.** The vertex of degree  $m$  must map to  $r$ ; its neighbors must map to  $\{x_{10}, \dots, x_{m0}\}$ . So each path  $P_{a_i}$  must be mapped to  $a_i$  vertices of  $\{x_{j1}, \dots, x_{jn}\}$  for some  $j$ . Those with the same  $j$  form a submultiset of sum  $\leq n$ . So we get a suitable partition.

Conversely, such a partition yields an embedding. (And if  $a_1 + \dots + a_t = mn$  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  $(v, k) \mapsto (f(v, k), g(v, k))$  is an embedding. For fixed  $v$ , let  $w_k = f(v, k)$  and  $r_k = g(v, k)$ . If  $w_k \neq w_{k+1}$  we must have  $r_k = 0$  or  $r_{k+1} = 0$ . And if, say,  $r_0$  is the only 0, we have  $w_0 \neq w_1 = \dots = w_4$  and  $\{r_1, r_4\} = \{1, 3\}$ , implying both  $w_0 \sim w_1$  and  $w_0 \not\sim w_1$ . A similar contradiction arises if  $r_k$  is the only 0. So



$(r_0, \dots, r_4)$  must be a cyclic permutation of  $(0, 1, 0, 3, 4)$  or  $(0, 1, 0, 4, 3)$ ; but none of those is compatible with  $(w_2, r_2) \text{---} (w_4, r_4)$ . 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  $(v, v) \text{---} (v, w) \text{---} (w, w)$  for all  $v \neq w$ , and  $(v, w) \text{---} (w, v)$  when  $v \text{---} w$ . Let  $t(G)$  be  $s(G)$  together with additional vertices  $\{v, w\}$  whenever  $v \text{---} w$ ; we have  $(v, v) \text{---} \{v, w\} \text{---} (w, w)$  when  $\{v, w\}$  exists. One can now prove that  $G \subseteq H$  if and only if  $s(G) \subseteq t(H)$ .

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  $\leq 3$ . Then  $f(v, w)$  and  $f(w, v)$  when  $v \text{---} w$  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  $v \not\text{---} w$ ,  $\{f(v, w), f(w, v)\}$  can be any two of  $(f(v), f(w))$ ,  $(f(w), f(v))$ , and possibly  $\{f(v), f(w)\}$ .

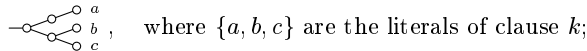
(Christine Solnon noticed that  $s(G)$  has a huge number of automorphisms, because one can independently swap  $(v, w)$  with  $(w, v)$  when  $v \neq w$ . To avoid this problem she uses *directed arcs*  $(v, v) \rightarrow (v, w) \rightarrow (w, w)$ .)

**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  $u \not\text{---} v$ , where  $\Lambda$  is always compatible with  $\Lambda$ . Finally — and this is the key point — introduce a new edge label  $L_J$ , where  $L_J(v, w) = [v \text{---} w]$ , 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  $l_I(v) = [v \text{---} u]$  for  $v \in G \setminus u$  and  $l_I(\hat{v}) = [\hat{v} \text{---} \hat{u}]$  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  $u \text{---} v$ , otherwise  $l_{I+1+j}(v) = \Lambda$ , using the compatibility relation of  $L_j$  and letting  $\Lambda$  be self-compatible.

(For directed graphs, however, we need more. Arc labels  $L_j(v, w)$  are given when  $v \rightarrow w$ . In part (a) let  $L_J(v, w) = 2[v \rightarrow w] + [v \leftarrow w]$ . In part (b), let  $l_I(v) = [v \rightarrow u]$ ,  $l_{I+1}(v) = [v \leftarrow u]$ ,  $l_{I+2+j}(v) = L_j(v, u)$  or  $\Lambda$ ,  $l_{I+2+J+j}(v) = L_j(u, v)$  or  $\Lambda$ .)

**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  $\circ \text{---} \circ \text{---} \dots \text{---} \circ \text{---} \circ$ , 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  $\text{---}$  by a path  $\text{---} \circ \text{---} \circ \text{---} \circ$ . Similarly,  $H$  is obtained from  $B_m^+$  by replacing the  $k$ th leaf  $\text{---}$  by the graph



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  $G \subseteq H$ , because we can match the ‘tip’ of leaf  $k$  to a literal  $a, b$ , or  $c$  that satisfies clause  $k$ . Conversely, if  $G \subseteq H$ , 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  $v \text{ --- } w$  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  $\{e \cdot E \mid e = (u \text{ --- } v) \text{ and } E = (U \text{ --- } V) \text{ for some } u \text{ and } U\}$ .

**171.** For SIP, there's a secondary item  $uv \cdot UV$  for every arc  $u \rightarrow v$  in the pattern and every nonarc  $U \not\rightarrow V$  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  $u \not\rightarrow v$  in the pattern and every arc  $U \rightarrow V$  in the target.

**172.** (a) If there are  $W_{mn}$  strict embeddings from  $\mathcal{G}_m$  to  $\mathcal{G}_n$ , then  $E(W_{mn}) = n^m/2^{\binom{m}{2}}$ , because each of the  $n^m$  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)),  $\Pr(\mathcal{G}_m \sqsubseteq \mathcal{G}_n) = \Pr(W_{mn} > 0) \leq E(W_{mn}) \leq n^{-\delta}$ .

(b) Clearly  $E(W_{mn}) \geq n^m 2^{-\binom{m}{2}} (1 - m^2/n)$ ; and when  $m = 2 \lg n + 1 - \delta$ , one can show that  $E(W_{mn}^2) \leq (n^m/2^{\binom{m}{2}})^2 (1 + O(n^{-2\delta/3}))$ . 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 T $\mu$  (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  $3 + 1.5 \text{ G}\mu$  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  $v_1, \dots, v_p$ , where  $d(v_1) \geq \dots \geq d(v_p)$ ; similarly, let  $f(v)$ 's neighbors in  $H$  be  $w_1, \dots, w_q$ , where  $d(w_1) \geq \dots \geq d(w_q)$  and  $q \geq p$ . Given  $k \leq p$ , there are indices  $1 \leq i_1 < \dots < i_k \leq q$ , depending on  $k$ , such that  $\{f(v_1), \dots, f(v_k)\} = \{w_{i_1}, \dots, w_{i_k}\}$ . 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. Solmon, *Constraints* **15** (2010), 327–353.]

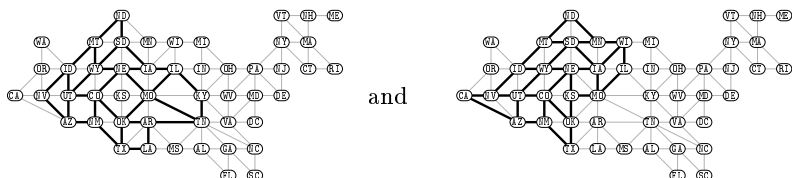
(b) (Solution by C. Solmon.) For  $1 \leq k \leq p$ , let  $v$  have  $a_k$  neighbors of degree  $k$ ; also let  $w$  have  $b_k$  neighbors of degree  $k$ , or of degree  $\geq k$  when  $k = p$ . Then check whether or not  $b_p \dots b_1$  majorizes  $a_p \dots a_1$ , namely whether or not  $b_p + \dots + b_k \geq a_p + \dots + a_k$  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  
Solmon  
majorizes

[This partial ordering of multisets is a distributive lattice. When restricted to multisets of at most  $r$  positive integers, all  $\leq s$ , it's the lattice  $L(r, s)$  of partitions into at most  $r$  parts  $\leq s$ , 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  $\mathbf{h}$ , so we can't map  $00 \mapsto AL$ .)

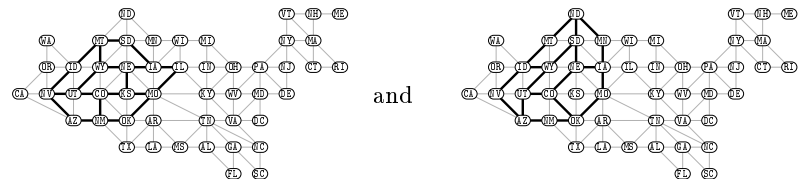
180.



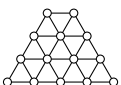
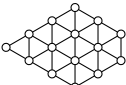
181. (a) One of  $4 \cdot 12$  embeddings for  $P_2 \square P_{12}$  is  $\begin{pmatrix} CA & OR & ID & WY & NE & IA & WI & MI & OH & WV & PA & NJ \\ AZ & NV & UT & CO & KS & MO & IL & IN & KY & VA & MD & DE \end{pmatrix}$ .  
 (b) And  $\begin{pmatrix} OR & ID & WY & NE & IA & IL \\ NV & UT & CO & KS & MO & KY \\ CA & AZ & NM & OK & AR & TN \end{pmatrix}$  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 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  $j \notin \{j_1, \dots, j_m\}$ . 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  $u_k \neq \perp$ , let  $v_k = m(u_k)$  and  $u_{k+1} = M(v_k)$ . If  $u_k = u_0$ , this sequence will be periodic, and  $u_k \rightarrow v_{k-1} \rightarrow u_{k-1} \rightarrow \dots \rightarrow v_0 \rightarrow u_0$  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  $u_{-l} \neq \perp$ , 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  $u_k \rightarrow v_{k-1} \rightarrow u_{k-1} \rightarrow \dots \rightarrow v_{-l} \rightarrow u_{-l}$ ; so  $y_j$  and  $\perp$  will be in the same strong component.

Conversely, if there's an oriented path  $x_i \rightarrow \dots \rightarrow y_j \rightarrow x_i$  or  $\perp \rightarrow \dots \rightarrow y_j \rightarrow x_i$  in  $T$ , we can convert the given matching to a feasible matching with  $x_i \sim y_j$  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  $x_1 \sim y_{j_1}, \dots, x_m \sim y_{j_m}$  is such a matching, the sequence  $j_1 \dots j_m$  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  $I \subseteq \{1, \dots, m\} \setminus i$  with  $|D'(I)| < |I|$ , where  $D'$  is the subdomain in the subgraph. Thus  $|D(I)| \leq |I|$ ; 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  $\{1, \dots, n\}$ , 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  $a' \geq a$  and  $a' + b' + c' \geq |D(A \cup B \cup C)| \geq a + b + c$ . Therefore  $2a' + b' + c' \geq 2a + b + c = 2a' + b' + c'$ , 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. König [*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  $\{x_1, \dots, x_n\}$ , and let  $G$  be the bigraph on  $\{x_1, \dots, x_n\}$ ,  $\{y_1, \dots, y_n\}$  whose edges are  $x_i \text{ --- } y_i$  for  $1 \leq i \leq n$  and  $x_i \text{ --- } y_j$  whenever  $x_i \rightarrow y_j$  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  $\{y_1, \dots, y_n\}$  with arcs  $\{y_{j_i} \rightarrow y_k \mid x_i \rightarrow y_k \text{ and } k \neq j_i\}$ . Then  $k \neq j_i$  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  $D_a$  of **a** must be a target vertex with a predecessor of out-degree  $\geq 4$ ; so  $D_a = \{6, 8, 13, 14, 15, 16\}$ . And  $D_d$  is the set of targets with a predecessor having out-degree  $\geq 1$ , in-degree  $\geq 1$ , and at least two neighbors, namely  $\{3, 6, 8, 12, 13, 14, 15, 16\}$ . But  $D_b = \{3\}$  is the only target with a predecessor of bi-degree  $\geq 1$ , where “bi-degree” is the number of two-way ( $\leftrightarrow$ ) edges;  $D_c = \{8\}$  is the only target with bi-degree  $\geq 1$  and out-degree  $\geq 2$ . Similarly,  $D_e = D_a$ ;  $D_f = \{0\}$ ;  $D_g = \{14\}$ .

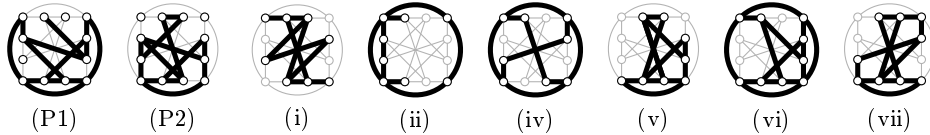
After the forced assignments **b**  $\mapsto$  3, **c**  $\mapsto$  8, **f**  $\mapsto$  0, **g**  $\mapsto$  14, the remaining domains reduce to  $D_a = D_e = \{6, 13, 15, 16\}$ ;  $D_d = D_a \cup \{12\}$ . LAD filtering now tells us that **e**  $\not\mapsto$  16, because  $D_d$  doesn't contain 16's successor (3). Similarly, **d**  $\not\mapsto$  13; **d**  $\not\mapsto$  16.

Marshall Hall  
pigeonhole principle  
perfect matchings  
Historical notes  
Berge  
perfect matching  
elementary bipartite  
Lovász  
Plummer  
König  
bigraph  
strongly connected  
Régin  
Gent  
Miguel  
Nightingale  
bi-degree

So we branch on  $e$ , and there are three cases: If  $e \mapsto 6$ , then  $d \mapsto 15$  and we discover two solutions,  $a \mapsto 13$  or  $16$ . If  $e \mapsto 13$ , then  $d \mapsto 12$  and  $a \mapsto 6, 15$ , or  $16$ . If  $e \mapsto 15$ , then  $d \mapsto 6$  and  $a \mapsto 13$  or  $16$ .

(b) With strict embedding the initial domain  $D_e$  is reduced to  $\{8, 13, 16\}$ . Only two of the previous solutions survive:  $(a, b, c, d, e, f, g) \mapsto (6 \text{ or } 15, 3, 8, 12, 13, 0, 14)$ .

**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.



**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  $f$ .)

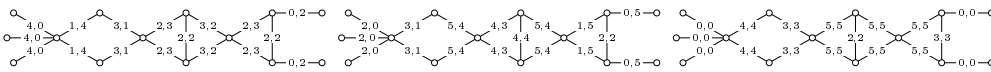
**191.** Spectacularly false. For example, if  $H = G - K_1$  then  $H^{\leq 2}$  is a complete graph.

**192.** The degree of 12 in  $G^{\leq 2}$  is 11. So we can exclude 22 vertices whose degree in  $H^{\leq 2}$  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  $\ell_G, \ell_H$  did exclude all of these except MN and NV, and picked off also NM and WI.)

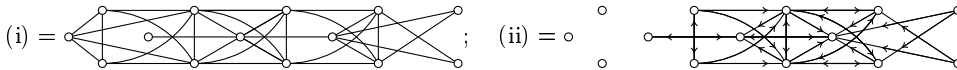
**193.**  $d_G^{P_{k+1}}(v)$  is the number of simple paths of length exactly  $k$  that begin at  $v$ . (Thus when  $k = 1$ ,  $d_G^{P_2}(v) = \deg(v)$ .) Consequently  $v$ 's degree in  $G^{\leq 2}$  is  $d_G^{P_2}(v) + [d_G^{P_3}(v) > 0]$ .

**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.



**197.** Indeed, if  $v \mapsto M0$  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  $M0$ , because no appropriate 4-cycle connects them to  $M0$ .

**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  $11 \mapsto NV$ , then  $12 \mapsto AZ, CA, ID, OR$ , or  $UT$ ; hence  $12 \mapsto UT$ . Similarly  $11 \mapsto NV$  implies  $21 \mapsto UT$ , a contradiction. An analogous contradiction rules out  $11 \mapsto WV$ .

**199.** One part has the neighbors  $u'$  of  $u$  in  $G$  (either  $u \rightarrow u'$  or  $u \leftarrow u'$  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  $u \rightarrow u'$  in  $G$  then  $v \rightarrow v'$  in  $H$ . (iii) If  $u \leftarrow u'$  in  $G$  then  $v \leftarrow v'$  in  $H$ . (iv) For each

line graph  
automorphisms  
domain

supplemental pair label that we've computed, satisfying (64),  $\ell_G(u, u') \leq \ell_H(v, v')$  and  $\ell_G(u', u) \leq \ell_H(v', v)$ . And if  $G$  is to be *strictly* embedded into  $H$ , we also have two more conditions: (v) If  $u \not\rightarrow u'$  in  $G$  then  $v \not\rightarrow v'$  in  $H$ . (vi) If  $u \not\leftarrow u'$  in  $G$  then  $v \not\leftarrow v'$  in  $H$ .

Condition (i) implies that  $d_G(u') \leq d_H(v')$  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  $G^\Sigma$  and  $H^\Sigma$  that we know are solutions to (65).

**200.** Count the number of *strict* embeddings  $S \subseteq G$  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  $p^{(1)}p^{(2)} \dots$  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  $u_1 \dots u_n$  so that, if the target vertices are  $\{1, \dots, n\}$ , we have  $t_j = k$  if and only if  $u_k = j$ . Initially  $t_j = u_j = j$  for  $1 \leq j \leq n$ . 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  $u_k \leq s_l$ .)

(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  $n \leq 64$  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  $H_{v'}^S$ ; bit  $u'$  of  $H_{v'}^S$  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  $u \neq v$ , we must set  $D_u \leftarrow D_u \& H_{v'}^S$ , 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  $D_u \leftarrow D_u \& \sim H_{v'}^S$ .)

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  $H_{v'}^S$ . If the graphs are directed, byte  $A_{uv}$  should represent  $u \rightarrow v$  in the pattern graphs, and bit  $u'$  of  $H_{v'}^S$  should represent  $u' \rightarrow v'$  in  $H^S$ . The transposed target graphs should also be represented separately, so that bit  $u'$  of  $H_{v'}^{ST}$  represents  $v' \rightarrow u'$  in  $H^S$ . If  $A_{vu}$  tells us that  $v \rightarrow u$  in  $H^S$ , we should set  $D_u \leftarrow D_u \& H_{v'}^{ST}$ .

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  $\text{dom} + 8k$ . Sorting is achieved by making byte  $\text{START}[i]$  point to the first domain of size  $i$ ;  $\text{NEXT}[k]$  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  $\geq m$ , because they can be treated in any order. Symbols  $t, u, h, i, j, k, \text{kk}$  denote registers \$255, \$0, \$1, \$2, \$3, \$4, \$5.

Sort	SET	j,0	$j \leftarrow 0$ .	LDB	k,start,0	$k \leftarrow \text{START}[0]$ .	
	SET	i,56	$i \leftarrow 56$ .	PBZ	k,2F	No domain empty?	
1H	STOU	j,start,i	$\text{START}[i \dots i + 7] \leftarrow 0$ .	1H	INCL	j,1	$j \leftarrow j + 1$ .
	SUB	i,i,8	$i \leftarrow i - 8$ .	8ADDU	kk,k,0		
	PBNN	i,1B	Repeat while $i \geq 0$ .	LDOU	t,kk,dom	$t \leftarrow D_k$ .	
	CMP	t,i,0	$t \leftarrow -1$ .	OR	u,u,t	$U \leftarrow U \cup t$ .	
	STB	t,m,start	$\text{START}[m] \leftarrow -1$ .	ANDN	t,t,h	$t \leftarrow t \setminus H$ .	
	SET	k,m	$k \leftarrow m$ .	BZ	t,Unfeas	To Unfeas if $t = \emptyset$ .	
1H	8ADDU	kk,k,0	$\text{kk} \leftarrow 8k$ .	STOU	t,kk,dom	$D_k \leftarrow t$ .	
	LDOU	t,kk,dom	$t \leftarrow D_k$ .	SADD	t,u,0	$t \leftarrow  U $ .	
	SADD	t,t,0	$t \leftarrow  D_k $ .	CMP	t,t,j	$t \leftarrow \text{sign}(t - j)$ .	
	CMP	i,t,m		BN	t,Unfeas	To Unfeas if $ U  < j$ .	
	CSP	t,i,m	If $t > m$ set $t \leftarrow m$ .	CSZ	h,t,u	If $ U  = j$ set $H \leftarrow U$ .	
	LDB	i,start,t		LDB	k,k,next	$k \leftarrow \text{NEXT}[k]$ .	
	STB	i,next,k	$\text{NEXT}[k] \leftarrow \text{START}[t]$ .	BP	k,1B	Repeat loop if $k > 0$ .	
	STB	k,start,t	$\text{START}[t] \leftarrow k$ .	BN	k,Feas	We're done if $k < 0$ .	
	SUB	k,k,1	$k \leftarrow k - 1$ .	2H	INCL	i,1	$i \leftarrow i + 1$ .
	PBP	k,1B	Loop while $k > 0$ .	LDB	k,i,start	$k \leftarrow \text{START}[i]$ .	
DoIt	SET	u,0	$u \leftarrow \emptyset$ .	PBP	k,1B	Repeat loop if $k > 0$ .	
	SET	h,0	$h \leftarrow \emptyset$ .	PBZ	k,2B	Increase size if $k = 0$ .	
	SET	i,0	$i \leftarrow 0$ . ( $j = 0$ )	Feas	...		

The total time is approximately  $(8\mu + 30v)m + 10\mu + 38v$ .

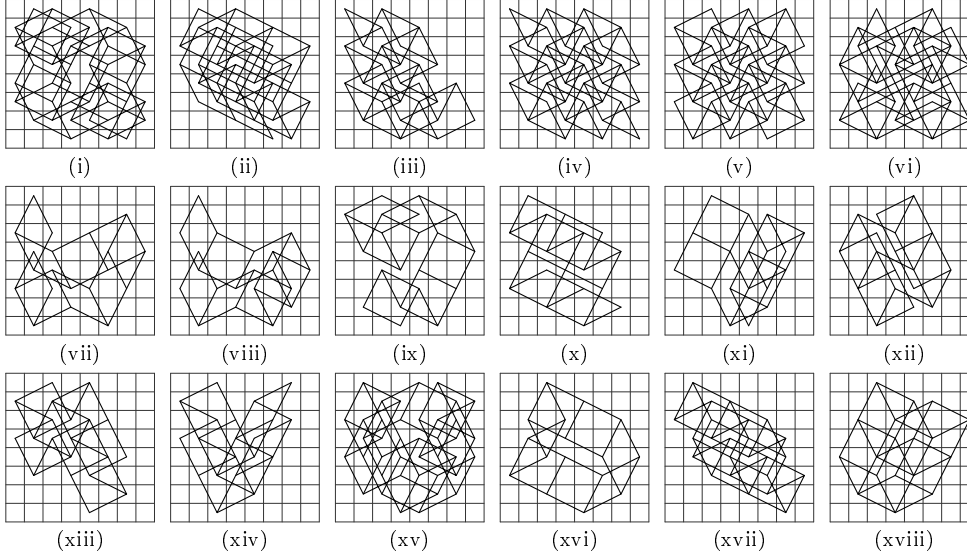
Complete GAD filtering can also be done with bitwise manipulation, but the algorithms are considerably more complicated and time-consuming. See P. Van Kessel and C.-G. Quimper, *Proceedings of the AAAI Conference* **26** (2012), 577–583.

**205.** (a) The problem is to find knight paths  $p_1 \dots p_m$  and  $q_1 \dots q_n$  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  $P_{n-2} \square P_{n-2} \subseteq N_n$  for all  $n \geq 4$ , 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  $P_2 \square P_2 \square P_2 \square P_2$ , which is also  $C_4 \square C_4$ , is *uniquely* embeddable in  $N_n$  for all  $n \geq 7$ , 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  $p_1 \dots p_m$ . 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  $q_1 q_2 \dots$  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  $(f_2(3), \dots, f_2(11)) = (1, 2, 7, 10, 15, 22, 29, 36, 46)$ ; and  $f_2(12) \geq 57$ , because of the knight path  $q_1 \dots q_{57}$  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:  $(f_3(3), \dots, f_3(14)) = (1, 1, 3, 5, 9, 12, 16, 20, 27, 33, 39, 48)$ ; and  $f_3(15) \geq 55$  because of a knight path  $q_1 \dots q_{55}$  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  $q_1 \dots q_{48}$ .

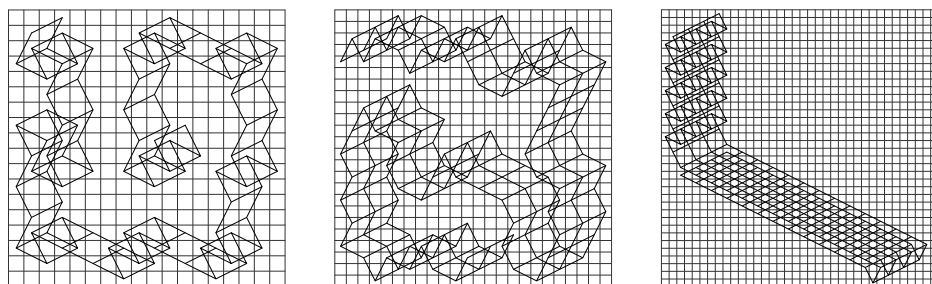
When  $m = 4$  we have  $(f_4(3), \dots, f_4(17)) = (1, 1, 1, 2, 4, 5, 7, 10, 15, 18, 22, 25, 34, 37, 43, 52)$ , and  $f_4(18) \geq 61$ . In this case the optimum solutions for  $13 \leq n \leq 16$  all occur when  $p_1 p_2 p_3 p_4$  is the zigzag path shown as ‘1 1 1 1’; such solutions prove that  $f_4(t) = t^2/4 - O(t)$ . However, the zigzag path yields only  $f_4(17) \geq 49$ . Hence the straight path wins when  $t = 17$ , and the sequence  $f_4(t)$  remains mysterious.

Turning now to induced subgraphs,  $(\bar{f}_2(3), \dots, \bar{f}_2(18)) = (1, 2, 5, 8, 8, 10, 12, 15, 19, 24, 28, 32, 36, 40, 46, 52)$ ; also  $(\bar{f}_3(3), \dots, \bar{f}_3(24)) = (1, 1, 3, 4, 5, 6, 7, 10, 11, 12, 14, 16, 20, 21, 25, 28, 32, 34, 41, 44, 49, 53)$ ; furthermore  $(\bar{f}_4(3), \dots, \bar{f}_4(36)) = (1, 1, 2, 4, 4, 5, 6, 8, 8, 10, 12, 12, 14, 15, 17, 18, 20, 20, 22, 24, 25, 26, 28, 29, 31, 32, 34, 35, 37, 38, 40, 41, 43, 44)$ . It appears that  $\lim_{t \rightarrow \infty} \bar{f}_2(t)/t^2 = \alpha_2$  and  $\lim_{t \rightarrow \infty} \bar{f}_3(t)/t^2 = \alpha_3$  for some (unknown) positive constants  $\alpha_2$  and  $\alpha_3$ . But  $\bar{f}_4(t) = O(t)$ , because none of the paths  $p_1 p_2 p_3 p_4$  allow us to “turn a corner.”



[illegible]

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



**Fig. A-15.** Champion knight's grids on larger boards.

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 06 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 27 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

December 5, 2024

Without reducing for symmetry, there are 44176 embeddings for  $n = 3$ , 171569126 for  $n = 4$ , and zillions for  $5 \leq n \leq 7$ . Restricting to solutions with central symmetry, these counts become (80, 66624, 69200, 1599680, 48560, 32000), for  $3 \leq n \leq 8$ .

Surprisingly, no 4-way symmetry is possible for  $9 \leq n \leq 30$ . 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  $9 \leq n \leq 12$ !]

**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  $n = (1, 2, 3, 4)$ .

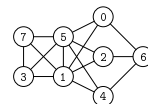
When  $n = 5$ , there are exactly 28800 embeddings. In fact, they are the mappings  $ij \mapsto p((2i+j) \bmod 5)q((3i+j) \bmod 5)$  and their transposes, when  $p$  and  $q$  are arbitrary permutations of  $\{0, 1, 2, 3, 4\}$ . Those maps also embed the toroidal  $5 \times 5$  knight moves.

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  $n$ .)

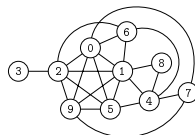
**210.** If solvable, there would be headline news: We could name 75 American collegiate football teams who played each other in 1990 if and only if 75 corresponding characters encountered each other in the first half of Victor Hugo's *Les Misérables* (1862)! But unfortunately this one is *not* solvable. 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  $1+1+1+1+1+1+1+1+2+2+2$  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  $S(\circ \! \! \! \curvearrowright) = \bigvee_{tuvw} x_{tu}x_{tv}x_{tw}x_{uv}\bar{x}_{uw}\bar{x}_{vw}$ , 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  $(\lambda(1), \dots, \lambda(6)) = (1, 3, 5, 8, 10, 14)$  and  $16 \leq \lambda(7) \leq 18$ . The minimum number of edges in an  $n$ -universal graph is  $(0, 1, 4, 11, 21)$  for  $1 \leq n \leq 5$ ; 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  $\lambda(7) > 16$ .]

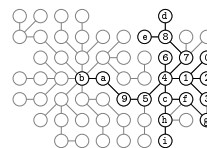
**212.** In fact we can obtain each  $V_{j+1}$  by “promoting” a vertex of  $V_j$ :  $V_1 = 0125 (K_{1,1,2})$ ;  $V_2 = 0126 (C_4)$ ;  $V_3 = 0136 (P_4)$ ;  $V_4 = 0236 (P_3 \oplus K_1)$ ;  $V_5 = 0237 (K_2 \oplus 2K_1)$ ;  $V_6 = 0247 (\overline{K_4})$ ;  $V_7 = 1247 (K_{1,3})$ ;  $V_8 = 1347 (L(3, 1))$ ;  $V_9 = 1357 (K_4)$ ;  $V_{10} = 1367 (C_3 \oplus K_1)$ ;  $V_{11} = 2367 (2K_2)$ . *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=KelZ0GPr3Zw).]

An interesting CSP now suggests itself: Given a digraph in which each vertex  $v$  has a given color  $c(v) \in \{1, \dots, d\}$ , we seek an oriented path  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_d$  such that each color occurs once in  $\{c(v_1), c(v_2), \dots, c(v_d)\}$ . 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  $1 \leq x \leq d$ , 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  $v_1, \dots, v_t$ , there are  $3t$  options ' $x v_1:\delta_{1s} \dots v_t:\delta_{ts}$ ', ' $\perp v_s:1 x-$ ', ' $x+ v_s:1 \top$ ', for  $1 \leq s \leq t$ . Also, for each arc  $v \rightarrow v'$  with  $c(v) \neq c(v')$ , 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  $\{0, \dots, 7\}$  and  $c(v)$  is the corresponding induced subgraph;  $v \rightarrow v'$  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  $\{0, 2, 3\}$ :

(b) Yes. Subtree  $S_r$  has nodes  $\{7, 8, d, e\}$ ; subtree  $T_e$  has nodes  $\{w, x, y, z, A, B\}$ ; map  $7 \mapsto w, 8 \mapsto x, d \mapsto y, e \mapsto z$ . (This example uses an extended hexadecimal code in which the letters  $[a..z]$  denote  $[10..35]$  and the letters  $[A..Z]$  denote  $[36..61]$ .)



(c) Let  $e_j = \frac{v}{w_j}$ , for  $1 \leq j \leq l$ . The stated embedding is possible if and only if there are analogous embeddings of  $S_{r_1}, \dots, S_{r_k}$  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  $b_i \text{ --- } g_j \iff \text{sol}[r_i][e_j]$ .

(e) Let  $v$ 's neighbors be  $\{w_0, \dots, w_l\}$ ; define  $e_j$  as in (c), but for  $0 \leq j \leq l$ . Now consider the graph of (d), but with  $l+1$  girls. The embedding for  $e = \frac{u}{v}$  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  $\iff g_j \in \{\text{QUEUE}[0], \dots, \text{QUEUE}[q-1]\}$  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  $e = \frac{u}{v}$  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  $e = \frac{u}{v}$ .) For  $1 \leq d \leq n$ , set  $\text{THRESH}[d]$  to the number of arcs with  $\deg(e) < d$ . If  $e = \frac{u}{v}$  and  $e' = \frac{u'}{v}$ , set  $\text{UERT}[e] \leftarrow u$ ,  $\text{VERT}[e] \leftarrow v$ ,  $\text{DUAL}[e] \leftarrow e'$ .

The heart of the computation is *solve*( $r$ ), a recursive procedure to set  $\text{sol}[q][e]$  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

Finally,  $S \subset T$  if and only if  $\text{sol}[1][e] = 1$  for at least one arc  $e$ .

[illegible]

**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  $T$ s); tree  $S = K_{1,12}$  is embeddable the least (31  $T$ s); tree  $T_1$  has the most embedded subtrees (74  $S$ s); 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).

$$S_1 = \text{graph 1} \subseteq \max; \quad T_1 = \text{graph 2} \supseteq \max; \quad S_2 = \text{graph 3} \not\subseteq T_2 = \text{graph 4}.$$

(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.

**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  $j \in D_v$ :

$$v \vee \bigvee_{(u,v,i,j) \in N} u_i v_j \vee \bigvee_{(v,u,i,j) \in N} v_j u_i.$$

For radio coloring we can in fact do better. Let  $D_v = [0 \dots d]$  for all  $v$ , and  $N = \{(u, v, i, j) \mid u \text{ --- } v, u < v, |i - j| < 2\}$ ; 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

$$'v \bigvee_{u \text{ --- } v} u_j \bigvee_{(u, v, i, j) \in N} u_i v_j \bigvee_{(v, u, j, i) \in N} v_j u_i'.$$

Prestwich  
Klavžar  
ternary-to-binary encodings  
unit clauses  
weakened encoding

**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.**  $v_2 v_1 = (00, 01, 10, 11)$  means  $v = (0, 1, 2, 1 \text{ or } 2)$ . The allowable  $u_2 u_1 v_2 v_1$  are 0001, 0010, 0011, 0100, 0110, 1000, 1001, 1100; hence  $u \neq v$ . (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  $01 \dots 1$  and  $02 \dots 2$ , there are two ways to 3-color each of the subgaskets  $1* \dots *$  and  $2* \dots *$ ; 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

$$\begin{array}{lll} 0\rho = \bar{1}; & 0\sigma = 1; & 0\tau = \bar{1}; \\ 1\rho = \bar{1}; & 1\sigma = \bar{1}; & 1\tau = 1; \\ 2\rho = 1; & 2\sigma = \bar{1}; & 2\tau = \bar{1}; \end{array} \quad \begin{array}{l} \left( ((a_1\rho) \dots (a_{n-1}\rho))_2, \right. \\ \text{and let } a_1 \dots a_{n-1} \mapsto ((1a_1\sigma) \dots (a_{n-1}\sigma))_2, \\ \left. ((a_1\tau) \dots (a_{n-1}\tau))_2 \right). \end{array}$$

For example,  $1202 \mapsto ((\bar{1}1\bar{1}1)_2, (1\bar{1}\bar{1}1\bar{1})_2, (1\bar{1}\bar{1}\bar{1})_2) = (-5, 5, 1)$ . 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  $\alpha$ , but only if  $\alpha$  is a ternary vector  $a_1 \dots a_{n-1}$ .

To go from the representation of triangle  $\alpha$  to its three vertices  $\alpha 0$ ,  $\alpha 1$ , or  $\alpha 2$ , add respectively  $(-1, -1, 1)$ ,  $(-1, 1, -1)$ , or  $(1, -1, -1)$ . The corner points are  $0 \dots 00 \mapsto (-2^n, 2^{n+1}, -2^n)$ ;  $1 \dots 11 \mapsto (-2^n, 0, 2^n)$ ; and  $2 \dots 22 \mapsto (2^n, 0, -2^n)$ .

**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:

	<i>without clique hints</i>									<i>with clique hints</i>							
	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



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:  $a_0 \text{ --- } b_1$ ,  $e_0 \text{ --- } d_1$ ,  $f_0 \text{ --- } c_1$ ,  $a_1 \text{ --- } b_1$ ,  $b_1 \text{ --- } c_1$ ,  $b_1 \text{ --- } d_1$ ,  $c_1 \text{ --- } d_1$ ,  $c_1 \text{ --- } e_1$ ,  $d_1 \text{ --- } f_1$ .)

**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  $a_1 \dots a_n$  with  $a_j < d$  for all  $j$ . (And we can independently remap the coordinates of those vertices in  $d' \text{ --- } d = d'(d' - 1) \dots (d' - d + 1)$  ways.)

**237.** The pure vertices  $j \dots j$  for  $0 \leq j < d$  ( $S_n^{(d)}$ ),  $2 \leq j < d$  ( $\widehat{S}_n^{(d)}$ ),  $1 \leq j < d$  ( $\overline{S}_n^{(d)}$ ).

**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  $j \dots j$ , 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  $(v_0 \vee \dots \vee v_{d-1})$  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  $v_1, v_2, v_4, \dots$ , denoting  $v = (\dots v_4 v_2 v_1)_2$ . Assert clauses of length  $l$  to exclude the cases  $d \leq v < 2^l$ . (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  $j$ th 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  $v^j = [v \geq j]$  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  $(u_1 \vee \dots \vee u_{d-1} \vee v_1 \vee \dots \vee v_{d-1})$  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  $(\bar{u}_j \vee v_1 \vee \dots \vee v_{j-1} \vee v_{j+1} \vee \dots \vee v_{d-1})$ , and the same with  $u \leftrightarrow v$ . Weakened: Assert  $(\bar{u}_j \vee u_{j+1} \vee \dots \vee u_{d-1} \vee \bar{v}_j \vee v_{j+1} \vee \dots \vee v_{d-1})$ . 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  $\{v^{(1)}, \dots, v^{(d)}\}$ . 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  $3 \leq c \leq d$ , we can assert the positive clause  $(v^{(i_1)} \vee \dots \vee v^{(i_c)})$  for all  $\binom{d}{c}$  subsets  $\{i_1, \dots, i_c\} \subseteq \{1, \dots, d\}$ . Similarly, if we know that  $v_k = 0$  for at least one such  $v$ , we can assert the negative clause  $(\bar{v}^{(i_1)} \vee \dots \vee \bar{v}^{(i_c)})$  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  $0 \leq j < d$ ; 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  $(u_j \vee v_j \vee w_j \vee x_j), \dots, (v_j \vee w_j \vee x_j \vee y_j)$ , 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 M $\mu$  to 0.8 M $\mu$  (median of nine runs) — better than K6!

**243.** Suppose we have a 4-coloring  $h$ , with  $h(a_1 \dots a_n) \in \{0, 1, 2, 3\}$  for all vertices  $a_1 \dots a_n$ . If  $\pi$  is any permutation of  $\{1, 2, 3\}$ , let  $0\pi = 0$ . Then  $h'(a_1 \dots a_n) = h((a_1\pi) \dots (a_n\pi))\pi^-$  is a 4-coloring; and  $h'(0 \dots 0j) = h(0 \dots 0(j\pi))\pi^- = j\pi\pi^- = j$ .

Consequently we can assume without loss of generality that  $h(0 \dots 011) = 0$ . Let  $v_k$  be the vertex  $a_1 \dots a_n$  such that  $k = (a_1 \dots a_n)_2$ . Then the sequence  $h(v_1), h(v_3), h(v_5), \dots, h(v_{2^n-1})$  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  $(0 \dots 011_0)$  and  $(\neg 0 \dots 101_3)$  *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.

	$\bar{S}_3^{(4)}$	$\bar{S}_4^{(4)}$	$\bar{S}_5^{(4)}$	$\bar{S}_6^{(4)}$	$\bar{S}_7^{(4)}$		$\hat{S}_3^{(5)}$	$\hat{S}_4^{(5)}$	$\hat{S}_5^{(5)}$	$\hat{S}_6^{(5)}$
Dir	580	200	24	12	13	Dir	460	650	160	25
Mul	580	130	33	18	17	Mul	22000	2500	250	30
Log	5900	2600	440	62	48	Log	8400	6700	1600	
Sup	2100	800	250	85	24	Sup	8700	3900	480	
Wea	140000	8900	1700	290	450	Pre	6100	4700	1200	1600
Red	9100	5000	200	20	24	Red	16000	1800	180	17
Ord	680	130	26	11	16	Ord	2500	1700	320	150



**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  $\hat{S}_n^{(3)}$ ,  $\bar{S}_n^{(4)}$ ,  $\hat{S}_n^{(5)}$  are respectively  $10^{n-11}$ ,  $10^{n-7}$ , and  $10^{n-5}$  sec; the units for  $L_q = L(J_q)$  are  $q^2 \mu\text{sec}$ . Algorithm 7.2.2.2C is totally eclipsed on the  $\hat{S}_n^{(3)}$  and  $\bar{S}_n^{(4)}$  benchmarks!)

	$\hat{S}_9^{(3)}$	$\hat{S}_{10}^{(3)}$	$\hat{S}_{11}^{(3)}$	$\hat{S}_{12}^{(3)}$	$\bar{S}_6^{(4)}$	$\bar{S}_7^{(4)}$	$\bar{S}_8^{(4)}$	$\hat{S}_5^{(5)}$	$\hat{S}_6^{(5)}$	$\hat{S}_7^{(5)}$	$\hat{S}_8^{(5)}$	$L_{1023}$	$L_{2047}$	$L_{4095}$	$L_{8191}$
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

Kissat  
Intel Xeon computer  
Biere  
clique  
permutation  
pigeonhole principle  
domain inconsistency  
direct representation  
 $\wp$ , tautology  
clique hints  
multivalued encoding  
van Dongen  
benchmarks  
competition

**247.** (a) Each of the  $n$  clusters  $\{x_{ij} \mid 0 \leq j < n\}$  is an  $n$ -clique, so their values must be a permutation of the domain. If  $i > 0$  and  $j > 0$ ,  $x_{i0} < 2$  implies  $x_{ij} \geq 2$ ; hence  $x_{i0} \geq 2$ . So the  $n-1$  variables  $\{x_{01}, \dots, x_{0(n-1)}\}$  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  $0 \leq i, j < n$ , 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  $0 \leq i, j, k < n$  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 M $\mu$  when  $n = 8$  and 90 G $\mu$  when  $n = 12$ . But the time is much, much longer when they're randomly ordered (e.g., 1880 G $\mu$  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 M $\mu$  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  $0 \leq i < n$ , where

$$\begin{aligned}
A_i &= \bigwedge_{j=0}^{(n-1)[i \neq 0]} \left( \left( \bigvee_{k=0}^{n-1} x_{ijk} \right) \wedge \bigwedge_{0 \leq k < k' < n} (\bar{x}_{ijk} \vee \bar{x}_{ijk'}) \right) & [\text{domain constraints}]; \\
B_i &= \bigwedge_{0 \leq j < j' < n} \bigwedge_{k=0}^{n-1} (i > 0? (\bar{x}_{ijk} \vee \bar{x}_{ij'k}): (\bar{x}_{j0k} \vee \bar{x}_{j'0k})) & [\text{clique constraints}]; \\
C_i &= \bigwedge_{k=0}^{n-1} (i > 0? (\bigvee_{j=0}^{n-1} x_{ijk}): (\bigvee_{j=0}^{n-1} x_{j0k})) & [\text{clique hints}]; \\
D_i &= (i > 0? \bigwedge_{j=1}^{n-1} ((\bar{x}_{i00} \vee \bar{x}_{ij1}) \wedge (\bar{x}_{i01} \vee \bar{x}_{ij0})): \wp) & [\text{constraint (ii)}].
\end{aligned}$$

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 M $\mu$ , and Algorithm 7.2.2.2C needed only 2.4 M $\mu$ . But without the clique hints the runtime is exponential—for example 270 G $\mu$  with 7.2.2.2C for  $n = 11$ . The multivalued encoding does poorly too (280 G $\mu$ ), 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  $u = (u_{n-1} \dots u_0)_2$  and  $v = (v_{n-1} \dots v_0)_2$ . It's the same problem, since  $\bar{u} = (\bar{u}_{n-1} \dots \bar{u}_0)_2 = 2^n - 1 - u$ . 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.**  $(R_{00} \vee R_{01} \vee R_{12} \vee R_{20}) \wedge (\bar{R}_{00} \vee u_0) \wedge (\bar{R}_{01} \vee v_0) \wedge (\bar{R}_{01} \vee u_0) \wedge (\bar{R}_{01} \vee v_1) \wedge (\bar{R}_{12} \vee u_1) \wedge (\bar{R}_{12} \vee v_2) \wedge (\bar{R}_{20} \vee u_2) \wedge (\bar{R}_{20} \vee v_0) \wedge (\bar{u}_0 \vee R_{00} \vee R_{01}) \wedge (\bar{u}_1 \vee R_{12}) \wedge (\bar{u}_2 \vee R_{20}) \wedge (\bar{v}_0 \vee R_{00} \vee R_{20}) \wedge (\bar{v}_1 \vee R_{01}) \wedge (\bar{v}_2 \vee R_{12})$  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  $\bar{R}_{001}$ .

**253.** (a) There are  $N = d_1 \dots d_k - G$  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  $d_1 + \dots + d_k$  clauses like the right of (81) have a total of  $d_1 + \dots + d_k + Gk$ . So the grand total is  $(3k + 1)G + d_1 + \dots + d_k$ .

**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  $R(a_1, \dots, a_k)$  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  $R'_b = \{c \mid ac \in R \text{ for some } a \text{ with } ab \notin R\}$ . 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  $a_1 \dots a_k \in R$  there's a solution with  $v_{1a_1} = \dots = v_{ka_k} = 1$ .

(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  $w_2x_2$ ,  $w_1y_0$ ,  $\dots$ ,  $y_2z_1$  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 ' $wx w:0 x:0$ ', ' $wx w:0 x:1$ ',  $\dots$ , ' $wx w:2 x:1$ '; 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.

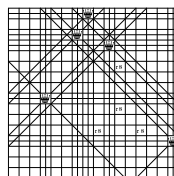
parity  
all-different  
median of three  
projections  
Historical note  
Rossi  
Petrie  
Dhar  
hidden variable  
Montanari  
positive versus negative table constraints  
negative versus positive table constraints

(i,j) False. But would be true if  $D_z$  were reduced to  $\{2, 3\}$ .

**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  $\leq 3$ .

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  $(\text{row}, \text{col}) = (1, 2)$  or  $(3, 7)$  or  $(7, 8)$ , then rotating by  $90^\circ$ ,  $180^\circ$ ,  $270^\circ$ .

**267.** Sometimes the case  $w = v$  is necessary. If, for example,  $S_c = \{u, v\}$  and  $\text{STAMP}(u) > \text{STAMP}(c)$ , 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  $\text{STAMP}(w) > \text{STAMP}(c)$ . 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  $\beta$ :

- $\text{BIND}(c)$  is the binding for which clause  $c$  lists potential supports;
- $\text{POS}(c)$  is the MEM location for the current support of  $\text{BIND}(c)$ ;
- $\text{IN}(\beta)$ , where  $\beta = (v, a)$ , is 1 if  $a$  is in  $v$ 's current domain, otherwise 0;
- $\text{LAST}(\beta)$  is the final clause  $c$  such that  $\text{MEM}[\text{POS}(c)] = \beta$ ;
- $\text{PREV}(c)$  is the previous clause  $c'$  such that  $\text{MEM}[\text{POS}(c')] = \text{MEM}[\text{POS}(c)]$ ;
- $\text{START}(c)$  is the MEM location just preceding clause  $c$ .

A stack  $S_0, S_1, \dots$  holds bindings that will soon be removed from their current domains.

- H1.** [Initialize.] Set  $\text{LAST}(\beta) \leftarrow 0$  and  $\text{IN}(\beta) \leftarrow 1$  for all bindings  $\beta$ . Also set  $c \leftarrow l \leftarrow s \leftarrow 0$ , so that the table of clauses, 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  $\{b_1, \dots, b_k\}$  be the values of  $w$  such that  $ab_j \in R(v, w)$ . If  $k = |D_w|$ , do nothing (the relation doesn't constrain  $\beta$ ). Otherwise if  $k > 0$ , set  $c \leftarrow c + 1$ ,  $\text{BIND}(c) \leftarrow \beta$ ,  $\text{START}(c) \leftarrow l$ ,  $\text{MEM}[l+j] \leftarrow (w, b_j)$  for  $1 \leq j \leq k$ ,  $l \leftarrow l + k$ ,  $\text{POS}(c) \leftarrow l$ ,  $\alpha \leftarrow \text{MEM}[l]$ ,  $\text{PREV}(c) \leftarrow \text{LAST}(\alpha)$ , and  $\text{LAST}(\alpha) \leftarrow c$ . [See (8g).] Otherwise if  $\text{IN}(\beta) = 1$ , set  $\text{IN}(\beta) \leftarrow 0$ ,  $S_s \leftarrow \beta$ ,  $s \leftarrow s + 1$ .
- H2.** [Prepare to loop.] Terminate the algorithm if  $s = 0$  (because all bindings with  $\text{IN}(\beta) = 1$  are supported). Otherwise set  $s \leftarrow s - 1$ ,  $\beta \leftarrow S_s$ , and  $c \leftarrow \text{LAST}(\beta)$ . (We need to find supports for bindings previously supported by  $\beta$ .)
- H3.** [Done with loop?] If  $c = 0$ , return to H2. Otherwise set  $c' \leftarrow \text{PREV}(c)$ ,  $\beta \leftarrow \text{BIND}(c)$ , and let  $\beta = (w, b)$ . Go to H6 if  $\text{IN}(\beta) = 0$  (because we've already deleted  $b$  from  $w$ 's domain and don't need support for it). Otherwise set  $k \leftarrow \text{POS}(c) - 1$ .
- H4.** [Done with  $c'$ ] If  $k = \text{START}(c)$ , go to H5. Otherwise set  $\alpha \leftarrow \text{MEM}[k]$ . If  $\text{IN}(\alpha) = 0$ , set  $k \leftarrow k - 1$  and repeat this step. Otherwise set  $\text{POS}(c) \leftarrow k$ ,  $\text{PREV}(c) \leftarrow \text{LAST}(\alpha)$ ,  $\text{LAST}(\alpha) \leftarrow c$ , and go to H6.

wipeout  
Weigel  
fourfold symmetry  
Lecoutre

**H5.** [Remove binding  $\beta$ .] Set  $\text{IN}(\beta) \leftarrow 0$  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):  $\overline{x_1 y_0 z_1} \Rightarrow \bar{w}_0$ ,  $\overline{x_2 y_1 z_0} \Rightarrow \bar{w}_1$ ,  $\overline{x_1 y_1 z_0} \Rightarrow \bar{w}_2$ ;  $\Rightarrow \bar{x}_0$ ,  $\overline{w_0 y_0 z_1} \wedge \overline{w_2 y_1 z_0} \Rightarrow \bar{x}_1$ ,  $\overline{w_1 y_1 z_0} \Rightarrow \bar{x}_2$ ;  $\dots$ ;  $\Rightarrow \bar{z}_2$ ; here  $x_a y_b z_c$  is a “compound” Boolean variable meaning  $x_a \wedge y_b \wedge z_c$ . Additional clauses such as  $\bar{x}_a \Rightarrow \overline{x_a y_b z_c}$ ,  $\bar{y}_b \Rightarrow \overline{x_a y_b z_c}$ ,  $\bar{z}_c \Rightarrow \overline{x_a y_b z_c}$ , complete the set.

The algorithm of exercise 268 is extended to allow “hyperbindings”  $\beta$  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

$$\begin{array}{c} \begin{array}{ccccc} & 0101 & 0122 & 1100 & 1212 & 2020 & 2211 \\ \begin{array}{c} 0 \\ 1 \\ 2 \end{array} & \left( \begin{array}{cccccc} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{array} \right) \end{array}$$

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:

$$c = 'w + x + y = 1'; \quad c' = 'x + y + z = 2'.$$

$$wxy \left\{ \begin{array}{c} \overbrace{xyz} \\ 011 \ 101 \ 110 \\ \left( \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{array} \right) \end{array} \right.$$

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. Nougier, 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  $\overline{be} = <$  has an empty left-hand side.) Then the algorithm of exercise 269 makes 67 deductions (such as  $\overline{qs} = < \Rightarrow \overline{fq} = +$ ) 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  $\overline{e} = --- \Rightarrow \overline{b} = +-+$ .) The final domains are of size 1 for the interior junctions  $\{d, g, h, i, j, k, l, n, p\}$ . In fact, *forward consistency* by itself gives excellent reduction.

Exterior junctions  $\{a, q, r\}$  of type V are left with domains of size 3; the others,  $\{b, c, e, f, m, o, s\}$ , 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  $\{ac, cd, cm\}$ ) 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  $ac = <$ ,  $be = -, \dots$ , 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  
Nougier  
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  $0 \leq k < j$ , 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  $0 \leq k < i$ ; (ii)  $r_{ik} = 1$  for some  $k < j$ ; and (iii) either  $r_{k0} \dots r_{k(j-1)} \neq 0 \dots 0$  or  $r_{k0} \dots r_{kj} = 0 \dots 00$ , 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  $1 \leq i \leq n$ . Also let  $x_{i,j}$  be secondary for  $1 \leq i \leq m$  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 ' $\# x_n : j x_{1,j} \dots x_{m,j}$ ' 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  $m < i \leq n$  and for all  $a, b, c \in \{0, 1, 2\}$  with  $(a = b \text{ or } a = c)$ .

(c) Define  $j(i)$  and  $k(i)$  in  $\binom{m}{2} \binom{m+1}{2} \dots \binom{n-1}{2} = 2^{m-n}(n-1) \overline{n-m} (n-2) \overline{n-m}$  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 gigamems, when each item  $R_i$  for  $m < i \leq n$  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 megamems.)

(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  $x_1, x_2, \dots$ .

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, *LNC3* **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  $m < i \leq n$ , perhaps different for each  $i$ . Many such generalizations are likely to be instructive.]

**280.** (a) Let  $X_i = [x_i \text{ is a sink}]$ . Then  $\mathbb{E} X_i = \Pr(X_i = 1) = q_{\max(i,m)+1} \cdots q_n$ , where  $q_i = \Pr(i \notin \{j(l), k(l)\}) = \binom{l-2}{2} / \binom{l-1}{2}$ . Hence  $\mathbb{E} X_i = \binom{\max(i,m)-1}{2} / \binom{n-1}{2}$ ; and  $S_{m,n} = \sum_{i=1}^n \mathbb{E} X_i = \frac{n}{3}(1 + 2m^3/n^3)$ .

(b) Set  $d \leftarrow n - m - 1$ ,  $a_{0,0,m} \leftarrow 1$ . Then for  $1 \leq i \leq d$  and  $0 \leq j \leq i$ , set

$$a_{i,j,m} \leftarrow \binom{m+i-j-2}{2} [j \neq i] a_{i-1,j,m} + ((\binom{m+i-1}{2} - \binom{m+i-j-1}{2})) [j \neq 0] a_{i-1,j-1,m}.$$

Then there are  $a_{i,j,m}$  cases in which  $x_1$  is connected to exactly  $j$  of the variables  $\{x_{m+1}, \dots, x_{m+i}\}$ . 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  $C_{m,n} \leq C_{2,n} \leq C_{m,n} + m$ , 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  $0 \leq k < n$ . Then  $x_0 \dots x_{n-1}$  is a solution if and only if  $(y_0 + \dots + y_{n-1}) \bmod d = 0$ . 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  $D_{j+1} = \{x_j, x_{j+1}\}$ ;  $D_k = \{0, \dots, d-1\}$  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  $d = n - 1$ .]

Hwang  
Mitchell  
backtracking  
exponentially large  
 $d$ -way branch  
forward consistency  
Prover–Delayer game  
Delayer  
binary branching  
Janson



**284.** There are exactly  $d$  ways to insert ‘ $n$ ’ into such a permutation of  $\{1, \dots, n-1\}$ , namely at the beginning or after one of  $\{n-1, n-2, \dots, n-d+1\}$ . So the answer is  $d!d^{n-d}$ , by induction. [The number of permutations with exactly  $k$  instances of  $p_{j+1} \geq p_j + d$  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  $j \geq 1$  is minimum with  $j \notin S$ . Remove  $p_1 \dots p_j$  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  $p_1 \dots p_n$  such that  $\{p_1, \dots, p_k\}$  is an *interval*, for  $1 \leq k \leq n$ ; 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  $1 \leq j \leq n$ ; also introduce  $(n-1)(n-2)$  secondary items  $j.k$  for  $1 \leq j < n$  and  $1 \leq k < m = n-1$ , which correspond to items  $y_k$  of the pairwise encoding trick that enforces  $p_{j+1} \leq p_j + 1$ . For each  $1 \leq j, k \leq n$  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  $t \leftarrow t \& (t-1)$ . (This contributes  $\alpha_{k-2}$ .) If  $j < n$  and  $k \leq m$ , set  $t \leftarrow -k$ ; then while  $t > -m$  include  $j.(-t)$  and set  $t \leftarrow t \& (t-1)$ . (This contributes  $\beta_{k-1}$ .) 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  $q_{k+1} \leq q_k + 1$ : Introduce new secondary items  $j.k$  for  $1 \leq j < n$  and  $1 \leq k < m$ . 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  $t_n = t_n(1) + t_n(2) + \dots$  is the desired number, where  $t_n(m)$  is the number of patterns with  $m$  in the upper right corner, we have the recurrences  $t_n(m) = a_n(m, m) + \sum_{m'=1}^{2n-m} b_n(m, m; m', m')$ ;  $a_n(l, m) = (l < 2? 0: l > 2n? 0: l = 2n? 1: l = 2? t_{n-1} - 2t_{n-1}(m) + a_{n-1}(m, m): a_{n-1}(l-2, m) + 2 \sum_{m'=1}^{2n-l} b_{n-1}(l-2, m; m', m'))$ ;  $b_n(l, m; l', m') = (m = m'? 0: l < 1? 0: l' < 1? 0: l + l' > 2n? 0: l + l' = 2n? 1: l = l'? t_{n-l} - t_{n-l}(m) - t_{n-l}(m') + b_{n-l}(m, m; m', m'): l < l'? a_{n-l}(l' - l, m') + \sum_{m''=1}^{2n-l-l'} b_{n-l}(m'', m'; l' - l, m') [m'' \neq m]: b_n(l', m'; l, m))$ . 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  $m \neq m'$  at the right, respectively parts of an  $l$ -omino and an  $l'$ -omino. Hence  $(t_1, t_2, \dots, t_{145}) = (1, 5, 33, 138, 715, 3524, \dots, 51376, 52565, 68766, 30928, 69800, 54061, 86098, 15559, 89493, 34784, 20112, 85272, 12992, 22603, 93822, 34860, 83493, 24519, 70607, 50508)$ . (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

**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  $0 \leq i < m$ ,  $0 \leq j < n$ .) 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  $C = c_1 + c_2 + \dots + c_s$  "clues," where  $s$  is the maximum label. Then every potential  $d$ -omino has  $d \leq \max(N - C, s)$ . 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  $c_1^+ = c_1$ ;  $c_2^+ = 2(c_2 - \text{the number of pairs of adjacent } 2\text{s})$ ; and for  $d \geq 3$ ,  $c_d^+ = d\lceil c_d/d \rceil + [c_d \bmod d = 0]$  and the  $d$ s aren't disjoint  $d$ -ominoes).

**292.**

	335555666	6244221244	1334141224	2244446633
221221	314159266	6244888244	2344144434	5153366313
1ee312	444999236	6688866688	2143122334	5553616345
e1e332	555899932	2684446888	1223133114	4444646345
(a) eeeee	535897932	2684226882	21i3i3i122	5566643145
(b) 233e1e	338877773	6448446482	2iiiii444	3565543345
(c) 213ee1	888772643	6248441448	3444413324	3552542215
(d) 122122	338462643	6248862248	33221i3421	3442513334
	344466644	6688666688	1444ii4433	2344544414
		6882268888	3334224223	2332242244

[Puzzle (c), by Chris Green, was posted at [puzzleparade.blogspot.com](http://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](http://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  $4 \times 4$  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.

4uuu	4uuu	4uuu	4uuu	uuu1	uuu3	uu5u	1u33	u1u1	u3u2	uu2u	uu41
uuu4	uuu4	uuu4	uuu4	u15u	uu3u	u5u5	55uu	2u4u	4u4u	3uu1	2uuu
1uuu	1uuu	2uuu	2uuu	u51u	u3uu	5u5u	u544	u5u4	u2u4	uuu4	u3uu
uuu3	uuu2	uuu2	uuu5	1uuu	3u3u	5u5u	5544	5u5u	4u4u	u2uu	3uu2
(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)	(xi)	(xii)

**Fig. A-16.** A gallery of interesting  $4 \times 4$  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

author  
downloadable programs  
Green  
Saarinen  
**@kobouzu17**  
symmetry



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  $4 \times 4$  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.)

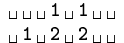
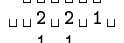
**294.** (Solution by N. Beluhov.) Let  $P$  be a maximal rookwise connected subset of the solution, having no labels  $\leq s$ . Every element of  $P$  must have the same label  $d = |P|$ , because the solution is unique. Let  $T$  be a spanning tree of  $P$ . Every edge  $u - v$  of  $T$  partitions  $T$ , 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  $u_1, \dots, u_k$ , with corresponding subtree sizes  $s_1 \geq \dots \geq s_k$ . Then  $s_1 \leq s_2 + \dots + s_k$  by Eq. 2.3.4.4-(7). Hence we have  $|P_v| = s_2 + \dots + s_k + 1 > |P_{u_1}| = s_1$  in the decomposition  $P = P_{u_1} \cup P_v$ . It follows that  $s_1 \leq s$ ; and  $d \leq ks + 1 \leq 4s + 1$ .

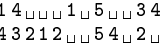
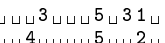
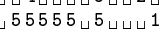
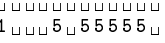

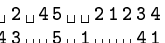
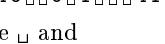
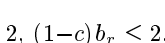
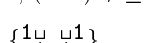
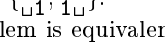
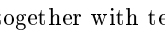
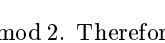
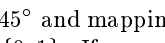
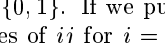
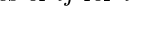











*Case 2,*  $T$  has two centroids,  $u - v$ . We may suppose that  $u = i(j-1)$  and  $v = ij$ , coordinatewise. If  $T$  also contains both of the edges  $u' = (i-1)(j-1) - u$  and  $v' = (i-1)j - v$  (or, similarly, if  $T$  contains both  $u' = (i+1)(j-1) - u$  and  $v' = (i+1)j - v$ ), we get a decomposition  $P = P_{u'} \cup P_{v'} \cup P_{uv}$  by deleting those edges, where  $|P_{u'}| \leq s$  and  $|P_{v'}| \leq s$ . On the other hand, if in the original tree  $T$  we delete only edge  $u - v$  and replace it with edge  $u' - v'$ , 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  $u - v$  and  $u' - v'$ , therefore,  $d = |P_{uv}| + |P_{u'v'}| \leq 2|P_{u'v'}| \leq 2|P_{u'}| + 2|P_{v'}| \leq 4s$ .


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  $\leq s$ . Hence  $d \leq 4s + 2$ .

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  $3 \times 3$  grid.

Here's  $s = 2$ : ; and here's  $s \geq 4$ , shown (with many redundant clues, for clarity) for  $s = 5$ : 

for  $s = 3$ , see exercise 292(a);

1 4  1 5  3 4  
4 3 2 1 2  5 4  2   
 3  5  3 1   
 4  5  5  2   
 5 5 5 5  5  1   
  5  5 5 5 5   
 2   5   4    
 1 3  5   3    
 2  4 5   2 1 2 3 4  
4 3  5  1   4 1

**295.** (Solution by N. Beluhov and P. Mebane.) When  $m = 1$  they are  and

$$1^a \text{ } \text{ } ^{b_1} 1 \text{ } \text{ } ^{b_2} 1 \dots 1 \text{ } \text{ } ^{b_r} 1^c \quad \text{for } r \geq 1, 0 \leq a, c \leq 1, 2 \leq b_j \leq 4, (1-a)b_1 \leq 2, (1-c)b_r \leq 2.$$

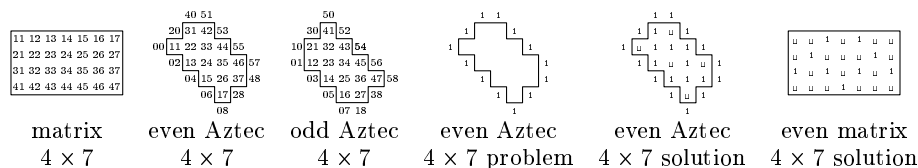
When  $m = 2$  they're  $\alpha \text{ } \text{ } ^\beta 1 \text{ } \text{ } ^\gamma$  for  $r \geq 0$ ,  $\alpha, \gamma \in \{\epsilon, \text{ } \text{ } ^1, \text{ } \text{ } ^1\}$ ,  $\beta_j \in \{\text{ } \text{ } ^1, \text{ } \text{ } ^1\}$ .

Now suppose  $3 \leq m \leq n$ . 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  $45^\circ$  and mapping  $ij \mapsto (m-i+j+\delta, i+j-\delta)/2$  for  $1 \leq i \leq m$ ,  $1 \leq j \leq n$ , 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  $45^\circ$

- kingwise adjacent
- halo
- Beluhov
- Gerdjikov
- Aztec rectangle
- Toroidal

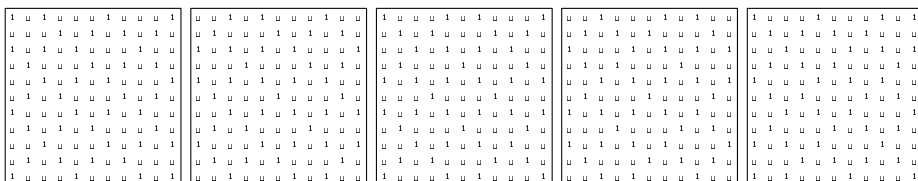


The two Aztec rectangles are isomorphic when  $mn$  is even. The solutions when  $m$  is even and  $\delta = 0$  are the first  $n$  columns of  $A_m$  when  $n \bmod (m+1) \in \{1, m\}$ , and of  $B_m$  when  $n \bmod (m+1) \in \{m-2, m\}$ , where  $A_m$  and  $B_m$  are infinite matrices


[illegible]

whose columns have period length  $2m + 2$ , illustrated here for  $m = 6$ .

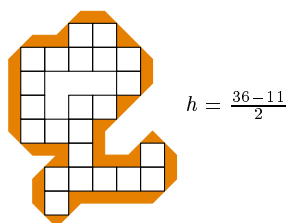
Solutions exist for odd  $m$  only when  $n$  is odd and  $\delta = 0$ . Two easy constructions always work: The even-numbered rows are  $\sqcup 1 \sqcup 1 \sqcup 1 \sqcup 1 \dots$ , and the odd-numbered rows alternate between  $1 \sqcup \sqcup \sqcup 1 \sqcup \sqcup \sqcup 1 \dots$  and  $\sqcup \sqcup 1 \sqcup \sqcup \sqcup 1 \sqcup \sqcup \sqcup 1 \dots$ . 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  $h \leq t/2$  implies  $d \leq b_t = \lfloor (t^2 + 4)/8 \rfloor$ . This bound is in fact sharp for all  $t \geq 4$ , with the maximum  $d$  attained by a  $\lfloor t/2 \rfloor \times \lfloor t/2 \rfloor$  Aztec rectangle with  $\delta = 0$  (see exercise 295).



$$h = \frac{36-1}{2}$$



Polyominoes without common edges have disjoint halos. Therefore if an  $n \times n$  fillomino pattern  $\Phi_n$  contains  $k$   $d$ -ominoes, we must have  $k(d + h_d) \leq n^2 + O(n)$ , where  $h_d = \lceil \sqrt{8d - 4} \rceil / 2$  is the smallest possible halo area of a  $d$ -omino. Consequently  $\#_d(\Phi_n) / n^2 \leq d / (d + h_d) + O(1/n)$ , and we have  $\delta_d \leq d / (d + h_d)$ . These upper bounds are:

$$d = \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 \\ \frac{d}{d-h_d} & = \frac{1}{2} & \frac{1}{2} & \frac{6}{11} & \frac{4}{7} & \frac{5}{8} & \frac{12}{19} & \frac{7}{11} & \frac{2}{3} & \frac{2}{3} & \frac{29}{20} & \frac{16}{17} & \frac{12}{13} & \frac{28}{30} & \frac{30}{41} & \frac{8}{11} & \frac{17}{23} & \frac{3}{4} & \frac{38}{51} & \frac{40}{53} & \frac{42}{55} & \frac{22}{30} & \frac{23}{31} & \frac{24}{32} & \frac{25}{35} \end{matrix}$$

Toroidal constructions (with appropriate offsets) give lower bounds for small  $d$ :

15	122	322	144	199	bbb1
( $d=1$ ) 555;	( $d=2$ ) 244;	( $d=3$ ) 331;	( $d=4$ ) 4144;	( $d=9$ ) 9999922;	( $d=11$ ) bbbbb333;
151	244	322	4413	2299	bbb1
		331	1433		
	ggg1	jjjjj2	mmmm2		
( $d=14$ ) eeee1	ggggg5	jjjj2	mmmm2		
eeeeee4444;	ggggg5555;	jjjjj1	mmmm7		
eeee1	ggg1	jjjjj55555	mmmm7		
			mmmm77777		

Stappers  
Demaine  
Friedman  
Aztec rectangles  
Snyder  
González-Morris  
biaxial  
axial  
Break symmetry  
unique  
Kurchan  
Lecoutre  
Szymanek  
tuples

(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  $\delta_d \geq d/(b_{2k-1} + k)$  when  $1 < b_{2k-2} < d \leq b_{2k-1}$ ;  $\delta_d \geq d/(b_{2k} + k)$  when  $b_{2k-1} < d \leq b_{2k}$ ; 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
$\delta_d \geq \frac{3}{8}$	$\frac{4}{9}$	$\frac{1}{2}$	$\frac{8}{15}$	$\frac{5}{8}$	$\frac{3}{5}$	$\frac{7}{12}$	$\frac{2}{3}$	$\frac{9}{14}$	$\frac{2}{3}$	$\frac{11}{16}$	$\frac{2}{3}$	$\frac{13}{18}$	$\frac{7}{10}$	$\frac{5}{7}$	$\frac{16}{23}$	$\frac{17}{24}$	$\frac{3}{4}$	$\frac{19}{27}$	$\frac{5}{7}$	$\frac{3}{4}$	$\frac{22}{31}$	$\frac{23}{32}$	$\frac{3}{4}$	$\frac{25}{32}$

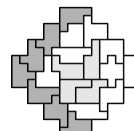
Are these the true values of  $\delta_d$ , or can some of them be increased?

**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.)

1	7	5	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1
1	5	1	1	1	1	1	1	1	1
5	7	1	1	1	1	1	1	1	1
8	3	2	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

**298.** (a) 9; 4; 7. (See exercise 7.2.2.1–386. Adding a domino to a  $2 \times 3$  hexomino gives  $(0, 1, 1, 1, 4)$  octominoes with respectively  $(180^\circ, \text{biaxial}, \text{axial}, \text{diagonal}, \text{no})$  symmetry; adding two nonadjacent monominoes gives  $(2, 1, 3, 0, 7)$ .)

(b) Using answer 7.2.2.1–266, piece  $\text{A}$  is, for example,  $0[1-4] \ 1[0-3]$ . Break symmetry by restricting piece  $\text{A}$ , 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  $\sigma$  in Fig. 117 has the respective values  $(1, 2, \dots, 14)$ . Therefore entries such as  $\begin{smallmatrix} y \\ 1 \end{smallmatrix}$  would improperly be omitted from the trail in lines 11 and 16. And the omission of  $\begin{smallmatrix} y \\ 5 \end{smallmatrix}$  on line 29 would give the incorrect value  $y = 4$  on line 30.

(b) If we associate a fresh value of  $\sigma$  to each node, the “correct” current stamps corresponding to  $(1, \dots, 14)$  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  $\text{STAMP}(x)$  are restored when backtracking. Backtracking should also restore  $\sigma$  to its previous state. For example, the trail at line 06 would be  $\begin{smallmatrix} x & y \\ 0 & 0,0 \end{smallmatrix} \begin{smallmatrix} x & y \\ 5 & 1,1 \end{smallmatrix} \begin{smallmatrix} x & y \\ 8 & 1,1 \end{smallmatrix}$ . (That’s  $1 + 3 + 3 + 1 + 3 + 3 + 1 + 3 = 18$  entries instead of  $1 + 2 + 2 + 2 + 2 + 2 + 1 + 2 + 2 + 2 + 1 + 2 = 19$ ; 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)  $\text{BITS}(v) = 2^d - 1$ . (b)  $\text{NEXT}_v(a) = a + 1$  and  $\text{IN}_v[a] = 1$  and  $\text{PREV}_v(a + 1) = a$ , for  $0 \leq a < d$ ;  $\text{NEXT}_v(d) = \text{IN}_v[d] = 0$ ;  $\text{PREV}_v(0) = d$ . (c)  $\text{DOM}_v[k] = \text{IDOM}_v[k] = k$ , for  $0 \leq k < d$ ;  $\text{SIZE}(v) = d$ .

**302.** (a)  $\rho(\text{BITS}(v) + 2^d)$ . (b)  $\text{NEXT}(d)$ . (c) Initialize  $\text{MIN}(v)$  to 0. If deleting  $a = \text{MIN}(v)$  in (110), also do this: Set  $t \leftarrow \text{MIN}(v) + 1$ ,  $\text{MIN}(v) \leftarrow d$ . For  $0 \leq k < \text{SIZE}(v)$ , if  $\text{DOM}_v[k] < \text{MIN}(v)$ , set  $\text{MIN}(v) \leftarrow \text{DOM}_v[k]$ , and break out of this loop if  $\text{MIN}(v) = t$ .

**303.** True. (See Eq. 4.1–(5).)

**304.** This algorithm uses an approach similar to Quicksort (Algorithm 5.2.2Q) to exchange elements of  $\mathbf{D}$  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 “ $\text{trail}(x)$ ” means “push the pair (address of  $x$ , value of  $x$ ) onto the trail.”

**R1.** [Initialize.] Set  $i \leftarrow 0$ ,  $s \leftarrow \mathbf{S}$ ,  $j \leftarrow s - 1$ .

**R2.** [Done?] (At this point all cases  $k$  for  $k < i$  and  $k > j$  are done.) If  $i > j$ , go to R7.

**R3.** [Try  $k = \mathbf{D}[i]$ .] Set  $k \leftarrow \mathbf{D}[i]$  and  $x \leftarrow b_k \& b'_k$ . If  $x = 0$ , go to R4. Otherwise, if  $x \neq b_k$ ,  $\text{trail}(b_k)$  and set  $b_k \leftarrow x$ . Set  $i \leftarrow i + 1$  and return to R2.

**R4.** [Done?] (We know that  $b_{\mathbf{D}[i]}$  should be zero.) If  $i = j$ , go to R7.

**R5.** [Try  $k = \mathbf{D}[j]$ .] Set  $k \leftarrow \mathbf{D}[j]$  and  $x \leftarrow b_k \& b'_k$ . If  $x = 0$ , set  $j \leftarrow j - 1$  and return to R4. Otherwise, if  $x \neq b_k$ ,  $\text{trail}(b_k)$  and set  $b_k \leftarrow x$ .

**R6.** [Swap.] Set  $\mathbf{D}[i] \leftrightarrow \mathbf{D}[j]$ ,  $i \leftarrow i + 1$ ,  $j \leftarrow j - 1$ , and return to R2.

**R7.** [Terminate.] If  $\mathbf{S} \neq i$ ,  $\text{trail}(\mathbf{S})$  and set  $\mathbf{S} \leftarrow i$ . ■

**305.** Let the tuples of  $R$  be  $\tau_i$  for  $0 \leq i < t$ . And for  $1 \leq j \leq k$ ,  $0 \leq a < d$ , let  $r[j, a]$  be the bitset whose  $i$ th bit is  $[v_j = a \text{ in } \tau_i]$ . (Thus, if  $R$  is the relation (78), we can let  $(\tau_0, \dots, \tau_6) = (000, 001, 010, 012, 020, 121, 211)$ , with  $t = 7$ ; then  $r[1, 0] = 1111100$ ,  $r[1, 1] = 0000010$ ,  $r[1, 2] = 0000001$ ,  $\dots$ ,  $r[3, 1] = 0100011$ ,  $r[3, 2] = 0001000$ .)

First we need to remove invalid tuples from the current  $R$ . For each  $j$  with  $\text{OSIZE}_j \neq \text{SIZE}_j$ , the values  $\text{DOM}_j[k]$  for  $\text{SIZE}_j \leq k < \text{OSIZE}_j$  have been deleted from  $D_j$  since  $R$ 's last propagation. So we intersect  $b$  with  $b'$ , where  $b'$  is either (i)  $\sim \text{OR}\{r[j, \text{DOM}_j[k]] \mid \text{SIZE}_j \leq k < \text{OSIZE}_j\}$  or (ii)  $\text{OR}\{r[j, \text{DOM}_j[k]] \mid 0 \leq k < \text{SIZE}_j\}$ ; use (i) if  $\text{OSIZE}_j - \text{SIZE}_j < \text{SIZE}_j$ , otherwise use (ii). This bitwise OR is computed by looking only at the  $\mathbf{S}$  words of  $r[j, a]$  whose indices appear in the first  $\mathbf{S}$  locations of  $\mathbf{D}$ .

Then we need to filter the domains. For each  $j$  with  $\text{SIZE}_j > 1$ , and each  $a = \text{DOM}_j[k]$  for  $0 \leq k < \text{SIZE}_j$ , 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  $\mathbf{S}$  entries of  $\mathbf{D}$ , 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 H       Verhaeghe (University of Louvain, 2021), viii + 169 pages.]

**306.** (a)  $t = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16 \ 17 \ 18 \ 19 \ 20 \ 21 \ 22 \ 23 \ 24 \ 25 \ 26 \ 27 \ 28 \ 29$   
 $l_t = 0 \ 1 \ 2 \ 1 \ 0 \ 1 \ 2 \ 3 \ 2 \ 1 \ 2 \ 3 \ 4 \ 3 \ 4 \ 5 \ 4 \ 3 \ 2 \ 1 \ 2 \ 3 \ 2 \ 3 \ 4 \ 3 \ 2 \ 1$   
 $s_t = 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 2 \ 2 \ 0 \ 2 \ 4 \ 0 \ 4 \ 2 \ 2 \ 3 \ 2 \ 1 \ 2 \ 2 \ 2 \ 3 \ 2 \ 3 \ 2 \ 1$

(b) Initially  $p_0 \leftarrow 0$ ,  $q_0 \leftarrow \infty$ ,  $r \leftarrow 0$ . For  $t = 0, 1, \dots$ , 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  $[0 \dots \infty]$ ,  $[1 \dots 2]$ ,  $[3 \dots 3]$  because  $t = 7$  is a valley at level 3. They're next updated to  $[0 \dots \infty]$ ,  $[1 \dots 2]$ ,  $[2 \dots 3]$ . When eventually  $t = 25$  they're  $[0 \dots \infty]$ ,  $[1 \dots 2]$ ,  $[2 \dots 5]$ ,  $[3 \dots 3]$ .

(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  $8 + 7 + 5 + 3 + \dots + 1 = 53 = 2 \sum_{j=1}^r (q_j - p_j + 1) + p_r - 1$ . (In general the shortest goes from 0 down to  $q_1$ , then up to  $p_1 - 1$ ,  $\dots$ , down to  $q_r$ , up to  $p_r - 1$ .)

(d)  $t = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16 \ 17 \ 18 \ 19 \ 20 \ 21 \ 22 \ 23 \ 24 \ 25 \ 26 \ 27 \ 28 \ 29$   
 either  $\begin{cases} l_t = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 4 \ 5 \ 6 \ 5 \ 6 \ 7 \ 6 \ 7 \ 8 \ 7 \ 8 \ 7 \ 8 \ 7 \ 8 \ 7 \ 6 \ 5 \ 4 \ 3 \ 2 \ 1 \\ s_t = 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 5 \ 0 \ 5 \ 6 \ 0 \ 6 \ 7 \ 0 \ 7 \ 8 \ 7 \ 8 \ 7 \ 8 \ 7 \ 6 \ 5 \ 0 \ 0 \ 0 \ 0 \end{cases}$   
 or  $\begin{cases} l_t = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 5 \ 6 \ 7 \ 6 \ 7 \ 8 \ 7 \ 8 \ 7 \ 8 \ 7 \ 8 \ 7 \ 8 \ 7 \ 6 \ 5 \ 4 \ 3 \ 2 \ 1 \\ s_t = 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 6 \ 0 \ 6 \ 7 \ 0 \ 7 \ 8 \ 7 \ 8 \ 7 \ 8 \ 7 \ 8 \ 7 \ 6 \ 0 \ 0 \ 0 \ 0 \end{cases}$

(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,  $441 \pm 200$ . By exercise 2.3.4.5-5, the average of  $l_t$  is exactly  $((m+1)4^m - (2m+1)\binom{2m}{m})/(2m\binom{2m}{m}) = \frac{1}{2}\sqrt{\pi m} + O(1)$ ; this is  $\approx 885$  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  $m \rightarrow \infty$  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  $\{x_i = a_i \mid 1 \leq i \leq M_{ja}\}$ .)

**G3.** [Begin further tests.] Set  $k \leftarrow M_{ja}$ .

**G4.** [Check relations with  $x_k$ .] If  $(a_1, \dots, a_k, a)$  doesn't satisfy all constraints between  $(x_1, \dots, x_k)$  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  $\text{NODE}[x]$ , 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  $\alpha$  on the first line, set  $i \leftarrow i + 1$ ,  $\text{SIZE}(4i) \leftarrow 0$ ,  $\text{NAME}(4i) \leftarrow \alpha$ . If  $\alpha$  names the first secondary item, also set  $s \leftarrow i$ . (As in the text,  $\text{SIZE}(t) \equiv \text{SET}[t - 1]$ ,  $\text{POS}(t) \equiv \text{SET}[t - 2]$ ,

Catalan numbers  
 unary constraint  
 Gaschnig  
 IJCAI: Proc Int Joint Conf on AI  
 Dechter  
 color  
 list heads  
 secondary items  
 author  
 ITEM

and  $\text{NAME}(t)$  occupies  $\text{SET}[t-4]$  and  $\text{SET}[t-3]$ . We're temporarily using only four slots of  $\text{SET}$  for each item.) At the end, set  $\text{ACTIVE} \leftarrow i$ ,  $\text{ITM}(0) \leftarrow 0$ .

**I2.** [Read an option.] Go to I3 if no input remains. Otherwise let the next line of input contain the distinct item names  $\alpha_1, \dots, \alpha_k$ , with respective colors  $c_1, \dots, c_k$  (where  $c_j = 0$  if  $\alpha_j$  has no color). Complain if a primary item is colored, or if all items are secondary. Do the following for  $1 \leq j \leq k$ : Find the index  $i_j$  for which  $\text{NAME}(4i_j) = \alpha_j$ , using an algorithm from Chapter 6. Set  $t \leftarrow \text{SIZE}(4i_j)$ ,  $\text{SIZE}(4i_j) \leftarrow t+1$ ,  $\text{ITM}(x+j) \leftarrow i_j$ ,  $\text{LOC}(x+j) \leftarrow t$ ,  $\text{CLR}(x+j) \leftarrow c_j$ . Finally, adjust the spacers by setting  $\text{LOC}(x) \leftarrow k$ ,  $x \leftarrow x+k+1$ ,  $\text{ITM}(x) \leftarrow -k$ . Repeat step I2.

**I3.** [Initialize ITEM.] Set  $k \leftarrow 0$ ,  $j \leftarrow 4$ . While  $k < \text{ACTIVE}$ , set  $\text{ITEM}[k] \leftarrow j$ ,  $k \leftarrow k+1$ ,  $j \leftarrow j+4+\text{SIZE}(4k)$ . If  $s < 0$ , set  $\text{SECOND} \leftarrow j$  and  $s \leftarrow \text{ACTIVE}$ .

**I4.** [Expand SET.] While  $k > 0$ , do the following: Set  $j \leftarrow \text{ITEM}[k-1]$  and  $\text{SIZE}(j) \leftarrow \text{SIZE}(4k)$ ,  $\text{POS}(j) \leftarrow k-1$ ,  $\text{NAME}(j) \leftarrow \text{NAME}(4k)$ ; if  $\text{SIZE}(j) = 0$  and  $k < s$ , terminate (primary  $\text{ITEM}[k-1]$  has no options); if  $k = s$  set  $\text{SECOND} \leftarrow j$ . Set  $k \leftarrow k-1$ .

**I5.** [Adjust NODE.] For  $x' = 1, 2, \dots, x-1$ , do the following if  $\text{ITM}(x') \geq 0$ : Set  $i \leftarrow \text{ITEM}[\text{ITM}(x')-1]$ ,  $j \leftarrow i+\text{LOC}(x')$ ,  $\text{ITM}(x') \leftarrow i$ ,  $\text{LOC}(x') \leftarrow j$ ,  $\text{SET}[j] \leftarrow x'$ . ■

**312.** Set  $\theta \leftarrow \infty$ . For  $0 \leq k < \text{ACTIVE}$ , do the following steps if  $\text{ITEM}[k] < \text{SECOND}$ : Set  $\lambda \leftarrow \text{SIZE}(\text{ITEM}[k])$ ; if  $\lambda = \theta$  and  $\text{ITEM}[k] < i$ , set  $i \leftarrow \text{ITEM}[k]$ ; if  $\lambda < \theta$ , set  $i \leftarrow \text{ITEM}[k]$ ,  $\theta \leftarrow \lambda$ , 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  $\theta$  will never be zero, although it could be zero in exercise 7.2.2.1-9.)

**313.**  $i = 23$  (meaning  $\mathbf{x}$ ) and  $c = \mathbf{B}$  will leave  $\mathbf{q}$  with no options.

**314.** Item 11 ( $\mathbf{q}$ ) is selected in step C2, deactivated in step C3, and hidden with  $\text{OACTIVE} = \text{ACTIVE} = 4$  in step C4. (Thus the options containing  $\mathbf{q}$ , represented by nodes  $2 = \text{SET}[11]$  and  $14 = \text{SET}[12]$ , leave the option lists for items  $23 = \text{ITM}(3)$ ,  $31 = \text{ITM}(4)$ ,  $4 = \text{ITM}(1)$ ,  $23 = \text{ITM}(15)$ .) Step C5 sets  $\text{TRAIL}[0] \leftarrow (4, 2)$ ,  $\text{TRAIL}[1] \leftarrow (31, 2)$ ,  $\text{TRAIL}[2] \leftarrow (17, 2)$ ,  $\text{TRAIL}[3] \leftarrow (23, 2)$ , and  $y_1 \leftarrow t \leftarrow 4$ . Step C6 tries option  $x_0 \leftarrow 2$ , deactivating items 23, 31, 4. Then C7, with  $\text{OACTIVE} = 4$  and  $\text{ACTIVE} = 1$ , hides (23, 0) and jumps to C11 while trying to hide (31, 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>i</i> SET[ <i>i</i> ]			<i>i</i> SET[ <i>i</i> ]										
LNAME	0	p	• 17	17		<i>k</i> :	0	1	2	3	4		
RNAME	1		18	7		ITEM[ <i>k</i> ]:	17	4	31	23	11	ACTIVE = 3	
POS	2	1	LNAME	19	x	<i>x</i> :	0	1	2	3	4	5	6
SIZE	3	1	RNAME	20		ITM( <i>x</i> ):	0	4	11	23	31	−4	4
• 4	6		POS	21	3	LOC( <i>x</i> ):	4	6	11	26	33	4	4
5	11		SIZE	22	2	CLR( <i>x</i> ):	—	0	0	0	A	—	0
6	1		• 23	12									
LNAME	7	q	24	8		<i>x</i> :	7	8	9	10	11	12	13
RNAME	8		25	15		ITM( <i>x</i> ):	17	23	31	−4	4	23	−2
POS	9	4	26	3		LOC( <i>x</i> ):	18	24	32	2	5	23	2
SIZE	10	2	LNAME	27	y	CLR( <i>x</i> ):	0	A	0	—	0	B	—
• 11	2		RNAME	28									
12	14		POS	29	2	<i>x</i> :	14	15	16	17	18	19	
LNAME	13	r	SIZE	30	2	ITM( <i>x</i> ):	11	23	−2	17	31	−2	
RNAME	14		• 31	18		LOC( <i>x</i> ):	12	25	2	17	31	—	
POS	15	0	32	9		CLR( <i>x</i> ):	0	A	—	0	B	—	
SIZE	16	2	33	4									

(Hmm; why are  $\text{SET}[22] = 2$  and  $\text{SET}[23] = 12$ ? Answer: When  $\mathbf{x}$  was purified by  $\text{hide}(23, \mathbf{A})$  in C7, option ' $\mathbf{p} \mathbf{x} : \mathbf{B}$ ' was deleted from  $\mathbf{p}$ 's list but not  $\mathbf{x}$ 's. The other two options involving  $\mathbf{x}$  had already been deleted from  $\mathbf{x}$ 's list by  $\text{hide}(11, 0)$  in step C4.)



**315.** Suppose there's a primary item  $i' \neq i$  whose options all involve  $i$ . Then  $\text{hide}(i, 0)$  will remove all options of  $i'$ . If it sees  $\text{FLAG} = 0$  when  $\text{SIZE}(i')$  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  $\text{SIZE}(i)$ . 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  $\text{SIZE}(i) = 0$ .

**316.** Before testing  $\lambda$  versus  $\theta$  in answer 312, go to C10 if  $\lambda = 0$  (see exercise 315). Otherwise add a large constant to  $\lambda$  if  $\lambda > 1$  and  $\text{NAME}(\text{ITEM}[k])$  doesn't begin with '#'.

**317.** The secondary item  $i'$  is inactive, so its option list has already been purified.

**318.** Let  $\text{FORCE}$  be an array whose size is at least the number of primary items. After setting  $s'$  in (118), say "If  $s' = 1$  and  $\text{FLAG} = 0$  and  $i' < \text{SECOND}$  and  $\text{POS}(i') < \text{ACTIVE}$ , set  $\text{FORCE}[f] \leftarrow i'$ ,  $f \leftarrow f + 1$ ." Also, before setting  $\text{FLAG}$  in (118), set  $f \leftarrow 0$ .

Steps C1<sup>+</sup> through C11<sup>+</sup> are identical to steps C1 through C11, except that C1<sup>+</sup> also sets  $f \leftarrow 0$ , and that C2<sup>+</sup> and C8<sup>+</sup> are revised (including new intermediate steps):

**C2<sup>+</sup>.** [Choose  $i$ .] Set  $\theta \leftarrow \infty$ . For  $0 \leq k < \text{ACTIVE}$ , do the following steps if  $\text{ITEM}[k] < \text{SECOND}$ : Set  $\lambda \leftarrow \text{SIZE}(\text{ITEM}[k])$ ; then if  $\lambda = 1$ , set  $\text{FORCE}[f] \leftarrow \text{ITEM}[k]$ ,  $f \leftarrow f + 1$ ; otherwise if  $\lambda = \theta$  and  $\text{ITEM}[k] < i$ , set  $i \leftarrow \text{ITEM}[k]$ ; otherwise if  $\lambda < \theta$ , set  $i \leftarrow \text{ITEM}[k]$ ,  $\theta \leftarrow \lambda$ .

**C2.1<sup>+</sup>.** [Forced?] If  $f > 0$ , set  $f \leftarrow f - 1$ ,  $i \leftarrow \text{FORCE}[f]$ , and go to C8.2<sup>+</sup>. Otherwise if  $\theta = \infty$ , go to C9<sup>+</sup>.

**C8<sup>+</sup>.** [Advance to the next level.] Set  $l \leftarrow l + 1$ .

**C8.1<sup>+</sup>.** [Not forced?] If  $f = 0$ , go to C2<sup>+</sup>. Otherwise set  $f \leftarrow f - 1$ ,  $i \leftarrow \text{FORCE}[f]$ , and repeat step C8.1<sup>+</sup> if  $\text{POS}(i) \geq \text{ACTIVE}$ .

**C8.2<sup>+</sup>.** [Force a move.] Perform steps C3<sup>+</sup> and C4<sup>+</sup>. Then set  $y_{l+1} \leftarrow t$  and go to C6<sup>+</sup>. ■

**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  $\text{SIZE}(i')$  by  $[i' \in o_1]$ , for each item  $i'$ . Hence every active primary item  $i'$  of the remaining problem will have  $\text{SIZE}(i') \geq d - 1 = \text{SIZE}(i)$ . Furthermore, if  $\text{SIZE}(i') = \text{SIZE}(i)$ , we'll have  $\text{POS}(i') \geq \text{POS}(i)$ .

**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<sup>+</sup>, except that it uses two additional arrays,  $d_0 d_1 \dots d_T$  and  $\text{LS}[s]$  for  $0 \leq s \leq T_0$ , 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

$\text{opt}(x, x') = 'x \leftarrow x'; \text{ while } \text{ITM}(x - 1) > 0 \text{ set } x \leftarrow x - 1'$

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  
 $\text{opt}(x, x')$

- 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  $\infty$  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  $i \leftarrow \text{FORCE}[f]$ . Repeat step B2 if  $\text{POS}(i) \geq \text{ACTIVE}$ ; otherwise set  $y_s \leftarrow t$  and go to B6.
- B3.** [Choose  $i$ .] Set  $\theta \leftarrow \infty$ . For  $0 \leq k < \text{ACTIVE}$ , do the following steps if  $\text{ITEM}[k] < \text{SECOND}$ : If  $\text{SIZE}(\text{ITEM}[k]) = 1$ , set  $\text{FORCE}[f] \leftarrow \text{ITEM}[k]$ ,  $f \leftarrow f + 1$ ; otherwise set  $\lambda \leftarrow h(\text{ITEM}[k])$ , where  $h$  is the given heuristic function; if  $\lambda = \theta$  and  $\text{ITEM}[k] < i$ , set  $i \leftarrow \text{ITEM}[k]$ ; if  $\lambda < \theta$ , set  $i \leftarrow \text{ITEM}[k]$ ,  $\theta \leftarrow \lambda$ .
- B4.** [Forced?] If  $f > 0$ , set  $f \leftarrow f - 1$ ,  $i \leftarrow \text{FORCE}[f]$ ,  $y_s \leftarrow t$ , 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  $t + \text{ACTIVE}$  exceeds the maximum available TRAIL size. Otherwise set  $\text{TRAIL}[t + k] \leftarrow (\text{ITEM}[k], \text{SIZE}(\text{ITEM}[k]))$  for  $0 \leq k < \text{ACTIVE}$ ; then set  $y_s \leftarrow t$  and  $t \leftarrow t + \text{ACTIVE}$ .
- B6.** [Try  $i$ 's first option.] Set  $d_i \leftarrow \text{SIZE}(i)$ ,  $x_i \leftarrow \text{SET}[i]$ , and do  $\text{opt}(x, x_i)$ . (We'll try to extend the current partial solution by including the option that starts at  $x$ .)
- B7.** [Deactivate  $\text{ITM}(x)$ .] Set  $i \leftarrow \text{ITM}(x)$ ,  $i' \leftarrow \text{LOC}(x)$ ,  $k \leftarrow \text{POS}(i)$ . If  $k \geq \text{ACTIVE}$ , go to B8. Otherwise do step B11; after finishing B11, set  $\text{ACTIVE} \leftarrow \text{ACTIVE} - 1$ ,  $i'' \leftarrow \text{ITEM}[\text{ACTIVE}]$ ,  $\text{ITEM}[\text{ACTIVE}] \leftarrow i$ ,  $\text{ITEM}[k] \leftarrow i''$ ,  $\text{POS}(i) \leftarrow \text{ACTIVE}$ ,  $\text{POS}(i'') \leftarrow k$ .
- B8.** [Advance  $x$ .] Set  $x \leftarrow x + 1$ . Return to B7 if  $\text{ITM}(x) > 0$ .
- B9.** [Enter new stage.] Set  $s \leftarrow s + 1$ .
- B10.** [Enter new level.] Set  $l \leftarrow l + 1$  and  $\text{LS}[s] \leftarrow l$ . Terminate with level overflow if  $l > T$  (there's no room to store  $x_l$ ); otherwise return to B2.
- B11.** [Hide incompatible options.] For  $j \leftarrow i + \text{SIZE}(i) - 1$  down to  $i$ , do the following if  $j \neq i'$ : Set  $x' \leftarrow \text{SET}[j]$ , and do step B12 if  $\text{CLR}(x) = 0$  or  $\text{CLR}(x') \neq \text{CLR}(x)$ .
- B12.** [Visit siblings of  $x'$ .] Do  $\text{opt}(x'', x')$ . Then while  $\text{ITM}(x'') > 0$ , do step B13 unless  $x'' = x'$ , and set  $x'' \leftarrow x'' + 1$ .
- B13.** [Hide option  $x''$ .] Set  $i'' \leftarrow \text{ITM}(x'')$  and  $j'' \leftarrow \text{LOC}(x'')$ . If  $j'' \geq \text{SECOND}$  and  $\text{POS}(i'') \geq \text{ACTIVE}$ , do nothing (item  $i''$  has already been purified). Otherwise set  $s' \leftarrow \text{SIZE}(i'') - 1$ . If  $s' = 0$  and  $j'' < \text{SECOND}$ , set  $f \leftarrow 0$  and go to B16 (the active primary item  $i''$  has no option beside  $x''$ ). Otherwise if  $s' = 1$  and  $j'' < \text{SECOND}$ , set  $\text{FORCE}[f] \leftarrow i''$  and  $f \leftarrow f + 1$ . If  $s' > 0$ , set  $x''' \leftarrow \text{SET}[i'' + s']$ ,  $\text{SIZE}(i'') \leftarrow s'$ ,  $\text{SET}[i'' + s'] \leftarrow x''$ ,  $\text{SET}[j''] \leftarrow x'''$ ,  $\text{LOC}(x'') \leftarrow i'' + s'$ ,  $\text{LOC}(x''') \leftarrow j''$ .
- B14.** [Visit a solution.] Visit the solution that's specified by nodes  $x_{\text{LS}[j]}$  for  $0 \leq j < s$ .
- B15.** [Back up.] Terminate if  $s = 0$ . Otherwise set  $t \leftarrow y_s$ ,  $s \leftarrow s - 1$ ,  $l \leftarrow \text{LS}[s]$ .
- B16.** [Purge  $x_i$ .] If  $d_i = 1$ , go to B15. Otherwise, for  $y_s \leq k < t$ , set  $\text{SIZE}(i') \leftarrow s'$  if  $\text{TRAIL}[k] = (i', s')$ . Then set  $\text{ACTIVE} \leftarrow t - y_s$ ,  $t \leftarrow y_s$ ,  $x'' \leftarrow x_i$ , and do step B13. Also set  $x' \leftarrow x_i$ , and do step B12. (Step B13 won't find  $s' = 0$ , because every active primary item has at least two active options when  $d_i > 1$ .) Return to B10. ■

secondary item  
heuristic function  
siblings  
subroutine  
level  
stage

(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  $x_0 \dots x_l$  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  $x_{\text{LS}[0]}$ ,  $x_{\text{LS}[1]}$ , ...) is actually needed.

**323.** Step C4<sup>+</sup> 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 G $\mu$  with Problem H, compared to 445.7 G $\mu$  with  $h(i) = \text{SIZE}(i)$  and 407.4 G $\mu$  with Algorithm C<sup>+</sup>.)

**324.** In step B1, provide space for  $\text{WT}(i)$  (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 " $\text{WT}(i) \leftarrow \text{WT}(i) + 1.0$ " will do nothing when  $\text{WT}(i) = 2^{24}$ .)

In step B13, set  $\text{WT}(i'') \leftarrow \text{WT}(i'') + 1.0$  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  $\{r_i \mid 1 \leq i \leq n\}$ ,  $\{c_j \mid 1 \leq j \leq n\}$  listed in organ-pipe order, together with secondary items  $\{a_s \mid 1 < s \leq 2n\}$ ,  $\{b_d \mid -n < d < n\}$ . Let's add primary items  $\{N_k \mid 0 \leq k < p\}$  for the knights, together with secondary items  $\{R_k \mid 0 \leq k < p\}$ ,  $\{C_k \mid 0 \leq k < p\}$ , whose colors will be row and column indices. Finally,  $n^2$  secondary items  $\{ij \mid 1 \leq i, j \leq n\}$  will keep the queens and knights separate. The queen options are ' $r_i c_j a_{i+j} b_{i-j} ij$ ' for  $1 \leq i, j \leq n$ ; the knight options are ' $N_k ij R_k : i C_k : j R_{k'} : i' C_{k'} : j'$ ', for all  $1 \leq i, j, i', j' \leq n$  and  $0 \leq k < p$  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  $d_0 \dots d_7 = 85321111$ . 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 kilomeems 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 M $\mu$  have elapsed; and  $(N_0, N_1, N_2, N_3, N_4)$  have acquired weights (3448, 4019, 4839, 4859, 2504). Since  $d_4 d_5 d_6 d_7 = 1111$ , 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 M $\mu$  and increases the knight-item weights. At the end (759 M $\mu$ ) 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 M $\mu$  : 819 M $\mu$ ; (b) 13.3 G $\mu$  : 118.8 G $\mu$ ; (c) 11.6 G $\mu$  : 13.6 T $\mu$ .

**328.** (Solution by P. Weigel.) Let there be primary items  $p_i$  for  $0 \leq i \leq n$ , representing pigeons, and secondary items  $h_j$  for  $0 \leq j < n$ , 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  $\text{WTD}^\dagger$  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  $\text{WTD}$  when  $n = 20$  is 6.8 gigamems, using  $\#$ , and 2.6 gigamems for  $\text{WTD}^\dagger$ , compared to 473 kilomems for  $\text{MRV}$ .

This XCC problem also turns out to be exponentially bad for  $\text{FRB}$  and  $\text{FRB}^\dagger$ .

**329.** In step B1, provide space for single-precision floating point numbers  $\text{TRY}(i)$  (initially 1.0) and  $\text{FR}(i)$  (initially 0.5) in the  $\text{SET}$  array, when  $i$  is primary. In step B6, also set  $i_0 \leftarrow i$  and  $\text{TRY}(i_0) \leftarrow \text{TRY}(i_0) + 1$ . Then at the end of step B8, set  $\text{FR}(i_0) \leftarrow \text{FR}(i_0) - \text{FR}(i_0)/\text{TRY}(i_0)$ . In step B13, set  $\text{FR}(i_0) \leftarrow \text{FR}(i_0) + (1.0 - \text{FR}(i_0))/\text{TRY}(i_0)$  before going to B16. (See answer 324.)

**330.** (a)  $1.1 \text{ G}\mu : 819 \text{ M}\mu$ ; (b)  $207.9 \text{ G}\mu : 118.8 \text{ G}\mu$ ; (c)  $17.1 \text{ T}\mu : 13.6 \text{ T}\mu$ .

**331.** Any heuristic function  $h$  that can be used in step B3 can also be used in step  $\text{C2}^+$ , provided that we replace 'if  $\lambda = 1$ ' in that step by 'if  $\lambda = 0$ , set  $f \leftarrow 0$  and go to  $\text{C10}^+$ ; 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  $\text{C4}^+$  with  $\text{FLAG} \leftarrow -1$ , and (118) will set  $\text{SIZE}(i') \leftarrow 0$ . This size will be trailed in step  $\text{C5}^+$ , and we'll find  $\lambda = 0$  after trying option  $o$  for item  $i$ .)

To implement the  $\text{WTD}$  heuristic (see exercise 324), increase  $\text{WT}(i')$  before setting  $\text{FLAG} \leftarrow 1$  in (118), and increase  $\text{WT}(\text{ITEM}[i])$  before going to  $\text{C10}^+$  from step  $\text{C2}^+$ . Similarly, to implement  $\text{FRB}$  (see exercise 329), update the failure rate of  $i = \text{ITEM}[k]$  by setting  $\text{TRY}(i) \leftarrow \text{TRY}(i) + 1$  in step  $\text{C4}^+$ ; set  $\text{FR}(i) \leftarrow \text{FR}(i) + (1.0 - \text{FR}(i))/\text{TRY}(i)$  in step  $\text{C7}^+$  before going to  $\text{C11}^+$ , and  $\text{FR}(i) \leftarrow \text{FR}(i) - \text{FR}(i)/\text{TRY}(i)$  in step  $\text{C8}^+$ . (Sample implementations are in the online programs  $\text{SSXCC-WTD0}$  and  $\text{SSXCC-FRB0}$ .)

**332.** Yes! Call them  $\text{WTD}^\dagger$  and  $\text{FRB}^\dagger$  as in (131). Then  $\text{WTD}^\dagger$  improves Problem K, achieving  $2.5 \text{ G}\mu$ ;  $\text{FRB}^\dagger$  improves Problems  $\text{O}^*$ , U,  $\text{Y}^*$ , achieving  $(5844.4, 117.5, 2.5) \text{ G}\mu$ .

**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  $\text{MATCH}(i)$ , for every *secondary* item  $i$ . Let  $\text{STAMP}$  be a 32-bit integer, initially 0; all the  $\text{MARK}$  fields are also initially 0. To get ready for testing option  $o$ , do this, assuming that  $\text{NODE}[o]$  is the spacer preceding option  $o$ : Set  $\text{STAMP} \leftarrow (\text{STAMP} + 1) \bmod 2^{32}$ . If  $\text{STAMP} = 0$ , set  $\text{STAMP} \leftarrow 1$  and  $\text{MARK}(i) \leftarrow 0$  for all  $i$ . For  $x = o + 1, o + 2, \dots$ , set  $i \leftarrow \text{ITM}(x)$ , and exit the loop if  $i \leq 0$ ; otherwise set  $\text{MARK}(i) \leftarrow \text{STAMP}$ , and if  $i \geq \text{SECOND}$  also set  $\text{MATCH}(i) \leftarrow (\text{CLR}(x) = 0? -1: \text{CLR}(x))$ .

Now, given an item  $i \notin o$  (equivalently,  $\text{MARK}(i) \neq \text{STAMP}$ ), do this: For  $j \leftarrow i, i + 1, \dots$ , exit the loop unsuccessfully if  $j = i + \text{SIZE}(i)$ ; otherwise set  $o' \leftarrow \text{SET}[j]$  and exit the loop successfully if  $o'$  does not fail the following compatibility test: "Set  $x' \leftarrow o'$  and  $x \leftarrow x' + 1$ . While  $x \neq x'$ , set  $i' \leftarrow \text{ITM}(x)$ ; if  $i' < 0$ , set  $o' \leftarrow x \leftarrow x + i' - 1$ ;

parity  
 $d$ -way  
 $\text{WTD}^\dagger$   
 $\text{FRB}$   
floating point numbers  
hide  
 $\text{MRV}$   
 $\text{WTD}$  heuristic  
 $\text{FRB}$   
online programs  
 $\text{WTD}^\dagger$   
 $\text{FRB}^\dagger$   
secondary  
spacer

otherwise if  $\text{MARK}(i') = \text{STAMP}$  and  $(i' < \text{SECOND}$  or  $\text{CLR}(x) \neq \text{MATCH}(i'))$ , 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  $\{c_1, \dots, c_s\}$ .

**336.**  $O_{-1} = \{00, 05, 10, 13, 16, 19\}$ ;  $O_0^{\text{init}} = \{00, 05, 13, 19\}$ ;  $O_0 = \{00, 19\}$ ;  $\text{AGE}(10) = \text{AGE}(16) = -1$  (purged);  $\text{AGE}(13) = 0$  (removed);  $\text{AGE}(05) = 0$  (purged). (Soon afterwards, a solution will be found at stage 2, with  $O_1^{\text{init}} = O_1 = \{19\}$ ,  $O_2^{\text{init}} = \emptyset$ ,  $\text{AGE}(00) = 1$ ,  $\text{AGE}(19) = 3$  (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  $o = \text{'p q'}$ ,  $o' = \text{'p r'}$ ,  $o'' = \text{'q s'}$ ,  $S[o, r] = S[o'', r] = o'$ ,  $S[o, s] = S[o', s] = o''$ . If we choose  $o$ , blocking  $o'$  will trigger  $(o'', r)$ , thereby enqueueing  $o''$ . 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 **ACTIVE** =  $|I_{s+1}|$  and **OACTIVE** =  $|I_s|$ ; but **ACTIVE** = **OACTIVE** =  $|I_s|$  if  $o$  is being removed or purged.

- O1.** [Move to left spacer.] While  $\text{ITM}(o) > 0$ , set  $o \leftarrow o - 1$ . Then set  $x \leftarrow o + 1$ .
- O2.** [Hide  $o$  from  $\text{ITM}(x)$ .] Set  $i \leftarrow \text{ITM}(x)$ ,  $p \leftarrow \text{LOC}(x)$ . If  $p \geq \text{SECOND}$  and  $\text{POS}(i) \geq \text{OACTIVE}$ , go to O3 (item  $i$  has been purified); otherwise set  $s' \leftarrow \text{SIZE}(i) - 1$ . If  $s' = 0$  and  $p < \text{SECOND}$ , go to O11 ( $i$  is wiped out); otherwise set  $x' \leftarrow \text{SET}[i + s']$ ,  $\text{SIZE}(i) \leftarrow s'$ ,  $\text{SET}[i + s'] \leftarrow x$ ,  $\text{SET}[p] \leftarrow x'$ ,  $\text{LOC}(x) \leftarrow i + s'$ ,  $\text{LOC}(x') \leftarrow p$ .
- O3.** [Loop on  $x$ .] Set  $x \leftarrow x + 1$ . Return to O2 if  $\text{ITM}(x) > 0$ .
- O4.** [Begin trigger loop.] Set  $\text{AGE}(o) \leftarrow \text{A}$ ,  $p \leftarrow \text{TRIG}(o)$ ,  $\text{HEAD} \leftarrow 0$ .
- O5.** [Loop done?] If  $p = 0$ , go to O10. Otherwise set  $o' \leftarrow \text{INFO}(p)$ ,  $q \leftarrow \text{LINK}(p)$ ,  $i' \leftarrow \text{INFO}(q)$ ,  $p' \leftarrow \text{LINK}(q)$ .
- O6.** [Is  $o'$  active?] Set  $a \leftarrow \text{AGE}(o')$ . If  $a > \text{A}$ , go to O7 ( $o'$  is active); otherwise set  $i \leftarrow \text{ITM}(o' + 1)$ . If  $\text{LOC}(o' + 1) \geq i + \text{SIZE}(i)$ , go to O8 ( $o'$  is inactive).
- O7.** [Is  $i'$  active?] Go to O9 if  $\text{POS}(i') < \text{ACTIVE}$  ( $i'$  is active).
- O8.** [Keep trigger.] Set  $\text{LINK}(q) \leftarrow \text{HEAD}$ ,  $\text{HEAD} \leftarrow p$ ,  $p \leftarrow p'$ , and return to O5.
- O9.** [Trigger becomes fixit.] Set  $\text{INFO}(p) \leftarrow o$ ,  $\text{LINK}(q) \leftarrow \text{FIX}(o')$ . If  $\text{FIX}(o') = 0$ , put  $o' \Rightarrow Q$  and set  $\text{AGE}(o') \leftarrow \infty$ . Then set  $\text{FIX}(o') \leftarrow p$ ,  $p \leftarrow p'$ ; return to O5.
- O10.** [Success.] Set  $\text{TRIG}(o) \leftarrow \text{HEAD}$  and terminate successfully.
- O11.** [Clear the queue.] If  $\text{QF} = \text{QR}$ , 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:

$$\text{unfix}(o) = \begin{cases} \text{Set } p \leftarrow \text{FIX}(o) \text{ and } \text{FIX}(o) \leftarrow 0. \\ \text{While } p > 0, \text{ set } o' \leftarrow \text{INFO}(p), q \leftarrow \text{LINK}(p), \text{INFO}(p) \leftarrow o, \\ \quad p' \leftarrow \text{LINK}(q), \text{LINK}(q) \leftarrow \text{TRIG}(o'), \text{TRIG}(o') \leftarrow p, p \leftarrow p'. \end{cases}$$

**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  $o'$   
 $\text{AGE}(o')$   
author

Step O9 makes  $\text{AGE}(o')$  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  $\text{QF} = \text{QR}$ , terminate successfully. Otherwise  $Q \Rightarrow o$ .

**E2.** [Is  $o$  active?] If  $\text{AGE}(o) = \infty$ , go to E3 ( $o$  is active). Otherwise  $\text{unfix}(o)$ , 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  $p \leftarrow \text{FIX}(o)$ .

**E5.** [Loop done?] If  $p = 0$ , set  $\text{FIX}(o) \leftarrow 0$  and return to E1. Otherwise set  $q \leftarrow \text{LINK}(p)$ ,  $i \leftarrow \text{INFO}(q)$ ,  $p' \leftarrow \text{LINK}(q)$ . (We needn’t look at  $\text{INFO}(p)$  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  $\text{INFO}(p) \leftarrow o$ ,  $\text{LINK}(q) \leftarrow \text{TRIG}(o')$ ,  $\text{TRIG}(o') \leftarrow p$ ,  $p \leftarrow p'$ , and return to E5.

**E8.** [Prepare to purge.] (There’s no active support for  $(o, i)$ .) Set  $\text{FIX}(o) \leftarrow p$  and  $\text{unfix}(o)$  as in step O11.

**E9.** [Purge  $o$ .] Set  $\text{OACTIVE} \leftarrow \text{ACTIVE}$  and call  $\text{opt\_out}(o)$ . 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  $\text{QR} \leftarrow \text{AVAIL}$ ,  $\text{QF} \leftarrow \text{QR}$ ,  $\text{A} \leftarrow -1$ ,  $\text{OACTIVE} \leftarrow \text{ACTIVE}$ ,  $o \leftarrow 0$ .

**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  $i \leftarrow \text{ITEM}[k]$ .

**A4.** [Find a support.] If  $\text{MARK}(i) = \text{STAMP}$ , 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  $\text{opt\_out}(o)$ , and go to A7 if that succeeds. Otherwise terminate unsuccessfully.

**A5.** [Record the support.] Set  $p \leftarrow \text{AVAIL}$ ,  $q \leftarrow \text{AVAIL}$ ,  $\text{INFO}(p) \leftarrow o$ ,  $\text{LINK}(p) \leftarrow q$ ,  $\text{INFO}(q) \leftarrow i$ ,  $\text{LINK}(q) \leftarrow \text{TRIG}(o')$ ,  $\text{TRIG}(o') \leftarrow p$ .

**A6.** [Item loop done?] Set  $k \leftarrow k + 1$ ,  $i \leftarrow \text{ITEM}[k]$ . Return to A4 if  $i < \text{SECOND}$ .

**A7.** [Option loop done?] Set  $o \leftarrow o + \text{LOC}(o) + 1$ . Return to A2 if  $o < \text{LAST}$ .

**A8.** [Empty the queue.] Call  $\text{empty\_q}()$ , terminating unsuccessfully if it fails. ■

**342.** Do the actions in the following paragraph for all options  $o$  with  $\text{AGE}(o) \geq 0$ :

Set  $p \leftarrow \text{TRIG}(o)$  and  $q' \leftarrow -1$ . While  $p \neq 0$ , do this: “Set  $q \leftarrow \text{LINK}(p)$  and  $p' \leftarrow \text{LINK}(q)$ . If  $\text{AGE}(\text{INFO}(p)) \geq 0$ , simply set  $q' \leftarrow q$ ; otherwise put  $p \Rightarrow \text{AVAIL}$ ,  $q \Rightarrow \text{AVAIL}$ , and set  $\text{TRIG}(o) \leftarrow p'$  if  $q' < 0$ ,  $\text{LINK}(q') \leftarrow p'$  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  $\text{TRIG}(o)$  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  $\text{AGE}(o') < c$ . 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  $\text{SS}[c \gg 1]$ , 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  $H[a]$  and  $T[a]$  of temporary list heads and tails, for  $0 \leq a < 2T_0$ . Initially  $H[a] = 0$  for all  $a$ .

**O4<sup>+</sup>**. [Begin trigger loop.] Set  $\text{AGE}(o) \leftarrow A$ ,  $p \leftarrow \text{TRIG}(o)$ ,  $a_{\min} \leftarrow \infty$ ,  $p' \leftarrow 0$ .

**O5.1<sup>+</sup>**. [Hint?] If  $o' \geq 0$ , proceed to step **O6<sup>+</sup>** (this entry is not a hint).

**O5.2<sup>+</sup>**. [Valid hint?] If  $-o' \leq A$  and  $i' = \text{SS}[(-o') \gg 1]$ , set  $p' \leftarrow p$  and go to **O10<sup>+</sup>**.

**O5.3<sup>+</sup>**. [Discard a useless entry.]  $p \Rightarrow \text{AVAIL}$ ,  $q \Rightarrow \text{AVAIL}$ ,  $p \leftarrow p'$ , and return to **O5<sup>+</sup>**.

**O7<sup>+</sup>**. [Is  $i'$  active?] Go to **O9<sup>+</sup>** if  $\text{POS}(i') < \text{ACTIVE}$  ( $i'$  is active). Otherwise set  $a \leftarrow A$ .

**O8<sup>+</sup>**. [Keep trigger.] If  $a < 0$ , go to **O5.3<sup>+</sup>**. If  $a < a_{\min}$ , set  $a_{\min} \leftarrow a$ . If  $H[a] = 0$ , set  $T[a] \leftarrow q$ . Then set  $\text{LINK}(q) \leftarrow H[a]$ ,  $H[a] \leftarrow p$ ,  $p \leftarrow p'$ , and return to **O5<sup>+</sup>**.

**O10<sup>+</sup>**. [Bucket sort.] If  $p' \neq 0$  and  $a_{\min} = -o' - 1$ , set  $p' \leftarrow \text{LINK}(q)$ ,  $p \Rightarrow \text{AVAIL}$ ,  $q \Rightarrow \text{AVAIL}$  (avoid a double hint). For  $a = a_{\min}, a_{\min} + 1, \dots, A - 1$ , do this: “If  $H[a] \neq 0$ , set  $\text{LINK}(T[a]) \leftarrow p'$ ,  $p \Leftarrow \text{AVAIL}$ ,  $q \Leftarrow \text{AVAIL}$ ,  $\text{LINK}(p) \leftarrow q$ ,  $\text{INFO}(p) \leftarrow -a - 1$ ,  $\text{INFO}(q) \leftarrow \text{SS}[(a + 1) \gg 1]$ ,  $\text{LINK}(q) \leftarrow H[a]$ ,  $H[a] \leftarrow 0$ ,  $p' \leftarrow p$ .” Then if  $H[A] \neq 0$ , set  $\text{LINK}(T[A]) \leftarrow p'$ ,  $p' \leftarrow H[A]$ ,  $H[A] \leftarrow 0$ . Finally set  $\text{TRIG}(o) \leftarrow p'$  and terminate successfully. (See Algorithm 5.2.5R.) ■

**345.** With the hints,  $6.3 \text{ G}\mu$  (compared to  $1.3 \text{ G}\mu$  for Algorithm C); without them,  $124.1 \text{ G}\mu$ . (The ratio is even more extreme, about 1 to 150, when  $n = 15$ . In that case Algorithm S runs in  $2.1 \text{ T}\mu$ , compared to  $432 \text{ G}\mu$  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  $\text{SS}[k] \leftarrow k$  for  $0 \leq k < s$  and **SSTAMP**  $\leftarrow s$ . (The values in **SS** are distinct, and less than **SSTAMP**; so they can safely be used in future hints.)

**347.** Let  $\text{left}(o, x)$  mean “ $o \leftarrow x - 1$ ; while  $\text{ITM}(o) > 0$  set  $o \leftarrow o - 1$ .”

**J1.** [Begin loop.] Do  $\text{left}(o, x_i)$ , and set  $p \leftarrow \text{OACTIVE} \leftarrow \text{ACTIVE}$ ,  $x \leftarrow o + 1$ ,  $i \leftarrow \text{ITM}(x)$ .

**J2.** [Is  $i$  inactive?] Set  $p' \leftarrow \text{POS}(i)$ . If  $p' \geq p$ , go to **J4** ( $i$  has been purified).

**J3.** [Deactivate  $i$ .] Set  $p \leftarrow p - 1$ ,  $i' \leftarrow \text{ITEM}[p]$ ,  $\text{ITEM}[p] \leftarrow i$ ,  $\text{ITEM}[p'] \leftarrow i'$ ,  $\text{POS}(i) \leftarrow p$ ,  $\text{POS}(i') \leftarrow p'$ . If  $i \geq \text{SECOND}$ , set  $\text{MATCH}(i) \leftarrow \text{CLR}(x)$  (see answer 334).

**J4.** [Loop done?] Set  $x \leftarrow x + 1$  and  $i \leftarrow \text{ITM}(x)$ . Return to **J2** if  $i > 0$ .

**J5.** [Begin another loop.] Set  $\text{ACTIVE} \leftarrow p$ . (We'll block the options  $\neq o$  from the lists of all the newly inactive items,  $\{\text{ITEM}[k] \mid \text{ACTIVE} \leq k < \text{OACTIVE}\}$ .)

**J6.** [Block  $\text{ITEM}[p]$ 's options.] Set  $i \leftarrow \text{ITEM}[p]$  and  $j \leftarrow i + \text{SIZE}(i) - 1$ . If  $i \geq \text{SECOND}$  and  $\text{MATCH}(i) \neq 0$ , purify  $i$  as follows: “While  $j \geq i$ , set  $o' \leftarrow \text{SET}[j]$ ,  $j \leftarrow j - 1$ , and call  $\text{opt\_out}(o')$  if  $\text{CLR}(o') \neq \text{MATCH}(i)$ .” Otherwise block  $i$ 's options  $\neq o$  as follows: “While  $j \geq i$ , do  $\text{left}(o', \text{SET}[j])$ , set  $j \leftarrow j - 1$ , and call  $\text{opt\_out}(o')$  if  $o' \neq o$ .”

**J7.** [Loop done?] Set  $p \leftarrow p + 1$ . Return to **J6** if  $p < \text{OACTIVE}$ .

hint  
validation code  
stage stamp  
stamping  
 $T_0$   
Bucket sort  
sort  
radix list sort  
hints  
purified  
purify

**J8.** [Deactivate  $o$ .] Set  $\text{SIZE}(\text{ITEM}[p]) \leftarrow 0$  for all  $p$  with  $\text{ACTIVE} \leq p < \text{OACTIVE}$  and  $\text{ITEM}[p] < \text{SECOND}$ . Also set  $\text{AGE}(o) \leftarrow A$ . ■

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  $\text{opt\_out}(o')$  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  $ij \xrightarrow{Q} i'j'$  be the adjacency relation for the  $m \times n$  queen graph, namely " $ij \neq i'j'$  and  $(i = i' \text{ or } j = j' \text{ or } i + j = i' + j' \text{ or } i - j = i' - j')$ "; similarly, let  $ij \xrightarrow{N} i'j'$  denote adjacency in the corresponding knight graph, " $(i - i')^2 + (j - j')^2 = 5$ ." Let  $A$  be the set  $\{i'j' \mid i'j' \xrightarrow{Q} ij\}$  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  $p_1 \leq a_1$  and  $p_0 \leq a_0$ , because  $P \subseteq A$ .

Let there be  $mn$  primary items  $i'j'$  and secondary items  $x_{i'j'}$ , for  $0 \leq i' < m$ ,  $0 \leq j' < n$ ; also  $mn$  primary items  $\#k'$  and secondary items  $y_{k'}$ , for  $1 \leq k' \leq mn$ . 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  $i'j' \xrightarrow{N} i''j''$  and  $k'' = 1 + (k' \bmod mn)$  and  $OK(i', j', k')$  and  $OK(i'', j'', k'')$  are true, where  $OK(i', j', k')$  means " $0 \leq i' < m, 0 \leq j' < n, (i' + j' + k' + i + j) \bmod 2 = 1$ , 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  $i'j' \xrightarrow{N} i''j''$ ,  $i''j''$  is white,  $i''j'' \xrightarrow{N} i'''j'''$ ,  $i'j' \neq i'''j'''$ ,  $k'$  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)  $G\mu$ . (And the FC method FRB<sup>†</sup>\* takes 1475  $G\mu$ .) The solutions shown here minimize and maximize the sum of attacked labels.

73347504713669661796 31346308613691141758  
76057235020918976865 64073235900918599215  
33740308377067169598 33308962376013165798  
06778401101938996415 06658601101938991293  
83320724850011146194 29020588850011949756  
78258231202360391263 66872803202384394895  
81307922598613629340 27046722837847965540  
26532887562190434845 68737077242180435249  
29805154895849464192 71267582794651544144  
52278857505542914447 74697225768142455053

**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  $m \leq n$  are (6, 7, 2, 3): 2 · 52 solutions, 3.6  $G\mu$ ; (7, 8, 2, 2): 16 solutions, 9.6  $G\mu$ ; (7, 8, 2, 3): 1206 solutions, 597.1  $G\mu$ ; (7, 8, 3, 3): 2 · 989 solutions, 1450.7  $G\mu$ ; (7, 10, 2, 4): 491 solutions, 1338.4  $G\mu$ ; (8, 8, 3, 3): 2 · 688 solutions, 2154.8  $G\mu$ ; (8, 9, 3, 4): 2 · 6010 solutions, 33.9  $T\mu$ ; 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  $9 \times 10$  case has exactly  $2 \cdot 1658756$  solutions. He has also found many thousands of solutions to the  $10 \times 10$  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  
 $\text{opt}(x, x')$   
 $\text{deactivate}(i)$   
multiplicities  
forced

```

18 09 06 39 34 37 05 12 03 56 31 28 51 54 69 18 01 38 67 22 53 12 65 10 53 46 21 02 55 48 23 14 10 63 36 19 72 65 34 43 23 20 25 78 59 10 57 76 61 14
05 28 17 36 07 40 44 01 06 13 52 55 30 27 36 39 68 17 02 13 66 09 52 55 20 01 54 47 22 13 56 49 37 20 09 64 35 44 71 66 26 79 22 09 18 77 60 13 56 75
10 19 08 33 38 35 11 04 45 02 29 32 53 50 19 70 37 04 21 08 23 54 11 64 45 52 19 12 03 50 15 24 62 11 18 01 06 67 42 33 21 24 19 80 11 58 17 74 15 62
29 04 27 16 41 32 20 43 16 07 14 39 26 33 40 35 20 31 16 03 14 45 56 51 18 11 64 51 16 05 32 57 21 38 05 08 17 70 45 68 36 27 82 03 08 05 12 63 52 55
20 11 30 01 24 15 17 10 19 46 23 36 49 38 27 30 05 42 07 24 59 48 63 46 63 44 17 04 31 58 25 06 12 61 02 39 04 07 32 41 83 02 37 06 81 66 53 16 73 64
03 26 13 22 31 42 42 21 08 15 40 47 34 25 34 41 28 25 32 15 44 61 50 57 10 41 36 61 38 07 28 33 55 22 59 16 31 40 69 46 28 35 84 01 04 07 40 65 54 51
12 21 02 25 14 23 09 18 41 22 35 24 37 48 29 26 33 06 43 60 49 58 47 62 43 62 39 08 35 30 59 26 60 13 56 03 52 49 30 27 85 88 31 38 41 48 67 70 45 72
40 09 42 37 60 27 34 29 23 54 15 58 25 28 47 50 34 29 90 87 32 39 46 43 50 69 14 57 24 53 48 51 26 29 89 86 33 30 47 42 49 68 71 44

```

**352.** Besides  $\text{opt}(x, x')$ , the following adaptation of Algorithm B uses the subroutine

$\text{deactivate}(i) = \begin{cases} \text{ACTIVE} \leftarrow \text{ACTIVE} - 1, i''' \leftarrow \text{ITEM}[\text{ACTIVE}], k \leftarrow \text{POS}(i); \\ \text{ITEM}[\text{ACTIVE}] \leftarrow i, \text{ITEM}[k] \leftarrow i''', \text{POS}(i) \leftarrow \text{ACTIVE}, \text{POS}(i''') \leftarrow k. \end{cases}$

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  $\text{BOUND}(i)$  and  $\text{SLACK}(i)$  into the SET array for each primary item  $i$ , initialized to  $v_i$  and  $v_i - u_i$  when  $i$ 's given multiplicities are  $[u_i \dots v_i]$ . Terminate if  $\text{SIZE}(i) < u_i$  for any  $i$ . Deactivate any items for which  $\text{SIZE}(i) = 0$ .
- F2.** [Not forced?] If  $f = 0$ , go to F3. Otherwise set  $f \leftarrow f - 1$  and  $i \leftarrow \text{FORCE}[f]$ . Repeat step F2 if  $\text{POS}(i) \geq \text{ACTIVE}$ ; otherwise set  $y_s \leftarrow t$ ,  $\theta \leftarrow 1$ , and go to F6.
- F3.** [Choose  $i$ .] Set  $\theta \leftarrow \infty$ . For  $0 \leq k < \text{ACTIVE}$ , do the following steps if  $\text{ITEM}[k] < \text{SECOND}$ : Set  $i' \leftarrow \text{ITEM}[k]$ ,  $s \leftarrow \min(\text{SLACK}(i'), \text{BOUND}(i'))$ ,  $\lambda \leftarrow \text{SIZE}(i') + 1 + s - \text{BOUND}(i')$ . If  $\lambda = 1$ , set  $\text{FORCE}[f] \leftarrow i'$  and  $f \leftarrow f + 1$ , for  $1 \leq j \leq \text{SIZE}(i')$ . (In that case, every remaining option of  $i'$  is forced.) Otherwise, if  $\lambda \leq \theta$  and  $(\lambda < \theta$  or  $(s \leq \sigma$  and  $(s < \sigma$  or  $(\text{SIZE}(i') \geq \sigma'$  and  $(\text{SIZE}(i') > \sigma'$  or  $i' < i))))$ , set  $\theta \leftarrow \lambda$ ,  $i \leftarrow i'$ ,  $\sigma \leftarrow s$ , and  $\sigma' \leftarrow \text{SIZE}(i')$ . (See exercise 7.2.2.1–166.)
- F4.** [Forced?] If  $f > 0$ , set  $f \leftarrow f - 1$ ,  $i \leftarrow \text{FORCE}[f]$ ,  $\theta \leftarrow 1$ ,  $y_s \leftarrow t$ , 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  $t + \text{ACTIVE}$  exceeds the maximum available TRAIL size. Otherwise set  $\text{TRAIL}[t + k] \leftarrow (\text{ITEM}[k], \text{SIZE}(\text{ITEM}[k]), \text{BOUND}(\text{ITEM}[k]))$  for  $0 \leq k < \text{ACTIVE}$ , omitting BOUND if  $\text{ITEM}[k] \geq \text{SECOND}$ . Then set  $y_s \leftarrow t$  and  $t \leftarrow t + \text{ACTIVE}$ .
- F6.** [Try  $i$ 's first option.] Set  $d_i \leftarrow \theta$ ,  $x_i \leftarrow \text{SET}[i]$ , and do  $\text{opt}(x, x_i)$ . (We'll try to extend the current partial solution by including the option that starts at  $x$ .)
- F7.** [Commit  $\text{ITM}(x)$ .] Set  $i \leftarrow \text{ITM}(x)$ ,  $i' \leftarrow \text{LOC}(x)$ ,  $k \leftarrow \text{POS}(i)$ . If  $k \geq \text{ACTIVE}$ , go to F10 (the secondary item  $i$  has been purified). Otherwise, if  $i < \text{SECOND}$ , set  $\text{BOUND}(i) \leftarrow \text{BOUND}(i) - 1$ , and go to F9 if  $\text{BOUND}(i) > 0$ .
- F8.** [Cover  $i$ .] (Now  $i \geq \text{SECOND}$  or  $\text{BOUND}(i) = 0$ .) Do step F13. Then  $\text{deactivate}(i)$  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  $\text{ITM}(x) > 0$ .
- F11.** [Enter new stage.] Set  $s \leftarrow s + 1$ .

- F12.** [Enter new level.] Set  $l \leftarrow l + 1$  and  $LS[s] \leftarrow l$ . Terminate with level overflow if  $l > T$  (there's no room to store  $x_l$ ); otherwise return to F2.
- F13.** [Hide incompatible options.] For  $j \leftarrow i + \text{SIZE}(i) - 1$  down to  $i$ , do the following if  $j \neq i'$ : Set  $x' \leftarrow \text{SET}[j]$ , and do step F14 if  $\text{CLR}(x) = 0$  or  $\text{CLR}(x') \neq \text{CLR}(x)$ .
- F14.** [Visit siblings of  $x'$ .] Do  $\text{opt}(x'', x')$ . Then while  $\text{ITM}(x'') > 0$ , do step F15, and set  $x'' \leftarrow x'' + 1$ .
- F15.** [Hide option  $x''$ .] Set  $i'' \leftarrow \text{ITM}(x'')$  and  $j'' \leftarrow \text{LOC}(x'')$ . If  $j'' \geq \text{SECOND}$  and  $\text{POS}(i'') \geq \text{ACTIVE}$ , do nothing (item  $i''$  has already been purified). Otherwise set  $s' \leftarrow \text{SIZE}(i'') - 1$ . If  $j'' < \text{SECOND}$  and  $s' < \text{BOUND}(i'') - \text{SLACK}(i'')$ , set  $f \leftarrow 0$  and go to F18 (the active primary item  $i''$  needs  $x''$  to attain its lower bound). Otherwise, if  $j'' < \text{SECOND}$  and  $s' = 0$ , deactivate( $i''$ ) (which has lost its last option). Otherwise, if  $s' > 0$ , set  $x''' \leftarrow \text{SET}[i'' + s']$ ,  $\text{SIZE}(i'') \leftarrow s'$ ,  $\text{SET}[i'' + s'] \leftarrow x''$ ,  $\text{SET}[j''] \leftarrow x'''$ ,  $\text{LOC}(x'') \leftarrow i'' + s'$ ,  $\text{LOC}(x''') \leftarrow j''$ .
- F16.** [Visit a solution.] Visit the solution that's specified by nodes  $x_{LS[j]}$  for  $0 \leq j < s$ .
- F17.** [Back up.] Terminate if  $s = 0$ . Otherwise set  $t \leftarrow y_s$ ,  $s \leftarrow s - 1$ ,  $l \leftarrow LS[s]$ .
- F18.** [Purge  $x_l$ .] If  $d_l = 1$ , go to F17. Otherwise, for  $y_s \leq k < t$ , set  $\text{SIZE}(i') \leftarrow s'$  if  $\text{TRAIL}[k] = (i', s', b')$ ; also set  $\text{BOUND}(i') \leftarrow b'$  if  $i' < \text{SECOND}$ . Then set  $\text{ACTIVE} \leftarrow t - y_s$ ,  $t \leftarrow y_s$ ,  $x' \leftarrow x_l$ ,  $d_l \leftarrow 1$ , 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  $\theta$  is an integer and  $\theta'$  is a floating point number. Later in that step, if  $\lambda > 1$ , set  $\lambda' \leftarrow \lambda/w$ , where  $w = \text{WT}(i')$  for WTD,  $w = \text{FR}(i')$  for FRB. Then if  $\lambda' \leq \theta'$  and  $(\lambda' < \theta'$  or  $(s \leq \sigma$  and  $(s < \sigma$  or  $(\text{SIZE}(i') \geq \sigma'$  and  $(\text{SIZE}(i') > \sigma'$  or  $i' < i))$ )), set  $\theta' \leftarrow \lambda'$ ,  $\theta \leftarrow \lambda$ ,  $i \leftarrow i'$ ,  $\sigma \leftarrow s$ , and  $\sigma' \leftarrow \text{SIZE}(i')$ .

**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  $7 \times 7$  square, with  $u_{xy} = 1$  and  $v_{xy} = 1 + [x=0 \text{ or } y=0 \text{ or } x=6 \text{ or } y=6]$  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	PYPY	P	PYYZY	ZVY	VWVW
VX	X	X	W	Y	P	OP	P	P	ZY
V	X	W	W	R	P	O	P	Z	ZX
T	W	W	R	R	R	O	Q	Q	XX
T	T	T	R	Z	Z	O	Q	U	UX
QT	S	S	S	U	Z	OU	Q	U	RR
QS	QS	Q	Q	U	UZ	UZ	Q	U	UR
									TT

(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  $\dots$  or  $x_k \neq a_k$ ', using a new primary item  $\#$  whose multiplicity is  $[0..k-1]$ .

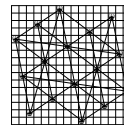
**357.** (This is essentially an application of the idea in the previous exercise.) If  $0 < p < q$  and  $p \perp q$ , the relevant lines ' $y = (p/q)x + \text{constant}$ ' of slope  $p/q$  can be enumerated by considering the lower left points at which they intersect an  $n \times n$  grid, namely  $\{(x, y) \mid 0 \leq x < n - 2q, 0 \leq y < n - 2p\} \setminus \{(x, y) \mid x \geq q, y \geq p\}$ . 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  $(N(4), \dots, N(12)) = (0, 12, 32, 76, 136, 252, 356, 572, 836)$ .)

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  $0 \leq k < N(n)$ , 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  $\gamma_t$  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  $\geq 3$  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  $0 \leq i < m$ ,  $0 \leq j < n$ , and  $0 \leq d < m + n - 1$ . There are two options for every cell  $ij$  of the board, one for queen placement and one for knight placement, namely

$$\begin{aligned} & 'Q r_i c_j a_{i+j} b_{i+n-1-j} \cup \{s_{kl}:0 \mid ij \text{ --- } kl \text{ in the } m \times n \text{ knight graph}\}', \\ & 'S s_{ij} r_i:0 c_j:0 a_{i+j}:0 b_{i+n-1-j}:0 \cup \{s_{kl}:0 \mid ij \text{ --- } kl \text{ in the } m \times n \text{ knight graph}\}'. \end{aligned}$$

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

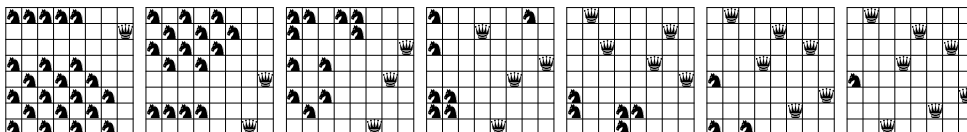
knight graph

Weigel

blank (unmatchable) color

unmatchable

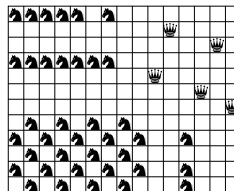
**361.** A bevy of  $q = 1, \dots, 7$  queens can coexist respectively with  $(22, 15, 11, 7, 5, 3, 1)$  knights in respectively  $(12, 40, 56, 328, 16, 40, 104)$  ways, ignoring symmetry; and no additional knights can be added. For example, here's a maximum solution for each case:



WTD heuristic  
FRB heuristic  
tilted square  
Historical notes  
ITA Software  
Hui  
MCC problem  
unique  
auxiliary variables

The dancing links technology of Algorithm 7.2.2.1M handles each of these 14 problems in fewer than 301  $M\mu$ . Algorithm F is faster yet — at most 183  $M\mu$ , when  $(q, s) = (2, 15)$ . But the WTD heuristic makes it worse, when  $q$  is small: More than 1  $G\mu$  when  $q = 1$ . And the FRB heuristic is champion in all cases with  $q < 5$ ; for instance, it needs only 33  $M\mu$  when  $(q, s) = (3, 12)$ , compared to 186  $M\mu$  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  $T\mu$  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.jssoftware.com/wiki/Essays/Queens\\_and\\_Knights](http://code.jssoftware.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 ‘ $\#06 051:0 061:1 071:0 161:0$ ’; ‘ $\#03 \#13 022:0 032:1 042:0 122:0 132:1 142:0 232:0$ ’; ‘ $\#02 \#11 \#12 013:0 023:1 033:0 103:0 113:1 123:1 133:0 213:0 223:0$ ’; ‘ $\#00 \#01 \#10 \#20 004:1 014:1 024:0 104:1 114:0 204:1 214:0 304:0$ ’.

It turns out that Tullis's tapestry is *unique*: There's only one solution, except for rotation and reflection(!).

4432441332  
4332414432  
4243433421  
1243343424  
2344142334  
2331442344  
4423324413  
3424324143  
3414234243  
3144233244

**363.** (a) There exists  $y = y_1 y_2 y_3$  such that  $S(x, y)$  and  $S(y, x)$  both hold.

(b) There exists  $z = z_1 z_2 z_3$  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  $x^{RRRL}$ , which is the  $k$ -tuple  $x_1^R x_2^R \dots x_{k-3}^R x_k^{RRRL}$  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  $x^{(i)} = x^{L^{i-1} R^{k-1}} = x_1^{L^{i-1} R^{k-1}} \dots x_i^{L^{i-1} R^{k-1}}$ . Also let  $y^{[j]} = y^{R^{k-j} L}$ , a  $k$ -tuple that ends with  $\dots y_j y_k^{R^{k-j} L}$ . Therefore  $S(x^{(i)}, y^{[j]})$  enforces  $x_i = y_j$ , and no other constraints between  $x$  and  $y$ .

(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  $s_1 \dots s_m$  tuples, where  $k = k_1 + \dots + k_m$ . 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  $\{\text{BANBECOD}, \text{BANBECON}, \dots, \text{SINSOLED}\}$ ; the domain of  $\mathcal{P}^*$  is  $D^8$ , where  $D = \{A, B, C, D, E, I, L, N, O, S, U\}$ ; and  $v$  is a variable that we might call ' $x_1 x_3 x_5 x_1 x_4 x_2 x_4 x_5$ '. 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  $R_\alpha$ , for one or more  $\alpha$ , is efficiently decidable.)

(b) In other words, we want to show that the CSP with  $3t + 1$  variables and all domains  $[0 \dots 3t]$ , which asks for a mapping  $u \mapsto u'$  such that  $R_{it}(u, v)$  implies  $R_{jt}(u', v')$ , is unsatisfiable. The path  $3i \rightarrow 3i + 1 \rightarrow 3i + 2 \rightarrow 3i + 3$  in  $R_{it}$  must map into the path  $3j \rightarrow 3j + 1 \rightarrow 3j + 2 \rightarrow 3j + 3$  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  $x \leq 3i$  be maximum such that  $x' = x - 3(j - i)$ . Then we have  $y' = x' + 1$  when  $y = x - 1$ , because  $y' \neq x' - 1$ . 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  $[0 \dots 3t]$ . Enforcing domain consistency leads to the unique solution  $u' = u - 2[u \geq 3i + 3]$ .

[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  $b \neq b'$ ' (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  $\cap$  two-fan = two-fan; correspondence  $\cap$  two-fan = correspondence; two-fan  $\cap$  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  $(v, a) \rightarrow (w, b')$  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  $\hat{G}$ . 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  $S(p) = \{\text{INFO}(p)\} \cup S(\text{LINK}(p))$ , where  $\text{INFO}(p) < \text{INFO}(\text{LINK}(p))$ .

We use a subroutine ‘align( $p, q$ )’, which finds the smallest element where  $S(p)$  and  $S(q)$  agree, or  $\infty$  if they have no common element: While  $\text{INFO}(p) \neq \text{INFO}(q)$ , set  $p \leftarrow \text{LINK}(p)$  if  $\text{INFO}(p) < \text{INFO}(q)$ , otherwise set  $q \leftarrow \text{LINK}(q)$ .

To compute the intersection we use  $m + 1$  pointer variables, initializing them by setting  $p \leftarrow Y$  and  $p_j \leftarrow \text{LINK}(X_j)$  for  $1 \leq j \leq m$ . Cyclically align( $p_{1+(j-1) \bmod m}, p_{1+(j \bmod m)}$ ) for  $j = 1, 2, \dots$ , until  $m$  consecutive alignments cause no further change. (At this point  $\text{INFO}(p_j)$  is constant for  $1 \leq j \leq m$ .) If  $\text{INFO}(p_1) < \infty$ , set  $\text{LINK}(p) \leftarrow \text{AVAIL}$ ,  $p \leftarrow \text{LINK}(p)$ ,  $\text{INFO}(p) \leftarrow \text{INFO}(p_1)$ ,  $p_j \leftarrow \text{LINK}(p_j)$  for  $1 \leq j \leq m$ , and resume the cyclic alignment loop. Otherwise set  $\text{LINK}(p) \leftarrow \perp$  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.**  $F(x_1, \dots, x_t) = [f(x_1, \dots, x_t) \geq \alpha]$ ,  $G(x) = [g(x) > \beta]$  are monotone functions.

**375.** Theorem 7.1.1H says that conjunctions of Horn clauses are equivalent to *min*-closed relations on  $\{0, 1\}$  if  $0 = \text{false}$  and  $1 = \text{true}$ . So they’re equivalent to *max*-closed relations on  $\{0, 1\}$  if we stand on our heads and let  $0 = \text{true}$  and  $1 = \text{false}$ .

[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 ‘ $(x_1 \geq a_1)$  or  $\dots$  or  $(x_t \geq a_t)$  or  $(x \leq b)$ ’, a special case of (142).]

**376.** If constraint  $c$  involves the variables  $v_1, \dots, v_k$ , we know that it contains tuples  $a_{11}a_{12} \dots a_{1k}, \dots, a_{k1}a_{k2} \dots a_{kk}$  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  $a_{11}a_{22} \dots a_{kk}$ .

**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 *V*-closed. All of the unary relations of  $\mathcal{P}$  are *V*-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  $a_1 a_2 a_3$  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  $c = a_1 a_2 a_3$ . This binary relation is  $\vee$ -closed; for if  $(a_2, a_1 a_2 a_3)$  and  $(a'_2, a'_1 a'_2 a'_3)$  both satisfy it, so does  $(a_2 \vee a'_2, a_1 a_2 a_3 \vee a'_1 a'_2 a'_3) = (\max(a_2, a'_2), \max(a_1, a'_1) \max(a_2, a'_2) \max(a_3, a'_3))$ .

**380.** For each  $a_1$  with  $0 \leq a_1 < d_1$ , extend  $a_1 \dots a_k$  to  $a_1 \dots a_k a_{k+1}$  for each  $a_{k+1}$  with  $\max\{l_1, \dots, l_k\} \leq a_{k+1} \leq \min\{r_1, \dots, r_k\}$ .

**381.** If, say,  $a_1 = a_2$ , so that  $\langle a_1 a_2 a_3 \rangle = a_1$ , 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  $R_{uvw} = R_{uv} \bowtie R_{vw}$ ; 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”  $0 \times 0$  matrix. The set  $\mathcal{A}^+$  contains all-zero matrices if and only if  $A_{00} = 1$ .

(The bivariate exponential generating functions  $A(w, z) = \sum_{m,n \geq 0} A_{mn} \frac{w^m}{m!} \frac{z^n}{n!}$ ,  $A^+(w, z) = \sum_{m,n \geq 0} A_{mn}^+ \frac{w^m}{m!} \frac{z^n}{n!}$  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  $m \geq 2$  and  $n \geq 2$ ;  $P_{m1} = P_{1n} = 1$  when  $mn > 0$ ; and  $P_{m0} = [m=0]$ ,  $P_{0n} = [n=0]$ . Exercise 384 now gives

$P_{mn}$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$P_{mn}^+$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$m=1$	1	1	1	1	1	1	$m=1$	2	4	8	16	32	64
$m=2$	1	7	7	9	11	13	$m=2$	4	16	46	114	264	592
$m=3$	1	7	16	13	16	19	$m=3$	8	46	155	418	1013	2326
$m=4$	1	9	13	41	21	25	$m=4$	16	114	418	1202	3046	7174
$m=5$	1	11	16	21	146	31	$m=5$	32	264	1013	3046	8107	19924
$m=6$	1	13	19	25	31	757	$m=6$	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  $p_1 \dots p_{m+n}$  of  $\{1, \dots, m+n\}$  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  $j - m \leq p_j \leq j + n$ , 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

$$Q_{mn} = \sum_{k \geq 0} k!^2 \left\{ \begin{matrix} m \\ k \end{matrix} \right\} \left\{ \begin{matrix} n \\ k \end{matrix} \right\}; \quad Q_{mn}^+ = \sum_{k \geq 0} k!^2 \left\{ \begin{matrix} m+1 \\ k+1 \end{matrix} \right\} \left\{ \begin{matrix} n+1 \\ k+1 \end{matrix} \right\}.$$

$Q_{mn}$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$Q_{mn}^+$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$m=1$	1	1	1	1	1	1	$m=1$	2	4	8	16	32	64
$m=2$	1	5	13	29	61	125	$m=2$	4	14	46	146	454	1394
$m=3$	1	13	73	301	1081	3613	$m=3$	8	46	230	1066	4718	20266
$m=4$	1	29	301	2069	11581	57749	$m=4$	16	146	1066	6902	41506	237686
$m=5$	1	61	1081	11581	95401	673261	$m=5$	32	454	4718	41506	329462	2441314
$m=6$	1	125	3613	57749	673261	6487445	$m=6$	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  $\frac{4}{6}$  to  $\frac{2}{6}$  to  $\frac{1}{6}$  to  $\frac{1}{5}$  to  $\frac{0}{5}$  to  $\frac{0}{5}$  to  $\frac{3}{5}$  to  $\dots$  to  $\frac{8}{8}$  to  $\frac{8}{8}$  to  $\frac{8}{8}$ .

Given the total number of rows,  $m$ , we shall set up simultaneous recurrence relations between variables of four types:  $X_j^i$ ,  $\bar{X}_j^i$ ,  $\underline{X}_j^i$ ,  $\bar{\underline{X}}_j^i$ . In all cases we have  $i \geq 0$ ,  $j \geq 0$ , and  $i + j < m$ . Furthermore  $i > 0$  in types  $X$  and  $\underline{X}$ ;  $j > 0$  in types  $X$  and  $\bar{X}$ . (Hence there are  $\binom{m-1}{2}$  variables of type  $X_j^i$ ,  $\binom{m}{2}$  variables of type  $\bar{X}_j^i$  and type  $\underline{X}_j^i$ ,  $\binom{m+1}{2}$  variables of type  $\bar{\underline{X}}_j^i$ .) Each variable has a parameter  $n$ ; for example,  $\bar{\underline{X}}_5^3(n)$  is the number of  $m \times n$  matrices whose final column has the annotations  $\bar{3}$  and  $\underline{5}$ .

The initial conditions are  $X_j^i(1) = 1$ ;  $\bar{X}_j^i(1) = [i = 0]$ ;  $\underline{X}_j^i(1) = [j = 0]$ ;  $\bar{\underline{X}}_j^i(1) = [i = j = 0]$ . The transition relations are

$$\begin{aligned} X_j^i(n+1) &= \sum \{X_{j'}^{i'}(n) \mid i' \geq i, j' \geq j\}; \\ \bar{X}_j^i(n+1) &= [i=0] \sum \{X_{j'}^{i'}(n) \mid i' \geq 1, j' \geq j\} + \sum \{\bar{X}_{j'}^{i'}(n) \mid i' \leq i, j \leq j' \leq m-i\}; \\ \underline{X}_j^i(n+1) &= [j=0] \sum \{X_{j'}^{i'}(n) \mid i' \geq i, j' \geq 1\} + \sum \{\underline{X}_{j'}^{i'}(n) \mid i \leq i' \leq m-j, j' \leq j\}; \\ \bar{\underline{X}}_j^i(n+1) &= [i=j=0] \sum \{X_{j'}^{i'}(n) \mid i' \geq 1, j' \geq 1\} + \sum \{\bar{\underline{X}}_{j'}^{i'}(n) \mid i' \leq i, j' \leq j\} + \\ &\quad [i=0] \sum \{\underline{X}_{j'}^{i'}(n) \mid 1 \leq i' \leq m-j, j' \leq j\} + [j=0] \sum \{\bar{\underline{X}}_{j'}^{i'}(n) \mid i' \leq i, 1 \leq j' \leq m-i\}; \end{aligned}$$

and  $R_{mn} = \sum_{i,j} \bar{\underline{X}}_j^i(n)$ . Variables are, of course, omitted from these formulas when they don’t exist. For example, some of the relations when  $m=3$  are  $X_1^1(n+1) = X_1^1(n)$ ;  $\underline{X}_0^1(n+1) = X_1^1(n) + \underline{X}_0^1(n) + \underline{X}_0^2(n)$ ;  $\bar{\underline{X}}_0^2(n+1) = \bar{\underline{X}}_1^0(n) + \bar{\underline{X}}_1^1(n) + \bar{\underline{X}}_0^0(n) + \bar{\underline{X}}_0^1(n) +$

bounded permutations  
Vesztergombi  
complete bipartite graph  
directed acyclic graphs  
acyclic orientations  
lonesum matrices  
row sums  
column sums  
parades  
author



$$\bar{X}_0^2(n); \bar{X}_0^0(n+1) = X_1^1(n) + \bar{X}_2^0(n) + \bar{X}_1^0(n) + \bar{X}_0^2(n) + \bar{X}_0^1(n) + \bar{X}_0^0(n).$$

$R_{mn}$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$R_{mn}^+$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$m=1$	1	1	1	1	1	1	$m=1$	2	4	8	16	32	64
$m=2$	1	7	17	31	49	71	$m=2$	4	16	56	176	512	1408
$m=3$	1	17	90	284	687	1411	$m=3$	8	56	289	1231	4623	15887
$m=4$	1	31	284	1398	4861	13555	$m=4$	16	176	1231	6655	30553	125197
$m=5$	1	49	687	4861	23020	83858	$m=5$	32	512	4623	30553	166186	790250
$m=6$	1	71	1411	13555	83858	386774	$m=6$	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  $\bar{\underline{c}}(w, z)$  be the generating function for the even more special case of parallominoes or “staircase polygons” (exercise 7.2.2.1–303). Then  $r(w, z) = c(w, z) + 2\underline{c}(w, z)^2/(1 - \bar{\underline{c}}(w, z))$ ; and

$$\bar{\underline{c}}(w, z) = (1 - w - z - \sqrt{\Delta})/2, \quad \text{where } \Delta = (w - z)^2 - 2w - 2z + 1;$$

$$\underline{c}(w, z) = wz/\sqrt{\Delta};$$

$$c(w, z) = 1 + wz((1-w)^3 + (1-z)^3 - 1 - wz((w-z)^2 + w + z - 5))/\Delta^2 - 4w^2 z^2/\Delta^{3/2}.$$

See Mireille Bousquet-Mélou, *Journal of Physics* **A25** (1992), 1925–1934, 1935–1944; also OEIS A001263, A008409, A324009. Incidentally,  $\bar{\underline{c}}(w, z) = (\underline{c}(w, z) + w)(\underline{c}(w, z) + z)$  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  $\tau$  and column  $n$  is annotated with  $\underline{0}$ . So  $X_j^i(n) = \bar{X}_j^i(n) = 0$ , and we need only the variables  $\bar{X}_j^i(n)$  and  $\bar{X}_0^i(m)$ ;  $S_{mn} = \sum_i \bar{X}_0^i(m)$ .

$S_{mn}$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$S_{mn}^+$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
$m=1$	1	1	1	1	1	1	$m=1$	2	4	8	16	32	64
$m=2$	1	4	8	13	19	26	$m=2$	4	13	38	104	272	688
$m=3$	1	8	29	73	151	276	$m=3$	8	38	147	506	1612	4856
$m=4$	1	13	73	266	749	1781	$m=4$	16	104	506	2103	7887	27477
$m=5$	1	19	151	749	2762	8321	$m=5$	32	272	1612	7887	34088	134825
$m=6$	1	26	276	1781	8321	31004	$m=6$	64	688	4856	27477	134825	597539

[The generating function  $\sum_{m,n} S_{mn} w^m z^n$  is  $1/(1 - \bar{\underline{c}}(w, z))$ , 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:

$$\begin{array}{ccccc} \begin{array}{c} 100 \\ 010(2), \\ 001 \end{array} & \begin{array}{c} 100 \\ 001(4); \\ 010 \end{array} & \begin{array}{c} 111 \\ 100(4), \\ 100 \end{array} & \begin{array}{c} 111 \\ 010(4), \\ 010 \end{array} & \begin{array}{c} 010 \\ 111(1); \\ 010 \end{array} & \begin{array}{c} 111 \\ 111(1). \end{array} \end{array}$$

(b) Now the only symmetry is transposition (interchanging rows and columns):

100	110	110	100	010	100	001	001	001	010	100
010(1);	010(2);	001(2);	001(1);	110(1);	011(2);	001(1);	101(2);	111(2);	110(2);	001(2);
001	001	001	011	001	001	111	011	001	011	111
100	100	100	100	100	100	110	001	001	001	010
010(2);	011(1);	100(2);	101(2);	110(2);	111(2);	110(1);	011(1);	101(2);	111(2);	110(2);
111	011	111	011	011	001	001	111	111	011	111
010	100	100	100	100	101	101	101	110	001	010
111(1);	011(2);	101(2);	110(2);	111(2);	001(1);	101(2);	111(2);	110(2);	111(2);	111(2);
011	111	111	111	011	111	011	001	011	111	111
100	101	101	101	110	111	011	101	110	111	111
111(2);	011(1);	101(2);	111(2);	111(1);	111(2);	111(1);	111(2);	111(2);	111(2);	111(1);
111	111	111	011	011	001	111	111	111	111	111

(c) Here, as in (a), we can exploit the eightfold symmetries of the square:

100	110	110	100	111	111	110	110	110	110
010 (2);	010 (8);	001 (4);	011 (4);	010 (4);	001 (4);	110 (4);	011 (8);	011 (4);	010 (4);
001	001	001	010	010	001	001	010	001	011
010	111	111	110	100	111	111	011	111	111
111 (1);	110 (4);	110 (8);	111 (4);	111 (8);	111 (8);	111 (4);	111 (2);	111 (4);	111 (1).
010	100	010	010	011	100	010	110	110	111

(d) And this case allows central symmetry, and reflection about either diagonal:

100	110	110	111	110	110	110	100	100	100	110	110	111
010 (1);	010 (4);	001 (2);	001 (2);	110 (2);	011 (2);	010 (2);	110 (2);	111 (4);	111 (4);	111 (1);	111 (2);	111 (1).
001	001	001	001	001	011	011	111	011	111	011	111	111

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

**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  $G_{lm}(z) = \sum_{n \geq 0} Q_{lmn}^+ z^n$  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  $\Delta$  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  $\Delta$  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  $|D| \geq 3$  is invariant under  $\Delta$  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  $(x \neq y? z : x)$ , which is called the "dual discriminator" because of its similarity to the discriminator in exercise 426.

**392.**  $\mu_d$  preserves every such relation. Conversely, if  $R(x_1, \dots, x_m)$  is preserved by  $\mu_d$ , so is  $\hat{R}(x_1, \dots, x_m) = R((x_1 - a_1) \bmod d, \dots, (x_m - a_m) \bmod d)$ , for any constants  $a_1, \dots, a_m$ . Therefore if  $R$  is nonempty, we can assume that  $\hat{R}(0, \dots, 0)$  is true. Then  $\hat{R}(x_1, \dots, x_m)$  and  $\hat{R}(y_1, \dots, y_m)$  imply  $\hat{R}((x_1 \pm y_1) \bmod d, \dots, (x_m \pm y_m) \bmod d)$ . So we can find linearly independent basis vectors  $\alpha_j = (\alpha_{j1}, \dots, \alpha_{jm})$  such that  $\hat{R}(x_1, \dots, x_m) \iff (x_1, \dots, x_m) \equiv t_1 \alpha_1 + \dots + t_k \alpha_k \pmod{d}$  for integers  $t_1, \dots, t_k$ . We can renumber the coordinates and convert the basis vectors to Smith normal form, so that  $\alpha_{ii}$  divides  $\alpha_{jj}$  and  $\alpha_{ij} = \alpha_{ji} = 0$  when  $1 \leq i < j \leq k$ ;  $\alpha_{kk}$  divides  $d$ ; and  $\alpha_{ii}$  divides  $\alpha_{ij}$  when  $1 \leq i \leq k < j \leq m$ . (See exercise 7-138.) Thus  $k < m$ ; and  $\hat{R}$  (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  $\tau_0 = (12, 20, 18, 14) \in \hat{R}$ . Dividing by 7 mod 36 (i.e., multiplying by 31) tells us that  $\tau_1 = 31\tau_0 = (12, 8, 18, 2) \in \hat{R}$ . Maybe also  $\tau_2 = (12, 12, 30, 6) \in \hat{R}$ ; hence  $\tau_3 = \tau_2 - 3\tau_1 = (12, 24, 12, 0) \in \hat{R}$ , and  $\tau_4 = \tau_1 - \tau_3 = (0, 20, 6, 2) \in \hat{R}$ .

Suppose now that all elements of  $\hat{R}$  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  $\hat{R}(x_1, x_2, x_3, x_4)$  is ' $18x_4 \equiv 0$  and  $3x_1 \equiv 0$  and  $x_2 \equiv 24t_1 + 20t_2 = 2x_1 + 10x_4$  and  $x_3 \equiv 12t_1 + 6t_2 = x_1 + 3x_4 \pmod{36}$ '. And the original relation  $R(x_1, x_2, x_3, x_4)$  is  $\hat{R}((x_1 + 3) \bmod 36, (x_2 + 1) \bmod 36, (x_3 + 4) \bmod 36, (x_4 + 1) \bmod 36)$ .

**393.** (a) There's a clause  $([a_1 \dots a_k] \vee [b_1 \dots b_k])$  whenever  $a_1 \vee b_1 = \dots = a_k \vee b_k = 1$ ,  $a_1 \dots a_k \neq 1 \dots 1 \neq b_1 \dots b_k$ , and  $a_1 \dots a_k < b_1 \dots b_k$ . 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  $k \leq 7$ ; but that method becomes infeasible for larger  $k$ . See OEIS A051185.]

(b) When 0 and 1 are interchanged, a function  $f(x_1, \dots, x_k)$  changes to its dual,  $f(\bar{x}_1, \dots, \bar{x}_k)$ . For example, (146) becomes  $([110] \vee [001]) \wedge \dots \wedge ([010] \vee [001])$ .

(c) When a Boolean polymorphism represents a family of sets,  $'[x_1 \dots x_k] = 1'$  means that the set  $S_x = \{j \mid x_j = 1\}$  is in the family. If  $[x_1 \dots x_k] = 1$  in the polymorphism for NAND, we have  $[y_1 \dots y_k] = 0$  for all sets  $S_y = \{j \mid y_j = 1\}$  that are disjoint from  $S_x$ , because of the clause  $([x_1 \dots x_k] \vee [y_1 \dots y_k])$ . 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 ( $'\perp'$ ), or the full set ( $'\top'$ ), or  $\{00, 11\}$  ( $'\equiv'$ ); so  $f_k(R) = 2^{2^k}$  in those cases. Every Boolean function with  $[1 \dots 1] = 1$  works when  $R = \{11\}$  (AND), or when  $R = \{10, 11\}$  ( $'\perp'$ ) or  $\{01, 11\}$  ( $'\bar{\vee}'$ ); so  $f_k(R) = 2^{2^k-1}$  in those cases (and dually also when  $R = \{00\}$  (NOR), or  $\{00, 01\}$  ( $'\bar{\wedge}'$ ), or  $\{00, 10\}$  ( $'\bar{\vee}'$ )). When  $R = \{01\}$  ( $'<'$ ) or  $\{10\}$  ( $'>'$ ), its polymorphisms are the operations with both  $[0 \dots 0] = 0$  and  $[1 \dots 1] = 1$ ; 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  $R = \{00, 01, 11\}$  ( $'\leq'$ ) and  $\{00, 10, 11\}$  ( $'\geq'$ ) are the monotone Boolean functions, with  $(f_1(R), \dots, f_5(R)) = (3, 6, 20, 168, 7581)$ .

(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  $([001] \vee [010] \vee [100]) \wedge \dots$ , 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 (from  $\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ ) is  $'[000][111] \in \{01, 11\}'$  (hence  $[000] \neq 2$  and  $[111] = 1$ ). The second of those constraints (from  $\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 2 & 0 \end{pmatrix}$ ) 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  $[000] \neq 2, \dots, [111] \neq 2$ ; 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

$$f(x, y, z) = (x = y? x : x = z? x : y = z? y : 0).$$

(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  
 $\perp$   
 $\top$   
AND  
 $\perp$   
NOR  
 $\perp$   
 $\bar{\vee}$   
 $\bar{\wedge}$   
XOR  
self-dual  
 $\bar{\vee}$   
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  $x_1 \dots x_k$  for all  $k$ -tuples with  $x_j \in D$ ; their “colors” will be the values of the variables  $[x_1 \dots x_k]$ . The primary items, for each  $R \in \Gamma$ , are  $Ri_1 \dots i_k$ , when  $R$  has  $t$   $m$ -tuples  $\tau_1 \dots \tau_t$  and  $1 \leq i_1, \dots, i_k \leq t$ ; they represent the matrices  $A$  in the definition of  $\mathcal{I}_k(\Gamma)$ .

The options for  $Ri_1 \dots i_k$  (see (150) and the online program INDICATOR-DLX) are as follows: Let  $a_{l1} \dots a_{lm} = \tau_{i_l}$  for  $1 \leq l \leq k$ , and  $\alpha_j = a_{1j} \dots a_{kj}$  for  $1 \leq j \leq m$ . (They are the rows and columns of  $A$ .) For each tuple  $b_1 \dots b_m \in R$ , include no option if  $\alpha_i = \alpha_j$  and  $b_i \neq b_j$ , for some  $1 \leq i < j \leq m$ . Otherwise include the option ‘ $Ri_1 \dots i_k \cup \bigcup \{\alpha_j : b_j \mid 1 \leq j \leq m \text{ and } \alpha_j \notin \{\alpha_1, \dots, \alpha_{j-1}\}\}$ ’.

*Note:* Some of the secondary items might not occur in any options. Then the corresponding variable  $[x_1 \dots x_k]$  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  $([\alpha 01], [\alpha 10], [\alpha 11]) = (0, 0, 1)$ , for any binary string  $\alpha$ . Thus  $f(x_1, \dots, x_k)$  is either the constant 1, or  $\bigvee \{x_j \mid j \in S\}$ , for some  $S \subseteq \{1, \dots, k\}$ . (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  $\alpha$ . So  $[\bar{\alpha} 1] = 1$  for all  $\alpha$ , 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  $2 \neq 2$ .

Suppose  $[001] = 0$ . Then  $[110] \neq 0$ , so  $[110] = 1$ . Also  $[220] \neq 0$ , so  $[220] = 2$ . Also  $[221] \neq 1$ ; 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  $[112] = 2$ ,  $[221] = 1$ ,  $\dots$ ,  $[xxy] = y$ . And  $[010] = 1$  or  $[100] = 1$  would imply  $[221] \neq 1$ .

Since  $[001] = 1$  implies  $[xxy] = y$ , we can conclude that  $[xyz] = z$  whenever  $x, y, z$  are distinct:  $[100] = 0$  implies  $[xyz] \neq [yxx] = x$ ;  $[010] = 0$  implies  $[xyz] \neq [xyx] = y$ .

Finally, if  $\pi$  is any permutation of  $\{0, 1, 2\}$  and  $g(x, y, z)$  is any polymorphism of ‘ $x \neq y$ ’, so is  $f(x, y, z) = g(x, y, z)\pi$ . We can choose  $\pi$  so that  $f$  is idempotent, because  $g(0, 0, 0) \neq g(1, 1, 1) \neq g(2, 2, 2) \neq g(0, 0, 0)$ . 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  $j$ ,  $1 \leq j \leq 3$ .

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<sup>+</sup> finds all 18 solutions in  $79\text{ M}\mu$ , with a search tree of 229 kilonodes. Algorithm B WTD finds them in  $3\text{ M}\mu$ , with a search tree of only 6208 nodes.]

[The same argument finds all polymorphisms of ' $x \neq y$ ' on  $\{0, 1, \dots, d-1\}$  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  $[x_1 \dots x_d] = y \neq x_d$ , 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  $z \notin \{x_1, \dots, x_d\}$ .]

**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  $\alpha \& \beta = \alpha \neq \beta$ ). (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  $\hat{R}$ , not  $R$ . The solution, with  $x = [011]$ ,  $y = [101]$ ,  $z = [110]$ ,  $w = [111]$ , is  $R(x, y, z) = \exists w(\hat{R}(x, y, w) \wedge \hat{R}(y, z, w) \wedge \hat{R}(z, x, w))$ .

**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  $4 \times 3$  matrix  $A$  (whose columns are  $(\alpha, \beta, \gamma)$ ) 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  $\alpha$  and  $\beta$ .]

**407.**  $S^{(3)}(x, y, z) = S^{(m)}(v, w, w, \dots, w) \wedge S^{(m)}(x, y, z, w, \dots, w)$  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  $[a_{11} \dots a_{k1}] \dots [a_{1m} \dots a_{km}] = a_{j1} \dots a_{jm}$  when the solution to  $\mathcal{I}_k(\Gamma)$  is the projection  $\pi_j$ . 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  $\Gamma$  for which  $[a_{11} \dots a_{k1}] \dots [a_{1m} \dots a_{km}] \notin R$ , then  $f$  would be a polymorphism of all gadgets derivable from  $\Gamma$ . 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.)

XCC  
projection  
Jeavons  
indicator problem

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  $R_0$ ,  $[x_1 \dots x_k]$  is unconstrained when  $\{1, 2\} \subseteq \{x_1, \dots, x_k\}$ , because such  $k$ -tuples never arise. Otherwise  $[x_1 \dots x_k]$  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  $R_2(x, y, z) = \exists v \exists w (R_1(x, v, z) \wedge R_1(y, w, z) \wedge R_1(v, w, z))$ .

Every polymorphism in this case is derived from a projection  $\pi_j$  in a curious way:  $[x_1 \dots x_k] = 0$  if and only if  $x_j = 0$ ; and  $[x_1 \dots x_k] = 2$  only if  $2 \in \{x_1, \dots, x_k\}$ . 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)  $x C_d^{\rightarrow} y = x \Theta_d y \Theta_d z_1 \Theta_d \dots \Theta_d z_{d-2} \Theta_d x$  for some  $z_1, \dots, z_{d-2}$ .

(c) The  $m$ -fold composition, namely ' $x R y = x \Theta_d z_1 \Theta_d \dots \Theta_d z_m \Theta_d y$  for some  $z_1, \dots, z_m$ ', 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  $C_d^{\rightarrow(s)} \circ R \circ C_d^{\rightarrow(t)}$  is true for all pairs  $(x, y)$  except  $((a-s) \bmod d, (b+t) \bmod d)$ .

From (b) and (c) we can therefore make a gadget  $R_x$  from  $\Phi_d$  that omits only the pair  $(x, x)$ , for  $0 \leq x < d$ . Hence  $N = R_0 \wedge \dots \wedge R_{d-1}$  is a gadget for the not-equal relation ' $x \neq y$ '—whose  $k$ -ary polymorphisms are known (see answer 401).

Every polymorphism of  $\Phi_d$  is a polymorphism of every gadget made from it. From (a), every  $k$ -ary polymorphism must satisfy  $[x \dots x] = x$ , for  $0 \leq x < d$ , because  $[x \dots x]$  is a unary polymorphism. So the argument of answer 401 proves that  $\Phi_d$  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  $[xy] = (xy = ab? \bar{a}: x)$ , 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  $\epsilon$ . 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)  $X^\nearrow = \{y \mid y \text{ is a multiple of } \text{lcm}(X)\}$ ;  $Y^\swarrow = \{x \mid x \text{ divides } \text{gcd}(Y)\}$ . Hence  $\{\overline{X}\} = \{A\} \cup \bigcup_{n \geq 1} \{\{d \mid d \text{ divides } n\}\}$ ;  $\{\overline{Y}\} = \{\emptyset\} \cup \bigcup_{n \geq 1} \{\{dn \mid d \geq 1\}\}$ .

(b)  $X^\nearrow = \{y \mid y^3 \text{ is a multiple of } \text{lcm}(X)^2\}$ ;  $Y^\swarrow = \{x \mid x^2 \text{ divides } \text{gcd}(Y)^3\}$ . If  $m = 2^{e_2} 3^{e_3} 5^{e_5} \dots$  and  $n = 2^{f_2} 3^{f_3} 5^{f_5} \dots$ , then  $m^3$  is a multiple of  $n^2 \iff 3e_p \geq 2f_p$  for all  $p \iff m$  is a multiple of  $2^{\lceil \frac{2}{3}f_2 \rceil} 3^{\lceil \frac{2}{3}f_3 \rceil} 5^{\lceil \frac{2}{3}f_5 \rceil} \dots$ ;  $m^2$  divides  $n^3 \iff 2e_p \leq 3f_p$  for all  $p \iff m$  is a divisor of  $2^{\lfloor \frac{3}{2}f_2 \rfloor} 3^{\lfloor \frac{3}{2}f_3 \rfloor} 5^{\lfloor \frac{3}{2}f_5 \rfloor} \dots$ . Hence  $\{\overline{X}\} = \{A\} \cup \bigcup_{Q \in \mathcal{Q}} \{\{d \mid d \text{ divides } n\}\}$ , where  $\mathcal{Q} = \{1, 2, 3, 5, 6, 7, 8, 10, \dots\}$  is the set of integers whose prime factorization has  $f_p \bmod 3 \neq 2$  for all  $p$ ;  $\{\overline{Y}\} = \{\emptyset\} \cup \bigcup_{n \geq 1} \{\{dn \mid d \geq 1\}\}$  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  $\begin{pmatrix} x & y \\ z & 0 \end{pmatrix}$ , by subtracting multiples of  $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$  (which commute with everything). If  $M$  and  $M'$  are  $2 \times 2$  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  $\begin{pmatrix} x & y \\ z & 0 \end{pmatrix}$  commutes with  $\begin{pmatrix} a & b \\ c & 0 \end{pmatrix}$  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  $\{M + \begin{pmatrix} t & 0 \\ 0 & t \end{pmatrix} \mid t \text{ integer and } M \in E(\begin{pmatrix} a & b \\ c & 0 \end{pmatrix})\}$ , where  $(a, b, c) = (0, 0, 0)$  or  $\text{gcd}(a, b, c) = 1$ .

(d)  $X^\nearrow = \{y \mid \sup X \leq y\}$ ;  $Y^\swarrow = \{x \mid x \leq \inf Y\}$ . Hence  $\{\overline{X}\} = \{\emptyset, A\} \cup \bigcup_{\alpha \text{ real}} \{(-\infty \dots \alpha] \cap A\}$ ;  $\{\overline{Y}\} = \{\emptyset, A\} \cup \bigcup_{\alpha \text{ real}} \{[\alpha \dots \infty) \cap A\}$ . (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  $\{\overline{X}\} = \{\emptyset, A\} \cup \bigcup_{\alpha \text{ real}} \{(-\infty \dots \alpha) \cap A\}$ ;  $\{\overline{Y}\} = \{\emptyset, A\} \cup \bigcup_{\alpha \text{ real}} \{(\alpha \dots \infty) \cap A\}$ . (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,  $Y^\swarrow = \{x \mid x \text{ is a root of } \text{gcd}(Y)\}$ .

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  $\text{gcd}(f(x), f'(x)) = 1$ .

**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  $\alpha$  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  $\alpha$  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

(c) In general there are  $f_n = d^{d^n}$   $n$ -ary operations on  $\{0, 1, \dots, d-1\}$ . Set  $d = 2$  and apply (a) to get  $e_4 = \binom{4}{4}2^{16} - \binom{4}{3}2^8 + \binom{4}{2}2^4 - \binom{4}{1}2^2 + \binom{4}{0}2^1 = 64594$ .

Willard  
projection functions  
nullary

**420.** (a) When  $d \geq k \geq 3$ , the function  $f(x_1, \dots, x_k) = [x_1, \dots, x_k]$  has  $k$  distinct elements] is full; but  $f_{ij}$  is constant (identically 0).

(b) True. For example, we can't have both  $f(a_2, a_2, a_3, \dots) = f(a_2, a'_2, a_3, \dots)$  and  $f(a_2, a'_2, a_3, \dots) = f(a'_2, a'_2, a_3, \dots)$  when  $f(a_2, a_2, a_3, \dots) \neq f(a'_2, a'_2, a_3, \dots)$ .

(c) We must have  $0 < r \leq k-3$ . By renumbering variables we can assume that  $i = 1$ ,  $j = k$ , and  $f(x_k, x_2, \dots, x_k) = h(x_{k+1-r}, \dots, x_k)$ , where  $h$  is full. Let  $g = f_{23}$ , and notice that  $g_{1k}(x_1, \dots, x_k) = f(x_k, x_3, x_3, \dots, x_k) = h(x_{k+1-r}, \dots, x_k)$  depends on  $x_k$ . Hence by (b),  $g$  depends on  $x_1$  or  $x_k$ . It also depends on  $x_{k+1-r}, \dots, x_{k-1}$ ; so it can't depend on  $x_3$ , by maximality of  $r$ .

(d) There are values  $b_1, \dots, b_r, b'_1$  such that  $f(b_1, b_2, \dots, b_r, x_{r+1}, \dots, x_k) \neq f(b'_1, b_2, \dots, b_r, x_{r+1}, \dots, x_k)$  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  $x_2, \dots, x_r$ . Hence  $f_{ij}$  is independent of  $x_{r+1}, \dots, x_k$ , by maximality of  $r$ . It follows that  $a_i = a_j$  implies  $f(a_1, \dots, a_k) = f(a_1, \dots, a_r, a_k, \dots, a_k) = h(a_1, \dots, a_r)$ .

Now suppose  $a_1 = a_j$  for some  $j > r$ , and let  $g = f_{1j}$ . Since  $g$  depends on  $x_2, \dots, x_r$ , it depends on at most one other variable; it must be independent of some variable  $x_l$  with  $l > r$ ,  $l \neq j$ . And  $a_j = a_1$  implies  $f(a_1, \dots, a_k) = f(a_1, \dots, a_{l-1}, a_1, a_{l+1}, \dots, a_k)$ , which is  $h(a_1, \dots, a_r)$ . The same argument works when  $a_i = a_j$  for  $i \leq r$  and  $j > r$ .

Finally, if  $a_i = a_j$  and  $i, j \leq r$ ,  $f_{ij}$  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  $l \leq r$ ,  $i \neq l \neq j$ . Then  $h_{ij}$  is also independent of  $x_l$ . We can evaluate  $f(a_1, \dots, a_k)$  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  $i_1 \dots i_l = 0 \dots 0$ . Otherwise begin with  $i_0 i_1 \dots i_{l-1} i_l \leftarrow 00 \dots 0m$  and  $r \leftarrow l$ ;  $r$  will be the position of the leftmost 'm'. After visiting  $i_1 \dots i_l$ , 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  $i_1 \dots i_l$ .

To generate the  $k$ -ary clone, represent the  $i$ th  $k$ -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  $n \geq k$  initial functions  $F_0, F_1, \dots, F_{n-1}$ ; 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  $i_1 \dots i_l$  with  $\max\{i_1, \dots, i_l\} = m$ , we compute the  $k$ -ary function  $F = g(F_{i_1}, \dots, F_{i_l})$ . 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  $n \leq d^{d^k}$ .

**422.** All functions  $a_1 x_1 + \dots + a_k x_k$  whose coefficients  $a_j$  are nonnegative integers and (i) exactly one  $a_j \equiv 1$ , (ii) no  $a_j \equiv 2$  (modulo 3). [This necessary condition is also sufficient, because we can obtain any such function  $f$  with  $\max\{a_1, \dots, a_k\} = a_j > 1$  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

$$f_{a_1 \dots a_{k-1}}(x_1, \dots, x_k) = t(f_{a_1}(x_1, x_k), x_k, f_{a_2 \dots a_{k-1}}(x_2, \dots, x_k)),$$

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  $a_1, \dots, a_{k-1}$ .

(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  $\sigma = a_1 \dots a_k$  is  $h(\sigma) = 1 + \max\{a_1, \dots, a_k\}$ . Every  $k$ -ary combo  $f$  of  $t$  defines a family of labelings  $\{\lambda(\sigma) = f(\sigma)\}$ , one for every RGS  $\sigma$  of length  $k$ , where  $0 \leq \lambda(\sigma) < h(\sigma)$ . Since there are exactly  $\binom{k}{h}$  RGSs of length  $k$  and height  $h$ , the total number of  $k$ -ary combos is at most  $N(k, d) = 1^{\binom{k}{1}} 2^{\binom{k}{2}} \dots d^{\binom{k}{d}}$ . 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*  $q(w, x, y, z) = t(t(w, x, y), t(w, x, z), z) = (w = x? y: z)$ . For every RGS  $\sigma = a_1 \dots a_k$  and label  $\lambda(\sigma)$ , let  $\mu(\sigma) = \min\{j \mid a_j = \lambda(\sigma)\}$ . Then the following combo works when  $k = 3$ :

$$f(x_1, x_2, x_3) = q(x_2, x_1, q(x_3, x_1, x_{\mu(000)}), x_{\mu(001)}), \\ q(x_3, x_1, x_{\mu(010)}, q(x_3, x_2, x_{\mu(011)}, x_{\mu(012)})))).$$

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  $\text{Inv}(O)$ , because  $R$  is preserved by every  $h(x_1, \dots, x_m) \in O_m$ : Given  $f_1, \dots, f_m$  in  $O_k$ , with corresponding  $d^k$ -tuples  $f_j(0, \dots, 0) \dots f_j(d-1, \dots, d-1) \in R$  for  $1 \leq j \leq m$ , we have  $h(f_1(x_1, \dots, x_k), \dots, f_m(x_1, \dots, x_k)) \in O_k$  because  $O$  is a clone; so  $h$ 's  $d^k$ -tuple belongs to  $R$ .

Now let  $P$  be the relation in  $\text{Inv}(O)$  that contains just the  $d^k$ -tuples for the projection operators  $\pi_1, \dots, \pi_k$ . Since  $g$  preserves  $\text{Inv}(O)$ , the  $d^k$ -tuple  $g(0, \dots, 0) \dots g(d-1, \dots, d-1)$  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  $a_{i1} + \dots + a_{im} = 1$  for  $1 \leq i \leq k$  and  $a_{1j} + \dots + a_{kj} = 1$  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)  $\Pi_{(1m)}D$ ; (b)  $\Pi_{(1m)}\emptyset$ ; (c)  $\Sigma_{12}D^2$ .

**433.**  $R \circ R' = \Pi_{14}\Sigma_{23}(R \otimes R')$ .

**434.**  $R \cap R' = \Pi_{12\dots m}\Sigma_{1(m+1)}\Sigma_{2(m+2)}\dots\Sigma_{m(m+m)}(R \otimes R')$ .

**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  $\{a_1 \dots a_n, b_1 \dots b_n\}$  satisfies  $a_1 \neq b_1, \dots, a_n \neq b_n$ ; (iv) products  $R_1 \otimes \dots \otimes R_t$  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  $k = 2$ .

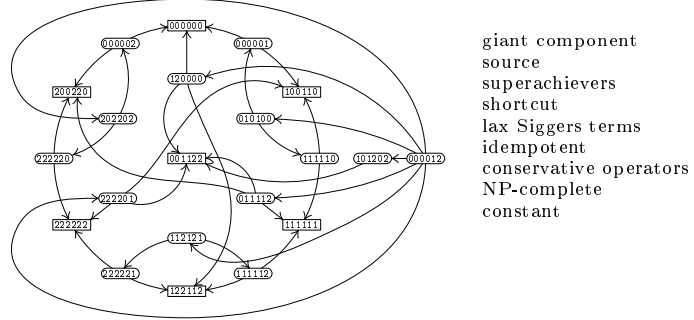
(c) Multiply answer (b) by  $d^d$ .

*Historical notes:* The significance of the NU condition was first pointed out by A. Huhn, *Acta Facultatis Rerum Naturalium Universitatis Comenianae Mathematica*, Mimoriadne Číslo (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  $v$ .



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 ‘ $!xy yxx \dots x:a yxx \dots x:a \dots xx \dots xy:a$ ’ for  $0 \leq a < d$ .

[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  $\prod_{t=1}^d t^{\{t\}_{d^t}}$ .

[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  $eD = \{x_1, \dots, x_m\}$  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  $\Gamma$  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  $\hat{\mathcal{P}}$  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  $\widehat{P}$ , without changing the complexity, because they merely break symmetry. Then all polymorphisms of  $\widehat{P}$  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  $w(x_1, \dots, x_k)$  is an LWNu in  $\text{Pol}(\Gamma)$ , and let  $e = ee$  be a squashing function. Let  $f(x) = w(x, \dots, x)$ , and  $\tilde{w}(x_1, \dots, x_k) = f \sim (w(e(x_1), \dots, e(x_k)))$ . Then  $\tilde{w}$  is also an LWNu in  $\text{Pol}(\Gamma)$ ; and  $\tilde{w}(x, \dots, x) = e(x)$  by exercise 442(d). So the restriction  $\hat{w}$  of  $\tilde{w}$  to the subdomain  $eD$  is a WNU for  $\widehat{\Gamma}$ .

Conversely, if  $\hat{w}$  is a WNU of  $\widehat{\Gamma}$ , let  $w(x_1, \dots, x_k) = \hat{w}(e(x_1), \dots, e(x_k))$ . [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  $\Gamma$ . We use the notation of answer 401, and derive some facts about every  $k$ -ary LWNu in  $\text{Pol}(\Gamma)$  for  $k \geq 3$ . Relation  $U$  implies that  $[x_1 \dots x_k] \neq 2$  whenever  $2 \notin \{x_1, \dots, x_k\}$ ; also  $[1-x_1, \dots, 1-x_k] = 1 - [x_1, \dots, x_k]$  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  $[0^{k-1}2] \neq 0$  implies  $[0^{k-1}1] \neq 0$  (by  $V$ );  $[1^{k-1}2] \neq 1$  implies  $[1^{k-1}0] \neq 1$  (by  $V$ ). And  $[2^{k-1}0] = 1$  implies  $[2^{k-1}1] = 1$  implies  $[0^{k-1}1] = 1$  (by  $W$ ).

Now let  $s = \min\{x_1 + \dots + x_k \mid 0 \leq x_1, \dots, x_k \leq 1 \text{ and } [x_1 \dots x_k] = 1\}$ . Then we have  $2 \leq s \leq k-s$ . Renumber the coordinates so that  $[1^s 0^{k-s}] = 1$ . Then  $[0^s 1^{k-s}] = 0$  (by  $U$ ); hence  $[0^s 2^{k-s}] = 0$  (by  $V$ ). Assume that  $[2^{k-1}0] = 2$ . Then  $[2^{s-1} 0 2^{k-s}] = 2$  (by LWNu). Hence  $[1^{s-1} 0 2^{k-s}] \neq 0$  (by  $V$ ). And  $[1^{s-1} 0 2^{k-s}] = 1$  implies  $[1^{s-1} 0^{k+1-s}] = 1$  (by  $W$ ), contradicting the minimality of  $s$ . Therefore  $[1^{s-1} 0 2^{k-s}] = 2$ ; but this implies  $[0^s 2^{k-s}] = 2$  (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  $\Gamma$  are precisely the functions  $f(x_1 \bmod 2, \dots, x_k \bmod 2)$ , where  $f$  is a monotone self-dual Boolean function, not a projection.)

**445.** The function  $\pi_j(x_1, \dots, x_k) = x_j$  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  $\mu_d$  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:

$$t(x_1, \dots, x_8) = f(f(x_1, x_4, x_4, x_4, x_4), f(x_2, x_5, x_5, x_5, x_5), \\ f(x_3, x_6, x_6, x_6, x_6), f(x_4, x_7, x_7, x_7, x_7), f(x_5, x_8, x_8, x_8, x_8)).$$

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  $(x, x, x, x, x, y, x, x)$  and  $(y, x, x, x, x, x, x, x)$ , to cover  $i \in \{1, 5\}$ ; evaluate it at  $(x, x, x, x, x, x, x, x)$ ,  $(x, x, x, x, x, x, y, x)$ ,  $(x, x, x, x, x, x, x, y)$ ,  $(x, x, x, x, x, y, x, x)$ , 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  $f(t_1[x, y], \dots, t_k[x, y]) = g(t_1, \dots, t_k)[x, y]$ , where  $g(t_1, \dots, t_k) = (\{x, y\} \text{ a majority pair? } \langle t_1 \dots t_k \rangle: (t_1 + \dots + t_k) \bmod 2)$ .)

*Historical notes:* If the minority law holds for all  $x$  and  $y$ , then  $g(x, y, z) = f(x, y, \dots, y, z)$  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, *Matematicheskii 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:  $\pi_1$ ,  $\pi_2$ ,  $\pi_3$ ,  $e\pi_1$ ,  $e\pi_2$ ,  $e\pi_3$ , 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  $0 \leq x, y < 4$ . 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  $0 \leq x, y < 4$ .

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

$$\square(h(x, y, z)) = (0 \in \{x, y, z\} ? 0 : 3 \in \{x, y, z\} ? 3 : h(x, y, z)),$$

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  $\sigma$  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  $(x=y=0? z \text{ or } 3 : xyz=0? 0 : [x=3] \neq [y=3]? 3 : z)$ .

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  $(xy=0? 0 : y=3 \text{ and } z>0? 3 : z>0? x : 3 \text{ or } x)$ .

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  $(xy=0? 0 : z=0? y \text{ or } 3 : x=3? 3 : y)$ .

Finally, each of the  $2 + 6 + 66 + 66$  (idempotent) Maróti pairs we’ve described yields a lax (non-idempotent) Maróti pair when we apply  $\sigma$  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

$$(r \star s)(x_{11}, \dots, x_{1n}, \dots, x_{m1}, \dots, x_{mn}) = r(s(x_{11}, \dots, x_{1n}), \dots, s(x_{m1}, \dots, x_{mn})).$$

universal algebra  
Dalmau  
Maltsev  
Historical notes  
Siggers  
Kearnes  
Marković  
McKenzie  
Barto  
Krokhin  
Willard  
mnemonic  
O’Donnell  
indicator problem  
self-dual  
symmetry  
Maróti 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  $f \sim f e = e$  and  $e f \sim = f \sim$  as in exercise 442. Now consider the term  $t(x_1, \dots, x_9) = ((f \sim q) \star (f \sim p))(e(x_1), \dots, e(x_9))$ . 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  $[x \dots x] = x$  for  $0 \leq x < 4$ . Then show that  $[x_1 \dots x_k] = 0$  only if  $0 \in \{x_1, \dots, x_k\}$ . We can't have  $[x_1 \dots x_k] = 1$  when  $\{x_1, \dots, x_k\} \subseteq \{2, 3\}$ , because that would imply  $[0 \dots 0] = 2$ . Hence  $[x_1 \dots x_k] \in \{x_1, \dots, x_k\}$ . Hence  $0 \notin \{x_1, \dots, x_k\}$  implies  $[x_1 \dots x_k] = x_j$  for some fixed  $j$ . Without loss of generality we can assume that  $j = k$ . Then  $[x_1 \dots x_{k-1} 0] = 0$ ,  $[x_1 \dots x_{k-1} 1] \in \{0, 1\}$ ,  $[x_1 \dots x_{k-1} 2] = 2$ ,  $[x_1 \dots x_{k-1} 3] \in \{0, 3\}$ . Equation (183) fails when  $(x, y) = (0, 2)$ .

**453.** The hinted gadget  $Ex = \exists uvwyz (uZvZw \cap zZyZx \cap zZw \cap uZx)$  can be found with the help of  $\mathcal{I}_2(\{Z\})$  as in exercise 405. Now  $2S = \exists uvw (uZvZwZu \cap uZx \cap vZy \cap wZz \cap Ex \cap Ey \cap Ez)$ . [Is that beautiful, or what?]

**454.** (a) True. If  $f(x_1, \dots, x_k)$  is a  $k$ -ary NU, let  $g(x_0, x_1, \dots, x_k) = f(x_1, \dots, x_k)$ . (b) False. Consider the  $k$ -ary Boolean relation  $x_1 \dots x_k \neq 0 \dots 0$ . [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  $a_1 \dots a_r \in \Pi_\sigma R$ , there are by hypothesis tuples  $\tau_j \in \langle R' \rangle_f$ , for  $1 \leq j \leq r$ , such that  $\Pi_\sigma \tau_j = a_1 \dots a_{j-1} b_j a_{j+1} \dots a_r$  for some  $b_j$ . Apply (144) to  $\tau_1, \dots, \tau_r$  (not  $\tau_r!$ ), to conclude that  $a_1 \dots a_r \in \langle \Pi_\sigma R' \rangle_f$ .

**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  $\approx 3.5$ ); another 2606  $f$ 's had the next-smallest observed value 580 (and average weight  $\approx 3.8$ ). The largest value was 1398; only 53  $f$ 's yielded a value exceeding 1000. The largest observed average weight was  $\approx 5.1$ , compared to  $\approx 7.4$  for (191).

**458.** For  $\Pi_{12}$  we need 002, 202, 212, 302 and either 012 or 013. For  $\Pi_3$  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  $j \notin \{i_1, \dots, i_{k-1}\}$ , 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  $t > d^{k-1} \ln(\binom{n}{k-1} d^{k-1})$ ; 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  $\tau_1, \dots, \tau_m$  (and the associated  $n$ -tuples  $\tau_1^*, \dots, \tau_m^*$ ). Use the procedure of answer 421 to visit every  $k$ -tuple  $\tau_{p_1} \dots \tau_{p_k}$  in order of increasing  $\max\{p_1, \dots, p_k\}$ . During each visit, if  $\tau = f(\tau_{p_1}, \dots, \tau_{p_k})$  isn't present in  $H$ , set  $m \leftarrow m + 1$ ,  $\tau_m = \tau$ ,  $\tau_m^* \leftarrow f(\tau_{p_1}^*, \dots, \tau_{p_k}^*)$ . And if  $\Pi_{\hat{\rho}}\tau \in R$ , also contribute  $\tau_m^*$  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  $f(x, y, z) = (x \cdot q)/(z \setminus (y \cdot q))$ , 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  $q \setminus (((q \cdot x)/y) \cdot z)$ .

**466.** The set partitions defined by the forkings on levels 1,  $\dots$ , 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  $a_1 \dots a_i \in \Pi_{1 \dots i} R$ , there is by hypothesis a tuple  $\tau = t_1 \dots t_m \in R'$  such that  $t_j = a_j$  for  $1 \leq j < i$ . Let  $R_i = a$ . Then  $(i, a, a_i)$  is a forking of  $R$ , hence also of  $R'$ . So  $R'$  contains tuples  $\tau' = x_1 \dots x_{i-1} a x_{i+1} \dots x_m$  and  $\tau'' = x_1 \dots x_{i-1} a_i y_{i+1} \dots y_m$ . By (144) and (198),  $\langle R' \rangle_f$  also contains  $f(\tau, \tau', \tau'') = a_1 \dots a_i z_{i+1} \dots z_m$ , 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),  $\dots$ , (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  $\tau \neq \tau'$ , there's a unique  $i$  such that  $t_1 = t'_1, \dots, t_{i-1} = t'_{i-1}$ ,  $t_i \neq t'_i$ . If  $(i, t_i, t'_i)$  isn't already a forking in  $S$ , insert both  $\tau$  and  $\tau'$  (thereby gaining also  $(i, t'_i, t_i)$ ). Otherwise contribute  $\tau$  and  $\tau'$  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  $\tau = cx_2 \dots x_{i-1} ax_{i+1} \dots x_m$ ,  $\tau' = y_1 \dots y_{i-1} ay_{i+1} \dots y_m$ ,  $\tau'' = y_1 \dots y_{i-1} a' z_{i+1} \dots z_m$ , then it contains also

$$f(\tau, \tau', \tau'') = cx_2 \dots x_{i-1} a' f(x_{i+1}, y_{i+1}, z_{i+1}) \dots f(x_m, y_m, z_m).$$

**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  $R'_{\perp c}$ , 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  $\tau = t_1 \dots t_m$  be a tuple of  $R$  with  $t_i = a$  and  $t_m = c$ . If  $a = a'$ , contribute  $\tau$  to  $R'_{\perp c}$ . Otherwise use the procedure of the text and exercise 470 to construct compact stubs  $R'_{t_1}, \dots$ ,

$(n, k)$ -universal set  
Bshouty  
superachievers  
Freese  
Maltsev  
historical note  
set partitions  
monoforkings  
trie  
forkings

$R'_{t_1 \dots t_{i-1}}$  for the subtrees  $R_{t_1}, \dots, R_{t_1 \dots t_{i-1}}$  of  $R$ . If  $a'c \notin \langle \Pi_{im} R'_{t_1 \dots t_{i-1}} \rangle$ , do nothing. Otherwise let  $\tau' = t_1 \dots t_{i-1} a' t'_{i+1} \dots t'_{m-1} c$  be a tuple of  $R$ ; contribute  $\tau$  and  $\tau'$  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  $P'_3, P'_{33}, \dots$ , leads to  $P'_{33302}$ , which has only two tuples:  $\tau$  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.

*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  $0 \leq j < n$ . (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  $\tau$ , where  $\tau$  is any element of  $S$ . Otherwise input a new constraint  $R$ , where  $R \subseteq D^r$  is an  $r$ -ary relation that constrains variables  $j_1, \dots, j_r$ . (We assume that  $f$  preserves  $R$ .)
- M3.** [Initialize  $q$ .] Set  $q \leftarrow 1$ . (We'll constrain variable  $j_q$ , given that variables  $j_1, \dots, j_{q-1}$  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  $T = \langle S_0 \rangle \cap R(j_1, \dots, j_r)$ , 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  $i j_1 \dots j_q$ , 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  $i j_1 j_2 j_3 j_4 j_5 = 577215$ , 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  $\tau$  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  $\tau^*$  be a tuple of  $T_0 = \langle S_0 \rangle$  such that  $\Pi_{\hat{\sigma}} \tau^* = \tau$ . (A dictionary  $H$  is used for this completion process, as in Algorithm N.) If  $a = a'$ , contribute  $\tau^*$  to  $S$ . Otherwise, as in answer 472, construct a compact Maltsev stub  $S'$  for the elements of  $T_0$  that begin with  $t_1 \dots t_{i-1}$ . Run through all tuples of  $\langle \Pi_{\hat{\sigma}} S' \rangle$ , looking for a tuple  $\tau'$  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  $\tau'^*$  be a tuple of  $T_0$  such that  $\Pi_{\hat{\sigma}} \tau'^* = \tau'$ . Contribute  $\tau^*$  and  $\tau'^*$  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  $R$ .

parity  
vector spaces  
MERCER  
contributes  
complete relation



**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  $a_1 \dots a_n$ , visit  $a_1 \dots a_n$  if it is a solution;  $O(\text{input length})$  gives us enough time to check. Then visit  $(a_1+2) \dots (a_n+2)$ ; 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  $\{0, 1, \dots, c\}$ . (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  $\Gamma$  is either  $\#P$ -complete, or there is a known algorithm to count the number of its solutions in polynomial time. In the latter case,  $\Gamma$  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  $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ . 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)  $R_{21} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$  puts *no* constraint on  $x_1 x_2$ ! The constraints begin to propagate as in (a): When  $k = 1$ ,  $R_{52} \leftarrow \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ ; then when  $k = 2$ ,  $R_{13} \leftarrow \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$ ,  $R_{53} \leftarrow \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ ; etc.;  $R_{55} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$  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  $R_{21} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}$ ,  $R_{31} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ , 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  $k = 2$ ,  $R_{13} \leftarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$  and  $R_{33} \leftarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ . Then  $R_{31} \leftarrow \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$ ,  $R_{32} \leftarrow \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$  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  $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ . (Don't ask what that means.)

**479.** It's true for  $r = 1$ ; so assume that  $r > 1$ . Let  $k = \max\{k_1, \dots, k_{r-1}\}$ , 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  $x_0 \dots x_p$  with  $x_0 = s$  and  $x_p = x$ , suitable  $x_p \dots x_q$  with  $x_p = x_q = x$ , and suitable  $x_q \dots x_r$  with  $x_q = x$  and  $x_r = t$ .

**480.** In each CSA below,  $I = \{q_0\}$  and  $Q$  is implicit.

(a)  $\Omega = \{q_1, q_3\}$ ;  $q_r \mapsto (* \leftarrow a^*? q_{(r+a) \bmod 5}: q_r)$ .

(b) Clearly  $n \leq 2d$ . Let  $e_a = 0^a 10^{d-1-a}$  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  $p_0 + \dots + p_{d-1} = n$ . Use the transitions  $q_0 \mapsto (* \leftarrow a^*? q(a, e_a): q_0)$ ;  $q(a, p) \mapsto R_{a,p}, (* \leftarrow b^*? q(b, p + e_b): q(a, p))$ , where  $R_{a,p} = \emptyset$  if  $p_a \neq 2$ ; otherwise  $R_{a,p} = \{v_1 \setminus a, \dots, v_n \setminus a\}$  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  $\Omega$  restricted to the  $\binom{d}{n/2}$  vectors  $p$  with no 1s.

(d) Let  $\Omega = \{q(n, a, b) \mid 0 \leq a < d, 0 \leq b < 2\}$ . Use transitions  $q_0 \mapsto R, (v_1 \leftarrow 0? q(1, 0, 0): q_0)$  and  $q(j, a, b) \mapsto R(j, a, b), (v_{j+1} \leftarrow a'^*? (a' = a + 1? q(j + 1, a + 1, 0): q(j + 1, a, 1)): q(j, a, b))$ , where  $R = \{v_j \setminus a \mid 1 \leq j \leq n, j \leq a < d\}$ ;  $R(j, a, 0) = \emptyset$ ; and  $R(j, a, 1) = \{v_{j+1} \setminus a + 2, v_{j+2} \setminus a + 3, \dots, v_n \setminus a + (n - j) + 1\}$ .

(e) Let  $S_j = (-1)^{v_1} + \dots + (-1)^{v_j}$ . A necessary and sufficient condition is that  $S_j \geq 0$  for  $1 \leq j < n$ , 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  $1 \leq j < n$  and  $0 \leq s \leq j$ .

[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  $a - 1 \in D_1 \cup \dots \cup D_{k-1}$ . 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  $j \geq 1$  and  $0 \leq a < d$ , together with the special state  $\perp$  that is reached when “symmetry is broken.” Let  $I = \{q(1, 0)\}$ . Use the transitions  $q(j, a) \mapsto (v_j \leftarrow a? q'(j, a): q(j, a + 1))$ ;  $q'(j, a) \mapsto v_{n+1-j} \setminus 0, \dots, v_{n+1-j} \setminus a - 1, (v_{n+1-j} \leftarrow a? q(j + 1, 0): \perp)$ ;  $\perp \mapsto (* \leftarrow *? \perp: \perp)$ . (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  $I = \{q_0\}$ ,  $Q = I \cup \{q(j, k) \mid 1 \leq j < k \leq n + 1\} \cup \{\#\}$ , 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

$$q(j, k) \mapsto (v_k \leftarrow a^*? (a < v_{k-j}? \#: a = v_{k-j}? q(j, k + 1): q(k, k + 1))).$$

[This method is attractive, but additional pruning is often possible. For example, if  $D_k$  is the  $k$ th 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  $i = \bar{i}$  (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  $R_i = \bar{i}$  and  $C_{\bar{i}} = i$ . 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:  $(b_{n'}, c_{n'}, d_{n'}, a_{n'}; \dots; b_{n-1}, c_{n-1}, d_{n-1}, a_{n-1})$ ;  $(c_{n'}, d_{n'}, a_{n'}, b_{n'}; \dots; c_{n-1}, d_{n-1}, a_{n-1}, b_{n-1})$ ;  $(d_{n'}, a_{n'}, b_{n'}, c_{n'}; \dots; d_{n-1}, a_{n-1}, b_{n-1}, c_{n-1})$ ;  $(\bar{d}_{n'}, \bar{c}_{n'}, \bar{b}_{n'}, \bar{a}_{n'}; \dots; \bar{d}_{n-1}, \bar{c}_{n-1}, \bar{b}_{n-1}, \bar{a}_{n-1})$ ; 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, ' $\text{not } ij$ ' is shorthand for ' $R_i \neq j, C_j \neq i$ '; this restriction is vacuous unless  $0 \leq i, j < n$ . 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 ' $\perp \mapsto (* \leftarrow *? \perp; \perp)$ ' as in answer 481.

$$\begin{aligned} q_1(i, j) &\mapsto R(i, j), (ij? \ q_2(i, j): q_1(i, j+1)); & r_1(i, j) &\mapsto \text{not } \overline{ij-1}, (ij? \ r_2(i, j): r_1(i, j+1)); \\ q_2(i, j) &\mapsto (j\bar{i}? \ q_3(i, j): s_2(i, j)); & r_2(i, j) &\mapsto (\bar{i}j? \ r_3(i, 0): \perp); \\ q_3(i, j) &\mapsto (\bar{i}j? \ q_4(i, j): s_4(i, j)); & r_3(i, j) &\mapsto \text{not } \overline{j-1}i, (j\bar{i}? \ r_4(i, j): r_3(i, j+1)); \\ q_4(i, j) &\mapsto (j\bar{i}? \ q_1(i+1, 0): \perp); & r_4(i, j) &\mapsto (j\bar{i}? \ r_1(i+1, 0): \perp); \end{aligned}$$

$s_2(i, j) \mapsto \text{not } j\bar{i}, (\bar{i}j? \ s_3(i, j+1): \perp); s_3(i, j) \mapsto \text{not } \overline{j-1}i, (j\bar{i}? \ r_4(i, j): s_3(i, j+1));$   
 $s_4(i, j) \mapsto \text{not } j\bar{i}, \perp; q_1(i, n) = r_1(i, n) = r_3(i, n) = s_3(i, n) = \perp$ . Here  $R(i, j)$  stands for the restrictions ' $\text{not } (j-1)\bar{i}$ ,  $\text{not } \overline{ij-1}$ ,  $\text{not } \overline{j-1}i$ ', as well as four more when  $i = \lceil n/2 \rceil$ : ' $\text{not } \bar{i}j$ ,  $\text{not } ji$ ,  $\text{not } i\bar{j}$ ,  $\text{not } j\bar{i}$ '. 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  $\text{not } jn'$  for  $n' < j < n$ ,  $(n'0? \ \perp; s_1(1)); s_1(j) \mapsto \text{not } \overline{j-1}n'$ ,  $(n'j? \ \perp; s_1(j+1))$  for  $0 < j < n'$ ; and  $s_1(n') \mapsto (n'n'? \ q_1(n'+1, 0): \perp)$ .

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  $\hat{Q}(n) = 8\hat{Q}_a(n) + 4\hat{Q}_s(n) + 2\hat{Q}_d(n)$  solutions (see answer 7.2.2.1–24), we have

$n$	10	11	12	13	14	15	16	17	18	19	20
$\hat{Q}_a(n)$	0	5	18	231	642	4040	25320	166201	1115373	8060958	61981118
$\hat{Q}_s(n)$	1	1	2	6	11	49	79	245	498	1192	3798
$\hat{Q}_d(n)$	0	0	2	2	0	0	12	17	0	0	60
$\hat{Q}(n)$	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

JOHNSON  
Richardson

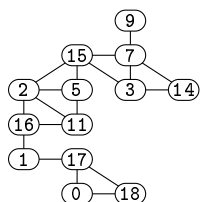
Algorithm 7.2.2.3A, 232.  
Algorithm 7.2.2.3B, v, 227–228.  
Algorithm 7.2.2.3C, v, 63–64.  
Algorithm 7.2.2.3C<sup>+</sup>, v, 227.  
Algorithm 7.2.2.3D, 52–53.  
Theorem 7.2.2.3D, 97.  
Algorithm 7.2.2.3E, 232.  
Corollary 7.2.2.3E, 22.  
Algorithm 7.2.2.3F, v, 235–236.  
Theorem 7.2.2.3F, 91–92.  
Algorithm 7.2.2.3G, 225.  
Algorithm 7.2.2.3H, 214–215.  
Theorem 7.2.2.3H, 26–27.  
Theorem 7.2.2.3I, 80–82.  
Algorithm 7.2.2.3I, 225–226.  
Algorithm 7.2.2.3J, 233–234.  
Algorithm 7.2.2.3M, 258.  
Lemma 7.2.2.3M, 104.  
Theorem 7.2.2.3M, 28.  
Algorithm 7.2.2.3N, 102.  
Lemma 7.2.2.3N, 99.  
Algorithm 7.2.2.3O, 231.  
Algorithm 7.2.2.3O<sup>+</sup>, 233.  
Lemma 7.2.2.3O, 22.  
Theorem 7.2.2.3P, 93.  
Algorithm 7.2.2.3P, 174.  
Algorithm 7.2.2.3Q, 162.  
Algorithm 7.2.2.3R, 224.  
Lemma 7.2.2.3R, 84.  
Theorem 7.2.2.3R, 85.  
Algorithm 7.2.2.3S, v, 72–73.  
Theorem 7.2.2.3S, 23.  
Definition 7.2.2.3V, 50.  
Algorithm 7.2.2.3V, 106.  
Corollary 7.4.1.2S, vii.  
Lemma 7.4.1.2S, vii.  
Algorithm 7.4.1.2T, viii–ix.  
Theorem 7.4.1.2T, viii.  
Algorithm 7.5.1H, x–xi.  
Theorem 7.5.1H, xi.  
Corollary 7.5.1K, xi.

*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)

# ANSWERS TO PUZZLES IN THE ANSWERS

LIFE



(see answer 69(b))

4 1 4 4	4 4 2 4	4 4 4 1	4 1 4 4	3 1 2 2	4 4 4 1	
4 4 1 4	4 4 2 4	4 2 2 4	4 4 5 4	3 4 4 1	2 4 3 3	
1 4 3 4	1 3 4 4	2 4 4 4	2 4 5 4	3 1 4 4	2 3 1 3	(see answer 293)
2 2 3 3	3 3 2 2	2 1 2 2	2 5 5 5	1 2 2 1	3 3 2 2	
(i)	(ii)	(iii)	(iv)	(xi)	(xii)	

```

1 4 4 4 4 1 m 5 3 3 3 4
4 3 2 1 2 2 m 5 4 2 2 4
4 3 2 3 3 3 m 5 4 3 1 4
4 3 4 4 4 4 m 5 4 3 2 4
4 5 5 5 5 5 m 5 4 3 2 1
m m m m m m m m m m m m m m
1 2 3 4 5 m 5 5 5 5 5 4
4 2 3 4 5 m 4 4 4 4 3 4
4 1 3 4 5 m 3 3 3 2 3 4
4 2 2 4 5 m 2 2 1 2 3 4
4 3 3 3 5 m 1 4 4 4 4 1
  
```

(see answer 294)

5 1 7 7 5 5 4 4 4	4 1 8 7 5 9 6 2 3	
5 5 7 3 3 5 5 4 1	2 5 9 3 8 6 7 4 1	
5 7 7 7 3 1 5 9 9	3 6 7 2 4 1 5 8 9	
5 7 2 9 2 2 9 9 7	5 7 6 8 1 2 9 3 4	
8 3 2 9 9 9 9 7 7	8 3 2 6 9 4 1 7 5	(see answer 297)
8 3 3 4 2 3 2 2 7	9 4 1 5 7 3 2 6 8	
8 8 4 4 2 3 3 5 7	7 9 4 1 2 8 3 5 6	
1 8 8 4 6 5 5 5 7	1 2 3 4 6 5 8 9 7	
8 8 6 6 6 6 6 5 7	6 8 5 9 3 7 4 1 2	

*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.

$<, \leq, >, \geq$  relations, 87–88, 245.

$\perp$  (empty relation), 245.

$\perp$  state, 260.

$\top$  (complete relation), 245.

$\iff$ : If and only if.

$0 \times n$  matrix, 241.

0-tuple, 113.

0/1/all constraints, 80.

0s and 1s, 107.

matrices of, 80, 148–149, 239–240.

$2 \times 2$  matrices, 115, 118, 151.

2-regular graphs, 121.

2SAT problem, 46, 87, 239.

3-coloring a graph, 41, 91, 135–136.

3-cube, 116.

3-PARTITION problem, 194.

3-regular graphs, 44, 121.

3D objects projected to 2D, 9; *see* Line labeling problem.

3DM (three-dimensional matching), 159.

3SAT problem, 46, 89, 90, 113, 130, 173.

3VP (3-valent polyhedral object),  
9–16, 115–119.

4-cube, 201.

4-fold symmetry, 203, 214, 220–221.

4-letter words of English, 74.

4 queens problem, 139.

4-regular graph, 30.

4-vertex graphs, 134.

5-letter words of English, 29, 74, 138.

5-queens domination problem, 78.

5 queens problem, 139.

5-vertex graphs, 134.

8 queens problem, 51, 67.

13-colonies graph, 17, 121.

16 queens problem, 54, 65, 79, 139, 146.

90-degree rotational symmetry, 203, 214.

180-degree rotational symmetry, *see*  
Central symmetry.

(132, 312)-avoiding permutations, 219.

(213, 231)-avoiding permutations, 219.

$\alpha$ -graceful graphs, 123–124, 178, 183.

$\alpha$ -labeling, 182.

$\gamma$  (Euler's constant), 188.

as source of “random” data, 116.

$\lambda$  (Laplace limit), 165.

$\mu$  (dual Laplace limit), 165.

$\nu x$  (1s count, sideways sum), v, 58,  
161, 193, 247.

$\pi$  (circle ratio), *see* Pi.

$\pi_j$  (projection to coordinate  $j$ ), *see*  
Projection function.

$\Pi_\sigma R$  (projection of a relation), 48, 96,  
99–101, 152, 241.

$\varpi_n$  (Bell number), 64, 251.

$\rho x$  (ruler function, rightmost bit), v,  
58, 183, 224.

$\phi$  (golden ratio), 163, 170.

as source of “random” data, 116.

**A** (current age), 231.

$a$ -code, *see* Asterisk codes for subcubes.

a.s.: Asymptotically almost surely, 130.

AAAI: American Association for Artificial  
Intelligence (founded in 1979);  
Association for the Advancement of  
Artificial Intelligence (since 2007).

Abrham, Jaromír, 175, 178.

AC3rm algorithm, 109.

AC6 algorithm, 54, 109.

AC2001 algorithm, 109.

Academic problems, 67, 106.

Acharya, Belmannu Devadas

(ಬೆಳಮಣ್ಣು ದೇವದಾಸ ಅಚಾರ್ಯ), 184, 185.

Active items and options, 54, 62, 71.

**ACTIVE** pointer, 62, 71, 143, 231.

Active variables, 50, 138.

Acyclic digraph (dag), 82.

Acyclic orientations, 242.

Adamaszek, Michał Jan, 120, 175.

Adams, Peter, 190.

Adjacency matrices, 179.

Affine equivalence, 125–127.

**AGE**( $\sigma'$ ), 232.

Age of an option, 71–72, 144, 231.

Alekseev, Vladimir Evgenievich (Алексеев,  
Владимир Евгеньевич), 171.

Algebra of relations, 96, 152.

Algebraic number, 249.

- All-different constraint, 6, 30, 34, 113, 131, 160, 164, 166, 213.
- All-interval tone rows, 74.
- All-ones matrices, 155, 241.
- All shortest paths, 107.
- All-zero matrices, 155, 241.
- All-zero rows, 148.
- Alon, Noga Mordechai (נוגה מרדכי אלון), 111, 204.
- Alphabet blocks, 56, 74.
- Amazing identity, 186.
- Amazons, 157.
- Analysis of algorithms, 22–23, 113, 114, 118–120, 122–125, 136–140, 142–144, 150.
- Ancestor in a tree, 129.
- AND operation, 142, 245.
- ANDN (and not), 201.
- Anick, David Jay, 24, 122, 182.
- Antiautomorphisms, 25, 125, 187.
- Antisymmetric relations, 148.
- Antisymmetries, 25, 125.
- Appleton, Jenna Bou, 190.
- Approximate GAD filtering, 133.
- Arc consistency, 109, *see* Domain consistency.
- Arcs, 3.
- Arity, 3, 112, 139.
- Artigues, Christian, 9.
- Assignment of value to variable, 50, 54, 58, 106.
- Associative law, 148.
- Asterisk codes for subcubes, 129.
- Asterisk in transition rule, 156.
- Asymptotic methods, 113, 115, 122, 130, 140, 225, 243, 256–257.
- At-least-one clauses, 39, 46.
- At-most-one clauses, 39, 46, 137.
- Atom in a molecule, 130.
- Attributes in a database, 4.
- Augmented Sierpiński simplex graph, 45.
- Augmented Sierpiński tetrahedron, 45, 137.
- Automaton, 112, 155–157.
- Automorphisms, 17, 21, 23, 25, 32, 33, 38, 120, 121, 125, 128, 151, 153, 166, 178, 179, 181, 182, 187, 195, 199, 201.
- Auxiliary variables, 46, 88–90, 212, 238, 239.
- AVAIL list, 145, 240.
- Average level, 143.
- Axial symmetry, 203, 223.
- Axioms, 148.
- Aztec diamonds, 142, 184.
- Aztec rectangles, 221–223.
- Bacchus, Fahiem, 216.
- Background plane, 172.
- Backmarking, 38, 142.
- Backtrack-free process, 80–85, 99–105.
- Backtrack programming, 38, 65, 109–110, 120, 181, 192, 202, 219, 237.
- Backtrack tree, 54, 218.
- Backward step, 142.
- Bad strong component, 81.
- Baker, Kirby Alan, 100, 252.
- Balanced pentomino patterns, 78, 79.
- Balanced ternary digits, 39, 85, 90.
- Balls, two-dimensional, 78.
- Bandwidth, 38, 113.
- Barette, Olivier Franz René, 110.
- Barto, Libor, 251, 254, 255.
- Barycentric coordinates, 135, 172.
- Basic constraints, 83.
- Basis vectors, 244.
- Baumert, Leonard Daniel, 109.
- Baxter, Nicholas Edward, 74.
- Bayramov, Rauf Abdulla oğlu, 110.
- BDD: A reduced, ordered, binary decision diagram, 204, 244, 245.
- Becket, Ralph William Allen Jude, 111.
- Beek, Peter George van, 110, 111, 168, 216.
- Beldiceanu, Nicolas, 112.
- Bell, Alexander Graham, 44.
- Bell, Eric Temple, numbers, 64, 251.
- Beluhov, Nikolai Ivanov (Белухов, Николай Иванов), iv, 186, 203, 204, 221, 222.
- Benchmark problems, v, 7, 56, 64–65, 69, 74, 78–79, 111, 134, 211.
- Bennaceur, Hachemi Ahmed, (هاشمي أحمد بن ناصر), 158.
- Bent trominoes, 221.
- Berge, Claude Jacques Roger, 198.
- Bessière, Christian Paul André, iv, 54, 109, 216.
- Bi-degree of a vertex, 198.
- Biaxial symmetry, 223.
- Biconnected graph, 118.
- Biere, Armin, 43, 211.
- Bigraph, *see* Bipartite graph.
- Binary branch, 50, 66, 77, 144, 218.
- Binary constraints, 2, 53, 58, 85, 107, 109, 137, 139, 148, 149, 158, 215, 225, representing  $k$ -ary constraints, 47, 137, 139; *see also* Hidden variables.
- Binary encoding, 40–42, 207–208.
- Binary notation, 39.
- Binary recurrence, 122.
- Binary relations, *see* Binary constraints.
- Binary representation, 135.
- Binary search trees, 219.
- Binary trees, 135.
- Binding, 50, 142.
- Binomial tree, 122.
- Bipair, 236.
- Bipartite graph, 33–34, 38, 47, 76, 121, 123, 124, 131, 132, 159, 191, 198, 215.
- Bipartite matching, ix, 33–34, 37, 131–132, 135.

- Birkhoff, Garrett, 92.  
 Birthday, 146.  
 Bit vectors, 58, 60, 142, 212.  
 Bitsets, 60, 142.  
 Bitwise operations, 54, 58, 108, 133,  
     181, 201.  
     AND, 181, 193, 212, 224.  
     complement, 174, 193.  
     OR, 193, 224.  
     XOR, 193.  
 Bitwise parity constraint, 104.  
 Bivariate exponential generating functions,  
     165, 241–242.  
 Bizarre graph, 64.  
 Blank (unmatchable) color, 237.  
 Blocked option, 71.  
 Bloom, Gary Stuart, 126, 188.  
 Blurred state, 174–175.  
 Bodnarchuk, Viktor Gavrilovich  
     (Боднарчук, Виктор Гаврилович),  
     86, 110.  
 Boltzmann, Ludwig Eduard, 5.  
*book* graphs, 134.  
 Boolean algebra, 43.  
 Boolean binary relations, 87, 149.  
 Boolean clones, 251.  
 Boolean functions, 89, 95, 204, 245.  
 Boolean lattice, 24, 125, 126.  
 Boolean matrices, 58, 140, 215.  
     multiplication of, 107.  
 Boolean operations, 151, 245.  
 Boolean satisfiability, 1.  
 Boolean ternary relations, 87–90, 150, 246.  
 Boolean variables, 2, 8, 39, 113.  
 Bosák, Juraj, 22.  
 BOUND field, 77–78.  
 Boundary cycle, 13, 117–118, 163, 170.  
 Bounded degree, 130, 196.  
 Bounded permutations, 242.  
 Bounds consistency, 112.  
 Bousquet-Mélou, Mireille Françoise, 243.  
 Boussemart, Frédéric Claude, 67, 68.  
 Bowtie graph ( $K_1 \text{ --- } 2K_2$ ), 180.  
 Bowtie operation  $\bowtie$  (join), 96, 241.  
 Bowtie relation between edges, 128–129.  
 Brady, Zarathustra Elessar (= Zeb), 111.  
 BRAIN83 (brain graph), 30–31, 35, 131.  
 Branching option, choice of, 77–78.  
 Branching variable, choice of, 50, 66, 138.  
 Breadth-first search, ix, xi, 196, 200.  
     versus depth-first search, 175.  
 Breaking symmetries, 21, 156, 176, 223.  
 Bridge in a graph, 170, 180.  
 Brinkmann, Gunnar, 179.  
 Bryant, Darryn Edward, 190.  
 Bshouty, Nader Hanna (نادر حنا بشوتي,  
     נאדר חנא בשותי), 257.  
 Bucket sort, 133, 233.  
 Bulatov, Andrei Arnoldovich  
     (Булатов, Андрей Арнольдович),  
     97, 98, 111, 254, 259.  
 Bull graph, 134.  
 Buratti, Marco, 127.  
 Burnside, William, lemma, 188, 191.  
 $C_5$  (5-cycle), 18, 22, 29, 67, 113, 128.  
 $C_6$  (6-cycle), 22, 191.  
 $C_m^{\rightarrow}$  (oriented  $m$ -cycle), 26, 126, 150, 159.  
 $C_n$ : cycle graph of order  $n$ , 36, 123,  
     178, 180.  
 Cabon, Bertrand, 56.  
 Cache-friendly data structure, 60.  
 Cahit Arkut, Ibrahim, 183.  
 Cancellation laws, 154.  
 Canonical form, 21.  
 Canonical solutions, 157.  
 Car sequencing problem, 7–9, 38, 114.  
 Cardon, Stéphane Michel, 216.  
 Carletti, Vincenzo, 200.  
 Carlsson, Mats Hugo, 112.  
 Carmichael, Robert Daniel, 189.  
 Cartesian coordinates, 116–117, 162.  
 Cartesian product, 2, 113.  
     of graphs ( $G \square H$ ), 29, 128, 129; *see*  
         also Grid graphs.  
     of relations ( $R \times S$ ), 96, 152.  
 Catalan, Eugène Charles, numbers, 225.  
 Caterpillar nets, 24, 124.  
 Cauchy, Augustin Louis, 165.  
 Central symmetry (180° symmetry), 10,  
     203, 223, 244, 261.  
 Centroid of a free tree, 221.  
 Certificate of correctness, xi, 261.  
 Certificate of unsatisfiability, 112, 157, 261.  
 Chain problem, 140.  
 Channelling constraints, 8.  
 Chatterjee, Sourav (সৌরভ চ্যাট্টোপাধ্যায়), 130.  
 Chaucer, Geoffrey, 158.  
 Checkerboard pattern, 221.  
 Chemical compounds, 108, 130.  
 Chen, Hubert (= Hubie) (張胡比), 111.  
 Chen, Xinguang (= Albert) (陈新光), 216.  
 Chessboard, 64, 133, 147, 184.  
 Chesterton, Gilbert Keith, 1.  
 Choi, Chiu Wo (= Jeff) (蔡昭和), 112.  
 Chordless cycle, 191.  
 Chosen option, 71.  
 Christie Mallowan, Agatha Mary Clarissa  
     Miller, 132, 157.  
 Chromatic numbers, 121.  
 Chunky-Octs®, 65, 142.  
 Chvátal, Václav (= Vasek), graph, 30–32,  
     35, 131, 132, 135.  
 Circuit analysis, 108.  
 Clauses, 1, 39.  
     learning, 108, 261.



- Clément de Saint-Marcq, Vianney  
Emmanuel Adrien Martin Marie  
Joseph, le, 110.
- Clique hints, 44, 136, 137, 207–211.
- Cliques, 3, 20, 29, 37, 65, 159, 204, 211.
- Clone, 94–97, 151, 152, 251, 252.
- Closed knight's tour, 75–76, 146.
- Closed subsets, 93, 96, 150.
- Clowes, Maxwell Bernard, 10, 11, 107.
- CLR field, 61–64, 143.
- Co-clones, *see* Relational clones.
- Codd, Edgar Frank, 4.
- Cohen, David Avram, 100, 110, 111,  
241, 254.
- Cohn, Paul Moritz, 2, 251.
- Cohoon, David Kent, 160.
- Collinear points, 237.
- Color of an item, 77, 225.
- Coloring a graph, 41–45, 64, 135–138, 259.
- Column convex matrix, 84.
- Column sums, 242.
- Combos, 94, 251.
- Commas, 2.
- Commutative law, 97, 148, 254.
- Commuting matrices, 249.
- Compact Maltsev stubs, 104, 154.
- Compact representation of digraphs, 26, 189.
- Compact stub for an NU polymorphism,  
101–103, 111, 154–155.
- Compact-Table data structure, 110, 142.
- Comparison of multisets, 131.
- Compatible options, 54, 69, 77, 144.
- Compatible vertices, 130.
- Competitions, 9, 111, 211–212.
- Complement of a graph, 159, 160, 190, 200.
- Complement of a labeling, 16, 19–21,  
120, 122.  
edge-complement, 123.
- Complement symmetry, 178.
- Complete binary tree, 195, 209.
- Complete bipartite graph ( $K_{m,n}$ ),  
123–124, 180, 242.  
oriented, 126.
- Complete graph ( $K_n$ ), 3, 19, 120–122,  
124, 195; *see also* Cliques.
- Complete relation  $[0..d]^n$ , 80, 84, 102,  
152, 154, 163, 213, 258.
- Complex variable theory, 165.
- Complexity of computation, 168, 173.
- Components of a graph, *see* Connected  
components, Strong components.
- Composite numbers, 76.
- Composition of binary relations, 53,  
139, 152, 248.
- Composition of functions, 94, 153, 255.
- “Compound” Boolean variables, 215.
- Computer science versus mathematics, 239.
- Computer vision, 9, 158.
- Concatenating relations, 96.
- Concave edges, 10–12, 116, 166.
- Conditional branches, 227.
- Configurations, 4–5.
- Conflict clauses, *see* Preclusion clauses.
- Conflict-directed heuristic, 67.
- Congruence enumeration, 188.
- Conjugates of an algebraic number, 249.
- Connected components, 117, 121, 122, 124.
- Connected graphs, 129.
- Connected patterns, 133.
- Connected row convex constraints,  
84–85, 110, 148–149.
- Connectome, *see* **BRAIN83**.
- CoNP-complete, 49.
- Consecutive 1s, 169.
- Conservative polymorphisms, 154, 244,  
246, 253, 256.
- Consistency, 48–53, 80, 109, 112, 156.
- Constant function, 253.
- Constant relation, 3, 91.
- Constraint, 1, *see* Relation.
- Constraint network, 12, 79, 107.
- Constraint satisfaction automaton,  
112, 155–157.
- Constraint satisfaction problem, 1–5, 115.
- Contest, 9, 211.
- Contiguous United States of America,  
18, 32–35.
- Contraction of a graph, vi, 209.
- Contributing to a compact stub, 102,  
154, 256–258.
- Converse of a digraph or binary relation,  
126, 245.
- Convex edge, 10–12, 116, 166.
- Convex polyominoes, 243.
- Coolsaet, Kris Karel, 179.
- Cooper, Martin Charles, 100, 110,  
150, 240, 241.
- Coordinates, 116–117, 162.
- Coplanar, 166.
- Copy of a graph, *see* SIP.
- Copying domains, 201.
- Cordella, Luigi Pietro, 200.
- Core of a set of constraints, 106, 153.
- Correspondence relation, 80.
- Counting, complexity of, 259.
- $\text{cover}(i)$ , 54.
- Covering the cycles, 135.
- Covering the nogoods, 212.
- Covering with disks, 146.
- CRC (connected row convex) constraints,  
84–85, 110, 148–149.
- Creases, 166.
- Critical block, *see* Hall set.
- Crossword diagram, 79.
- Crystal maze puzzle, 6, 113.
- CSA: Constraint satisfaction automaton,  
112, 155–157.

- CSP: The constraint satisfaction problem,  
 1, 30, 205.  
 represented as SAT, 2, 113.  
 represented as XC, 135.  
 represented as XCC, 2, 113, 135.
- CSP solvers, 39, 112.
- CSPLIB problems, 7.
- Cube, subcubes of, 129.
- Cubes, colored, 64.
- Cubic (3-regular) graphs, 44, 121.
- Cubies, 115, 167.
- Cung, Vàn Đàt, 9.
- Current domain, 50.
- Cuts, Dedekind, 249.
- Cutting edges, 196.
- Cycle graph  $C_n$ , 22, 36, 67, 113, 124, 178.  
 when  $\alpha$ -graceful, 123.  
 when graceful, 183.  
 when rainbow graceful, 28.
- Cyclic  $(v, k, \lambda)$ -difference set, 126, 127.
- Cyclic shifts, 156.
- $D_v$  (domain of variable  $v$ ), 52.
- $d^+(v)$  and  $d^-(v)$  (out-degree and in-degree), 126.
- $d$ -ary branching, 66, 75.
- $d$ -ary sequence: Sequence consisting of digits  $\{0, 1, \dots, d-1\}$ , 55.
- $d$ -way branching, 66, 77, 109, 139, 144, 218, 230.
- Dag: A directed acyclic graph.
- Dalmau Lloret, Victor Xavier, iv, 111, 255.
- Dancing cells, 61–65, 77, 146, 227.  
 versus dancing links, 64–65, 73, 78, 147.
- Dancing links, 55, 59, 69, 73, 75, 77, 79, 142, 146, 208, 211, 238.
- Dart graph, 180.
- Data compression, 99, 104.
- Data structures, ix, 17, 20, 26, 55–62, 72, 102, 110, 112, 121, 133, 142, 145, 174, 182, 200, 214.
- Databases, 4, 96, 216.
- DC: Domain consistency, 50.  
 versus FC, 51–56, 74–76, 138.
- de Givry, Simon, 56.
- deactivate( $i$ ), 235.
- Dead state, 260.
- Debruyne, Romuald Gerard Édouard, 216.
- Dechter, Rina Kahana (רינה כהנא דכטר), 110, 111, 215, 225.
- Decision problem, 96, 117, 119.
- Decomposable constraints, 48, 100, 102, 137, 246.
- Dedekind, Julius Wilhelm Richard, 87, 249.
- Degenerate binary search trees, 219.
- Del Fra, Alberto, 127.
- Delayer, 218.
- Deletion from sparse-set, 227.
- Delorme, Charles Jean-Pierre, 126.
- Demaine, Erik Dylan Anderson, 223.
- Demeulenaere, Jordan Patrick Christian, 110.
- Demirović, Emir, 112.
- Density of  $d$ 's, 141.
- Dependence on arguments, 91.
- Depth-first forest, vi.
- Depth-first search, vi, ix, xi.  
 versus breadth-first search, 175.
- Derived operators, 94–95.
- Determinant, 116, 122, 170, 182.
- Deville, Yves Dominique Bernard, 110, 196.
- Dhar, Vasant Kumar (वसन्त कुमार धर), 213.
- Diaconis, Persi Warren, 130.
- Diagonalization, *see* Equality selection relation.
- Diameter 4, 24.
- Dichotomy conjecture, 111.
- Dichotomy theorem, 96–98, 111, 155, 259.
- Dieker, Antonius Bernardus, 217.
- Differential equation, 165.
- Digits, extended hexadecimal, 188.
- Digraceful graph, 27, 126–127.
- Digraphs (directed graphs), 2–3, 81–82, 125, 130, 132, 135, 152, 161, 195, 239, 248.  
 graceful, 24–27, 125–127.  
 representation of, 26, 125.  
 small, 126.
- Dinçbaşı, Mehmet, 8.
- Direct encoding, 2, 39, 40, 46, 47, 137, 207–212.
- Direct product of graphs ( $G \otimes H$ ), 124.
- Directed acyclic graphs, 38, 130, 242.
- Directed cycles, 135.
- Directed graphs, *see* Digraphs.
- Discrete disks, 78, 146.
- Discrete Fourier transform, 163.
- Discriminator, 152, 244.
- Disequality, *see* Nonequality relation.
- Disney, Walter Elias, 159.
- Distinct representatives, 197.
- Distributive lattices, 197.
- Dobbelaere, Bert Omer Rachel, 181.
- Dodecahedron, 121, 131.
- dom/wdeg heuristic, 68.
- Domain consistency, 50–53, 69–73, 83–85, 109, 112, 138–140, 142, 145, 148, 174, 211, 221, 239.
- Domain filtering, 49–53, 139.  
 bitwise, 201.
- Domain versus range, 113.
- Domains, 1, 8, 19, 70, 98, 113, 133, 147, 199.  
 representation of, 56–60, 142, 147.
- Domination, 78.
- Dongen, Marc Remko Casper van, 111, 211, 217.
- Double paw graph, 118, 180.
- Double-snake puzzle, 64.
- Doubly linked lists, 58–61, 63, 72, 142, 225.

- Downloadable programs and data, ii, v, 163, 165, 181, 182, 206, 220, 230, 246.
- Downpath, viii.
- Draper, Stephen William, 171.
- Dreweke, Alexander, 196.
- Dual discriminator, 244.
- Dual Horn clauses, 49, 215.
- Dual Laplace limit, 165.
- Dual of a Boolean function, 245.
- Dual of a CSP, 139.
- Dudeney, Henry Ernest, 64.
- Dull, Brutus Cyclops, 160.
- Dürer, Albrecht, 115, 203.
- Duval, Jean-Pierre, 260.
- Dynamic programming, 162, 164, 172.
- Dynamic variable ordering, 65–69, 144.
- $e$  (base of natural logarithms), as source of “random” data, 141.
- $ea(f)$  (essential arity of  $f$ ), 91–92, 151.
- ECAI: European Conference on Artificial Intelligence* (1982–), formerly called *Artificial Intelligence and Simulation of Behavior* (1974–1980).
- Echelon form, 244.
- Edges of a graph, 3, 9.
- Efficiency, 54–56, 121, 147, 239.
- Eigenvalues, 170.
- EJOR: European Journal of Operational Research* (1977–).
- El-Zanati, Saad Ibrahim (سعد إبراهيم الزناتي), 183, 190, 191.
- Elementary bipartite graphs, 198.
- Elkies, Noam David (נעם דוד אלקיס), 122.
- Ell tetrominoes, 147, 167.
- Elliott, Gordon Lewis, 109.
- Embedding of graphs, *see* SIP.
  - isometric, 29–30, 128–129, 194.
  - strict, *see* ISIP.
- Empty queue, 145.
- Empty set ( $\emptyset$ ), 96, 99, 106, 240.
- Empty (unsatisfiable) clause, 43.
- Empty (unsatisfiable) relation, 152, 213, 252.
- $\text{empty\_q}()$  subroutine, 72, 145.
- Encoding into SAT clauses, 39–42, 46–48, 135, 137, 207–212.
- Endomorphisms, 86, 106, 153.
- Energy, 4.
- Equality relation, 152.
- Equality selection relation, 96, 152.
- Equivalence relations, 103, 129, 154, 157.
- Equivalent relations, 138.
- Ernst, Bruno (= Joannes (= Hans) Adrianus Friedrich de Rijk), 171.
- Error-correcting codes, 175.
- Essential arity, 91–92, 151.
- Essential spectrum, 151.
- Essentially different solutions, 17, 32, 125, 126, 131, 187.
- Eulerian numbers, 219.
- Eulerian trail, 113.
- Even/odd coordinate system, 135, 163.
- Exact cover problem, 31, 39, 114, 130, 136–138, 141, 146, 190.
  - with colors, *see* XCC problem.
- Exponential generating function, 115.
  - bivariate, 165, 241.
- Exponentially large, 218.
- Extended binary tree, 135.
- External nodes, 94.
- Extreme ( $p/q$ ) string, 114.
- “Extreme” XC problem, 64, 145.
- Faces of a polyhedron, 9–10.
- Factorial series, 121.
- Factorization, 185.
  - “Fail early” strategy, 68.
- Failed literals, 216.
- Failure rate based heuristic (FRB), 68.
- Family of sets, 245.
- Fano, Robert Mario, hypergraph, 215.
- Far vertices, 133.
- FC: Forward consistency, 50.
  - versus DC, 51–56, 74–76, 138.
- Feder Bermann, Tomás, 4, 111, 239, 254.
- Feedback vertex set, 135.
- Fibonacci, Leonardo, of Pisa (= Leonardo filio Bonacii Pisano), numbers, 42, 118, 161, 169, 208.
- Field (in algebra), 151.
  - finite, 79, 127.
- Fikes, Richard Earl, 107, 109, 158.
- Fillomino puzzles, 56, 74, 141.
- Filter all domains, 83, 112.
- Finite fields, 79, 127.
- FIRST** array, 17, 26, 120, 125.
- First item of an option, 145, 232.
- First item of every option, 72.
- First moment principle, 196.
- Fischer, Ingrid, 196.
- Fischer, Julien Siegfried, 111.
- Five-letter words of English, 29, 74, 138.
- Fix a variable, *see* Assignment of value to variable.
- Fixit stack, 71–72, 145.
- FLAG** variable, 63, 143.
- Flippo, Maarten Lucas, 112.
- Floating point calculations, 68, 229, 230.
- Flower snark graphs, 131.
- Flower snark line graphs, 44, 136–137.
- Floyd, Robert W, 57, 107.
- Focus, 69, 208.
- Foggia, Pasquale, 200.
- Football teams, 204.
- Forced moves, 65, 67, 73, 78, 144, 227, 235.
- Forcing representations, 48.
- Forests, vi, 38, 104, 128, 130.
- Fork graph, 23.
- Forking, 103–105, 154, 257.

- Forward consistency, 50–54, 58, 69, 71, 73, 109, 138–140, 216, 218, 229.  
 Forward step, 142.  
 Fourfold symmetry, 203, 214.  
 Fourier, Jean Baptiste Joseph, transform, 163.  
 Fractal, 118.  
 FRB heuristic, 68, 144, 230, 234, 238.  
 FRB<sup>†</sup> heuristic, 230, 234.  
 Free boundary, 14, 169, 171.  
 Free trees, 23, 38, 122, 134.  
 Freese, Ralph Stanley, iv, 257.  
 Freuder, Eugene Charles, 55, 110, 239.  
 Friedman, Erich Jay, 223.  
 $fsnark(q)$ , 44, 136–137.  
 Full function, 91–92, 151.  
 Fun, 6, 16, 32, 38, 44, 76.  
 Fun fact, 121.
- $G\mu$ : One gigamem (one billion memory accesses), 73.  
 $G_m^\pi$  (“random” graceful graph), 121.  
 GAD (globally all different) filtering, 34–35, 37, 112, 131–133.  
 Gadgets, 88–91, 96, 98, 150, 154.  
   universal, 90.  
 Gallian, Joseph Anthony, 28.  
 Galois, Évariste, 151.  
   connections, 93, 110.  
*games* graphs, 134.  
 García-Molina, Héctor, 4.  
 Gardner, Martin, 108, 159–160, 168.  
 Garey, Michael Randolph, 194.  
 Gasarch, William Ian, xiii.  
 Gaschnig, John Gary, 109, 225.  
 Gault, Richard Leslie, 246.  
 Gauß (= Gauss), Johann Friderich Carl (= Carl Friedrich), elimination algorithm, 79, 107.  
 GCP (general combinatorial problem), 3, 113.  
 Geek art, xiv, 16, 119, 201–203, 253.  
 Geiger, David Scott, 110.  
 General combinatorial problem, 3, 113.  
 General position, 11, 117, 171, 172.  
 Generalized majority-minority operation, 153.  
 Generalized toruses, 184.  
 Generating all solutions, *see* Visiting all solutions.  
 Generating functions, 5, 113–115, 149, 165, 208, 241, 243, 244.  
 Gent, Ian Philip, 7, 198.  
 Geometric series, 217.  
 Gerdjikov, Stefan Vladomirov (Герджиков, Стефан Владимиров), 222.  
 Gessel, Ira Martin, iv, 162, 165.  
 Giant component, 253.  
 Gigamems (billions of memory accesses), 73.
- Gill, Mukhtiar Kaur (ਮੁਖਤਿਆਰ ਕੌਰ ਗਿਲ), = Acharya, Mukti (ਮੁਖਤ ਅਚਾਰੀਆ), 184.  
 GIP (graph isomorphism problem), 30.  
 Givry, Simon de, 56.  
 Glasgow solver, 201, 203.  
 Global constraints, 6, 107, 112, 156.  
 Global variables, 63.  
 Globally All Different filtering, 34–35, 37, 112, 131–133.  
 Gocht, Stephan, 112.  
 Goedgebeur, Jan, 179.  
 Goldberg, Reuben Garrett Lucius, 89.  
 Golden ratio ( $\phi$ ), 163, 170.  
   as source of “random” data, 116.  
 Golomb, Solomon Wolf, 108, 109, 120, 121.  
 González-Morris, Germán Antonio, 223.  
 Googol ( $10^{100}$ ), 141.  
 Gordon, Daniel Martin, 189, 191.  
 Goriely, Alain Isaïe, 30.  
 Grabarchuk, Serhiy Oleksiyovych (Грабарчук, Сергій Олексійович), 64.  
 Graceful digraphs, 24–28.  
 Graceful labeling, 16–28, 38, 55, 108, 120–128, 178, 182–183.  
 Graceful permutations, 19, 120, 121.  
 Graceful tree conjecture, 24, 28.  
 Graham, Ronald Lewis (葛立恒), 194.  
 Graph coloring, 41, 64, 138, 159, 259.  
 Graph isomorphism problem, 30.  
 Graph labeling, *see* Graceful labeling.  
 Graph representation, 17, 26, 125, 187.  
 Graphs, small, 121, 126.  
 Gray, Frank, code, 122, 182.  
   for 4-vertex graphs, 134.  
 Greatest common divisor, 249.  
 Greedy heuristics, 9.  
 Green, Chris, 220.  
 Grid graphs ( $P_m \square P_n$ ), 33, 123, 124, 131, 132, 138.  
 Ground states, 5.  
 Gruenberger, Fred Joseph, 159.  
 GTC: The graceful tree conjecture, 24, 28.  
 Gyssens, Marc Luc Henri Karel, 110, 111, 241.
- H*-coloring problem, 159.  
 Half edges, 10, 163.  
 Hall, Marshall, Jr., 198.  
 Hall, Philip, 197–198, 251.  
   set, 131–132, 197–198, 200.  
 Halo of a polyomino, 222–223.  
 Hamiltonian cycles and paths, 37, 146, 160.  
 Hamming, Richard Wesley  
   distance, 191–192.  
   embedding, 128.  
   graph, 128.  
 Haralick, Robert Martin, 109.  
 Hardy, Godfrey Harold, 188.  
 Hartert, Renaud Fortunat, 110.  
 Hartke, Stephen Gold, 190.

- Harvey, Warwick, 112.
- Hash code for a graph, 179, 180.
- Haystack problem, 137.
- HC network, 12–13, 117–119.
- HC picture, 10–16, 117–119.
- Head of list, *see* List head.
- Heap-ordered lists, 208.
- Heawood, Percy John, 166.  
graph, 179, 215.
- Height of restricted growth string, 251.
- Hell, Pavol, 159.
- Hello world, 6.
- Hémery, Fred, 67, 68, 109.
- Heuristic function  $h(i)$ , 67, 73, 144, 228.
- Hexadecimal code, extended, 205.
- Hexadecimal constants, v.
- Hexagon, 65, 197.
- Hexagonal grid, 119.
- Hexominoes, 221.
- Hidden variables, 47, 53, 142, 148, 213, 215.
- hide( $i, c$ ), 62–64, 143, 230.
- Hints for  $d$ -cliques, 45, 145, 233.
- Hinz, Andreas Michael, 209.
- Historical notes, 8, 68, 86, 107–111, 158,  
159, 165, 166, 171, 175, 176, 182,  
183, 188, 189, 191, 197, 198, 201,  
204, 207, 213, 215–219, 237, 238, 246,  
251, 252, 254, 255, 257.
- Histoscapes, 9–13, 52–53, 115, 139, 173.
- Ho, Boon Suan (何文軒), 205.
- Homogeneous equations, 190.
- Homomorphisms, 3–4, 30, 111, 113, 147.
- Honaker, Garland Lee, Jr., 75.
- Hopcroft, John Edward, algorithm with  
Karp, ix–xi, 37–38, 131, 135.
- Horizontal and vertical symmetry, 203.
- Horn, Alfred, clauses, 49, 148.
- Horn core algorithm, 49, 53, 139, 216.
- Horn-SAT, 259.
- Horsley, Daniel James, iv.
- House of Graphs, 179, 180.
- Hsu, Derbiau Frank (許德標), 126, 188.
- Huffman, David Albert, 10–16, 107, 171.
- Hugo, Victor Marie, 204.
- Huhn, András Péter, 252.
- Hui, Roger Kwok-Wah (許國華), 238.
- Human Connectome Project, 31.
- Hunt, James Henry Leigh, 264.
- Hwang, Joey Cho Yee (黃祖怡), 218.
- Hyperbindings, 215.
- Hypergraphs, 47, 215.
- Hypersolid pentominoes, 79.
- Icosahedron, 121.
- Idempotent operators, 97, 98, 148,  
246, 253–255.
- Identity relation, 107, 248, 256.
- IJCAI: *International Joint Conference on  
Artificial Intelligence* (1969–), 225.
- ILOG Solver, 110.
- Image of a graph or vertex, 28.
- Implicational constraints, 80–82, 85–86,  
110, 147, 149.
- Impossible objects, 15–16, 119.
- Inactive options, 232.
- Inactive variables, 50, 54, 138.
- Inclusive ancestor in a tree, 129.
- Incompatible options, 54, 69, 77, 144.
- Independent sets, 113, 204.
- Indicator problems, 87–91, 96, 98, 110, 149,  
150, 153, 245–247, 255, 256.
- Induced subgraph isomorphism problem,  
*see* ISIP.
- Inefficiency, 239.
- Inequality relation ( $x \neq y$ ), *see* Nonequality  
relation.
- Initial domains, 1, 33, 36, 132, 142, 154, 200.
- Inner degree, 134.
- Inprocessing, 55.
- Input states, 155.
- Instantiation, *see* Assignment of value  
to variable.
- Intel Xeon computer, 211.
- Interior angles, 168.
- Internal code number, 67.
- Internal nodes, 94.
- Internal numbers, 61.
- Internet, ii, iii, v, vi, ix, 30, 166, 179,  
205, 242.
- Intersecting sets, 149.
- Intersection of planes, 14–15.
- Intersection of relations, 152, 241.
- Intersection of sets, 147.
- Intervals, 112, 143, 185, 219.
- Inv( $O$ ), 95–96, 110.
- Invariant properties of data structure, 62.
- Invariant properties of relations, *see*  
Polymorphisms.
- Inverse arrays, 59, 62.
- Inverse of a permutation, 140.
- Inverse variables, 8.
- Inversions of a permutation, 164.
- Irreflexive binary relation, 3.
- Ising, Ernst, 4.  
configurations, 113.  
model, 4.
- ISIP (induced subgraph isomorphism  
problem), 29–30, 108, 128, 130,  
134–136, 200.
- Isolated vertices, vii, 17, 23, 121, 174, 204.
- Isometric embeddings, 29–30, 128–129, 194.
- Isomorphism between graphs, 30, 113,  
179, 196.
- ITA Software, 238.
- ITEM array, 61–64, 225.
- Items and options, 1, 77.
- Iteratively closed systems, 251.
- ITM field, 61–64, 143.

- $J$ , the all-1 matrix, 155, 241.  
 Jacobi, Carl Gustav Jacob, symbol, 119.  
 Jacquet, Philippe Pierre, 165.  
 Jakovac, Marko, 209.  
 Janson, Carl Svante, 218.  
 Janssen, Philippe, 216.  
 Järvisalo, Matti Juhani, 158.  
 Jeavons, Peter George, iv, 3, 4, 100, 110, 111, 113, 159, 240, 241, 246, 247, 254.  
 Jégou, Philippe Rémy Paul, 216.  
 Jelliss, George Peter, 75.  
 Joachimsthal, Ferdinand, 166.  
 Johnson, David Stifler, 194.  
 Johnson, Samuel, 262.  
 Join graph, 216.  
 Join of relations ( $R \bowtie S$ ), 4, 96, 241.  
   lossless, 48.  
 Joke, 191.  
 Junctions, 10–14, 117–119, 163, 171–173.  
  
 $k$ -ary constraints, 109, 142, 158, 225.  
   regarded as binary constraints, 47, 137, 139; *see also* Hidden variables.  
 $k$ -ary operations, 86, 94.  
 $k$ -center problem, 146.  
 $k$ -dimensional matching, 159.  
 $k$ -tuples, 147.  
 $k$ -wise projections, 99, 154.  
 $K_{2,1,1}$ , 191.  
 $K_{3,3}$ , 168.  
 $K_{m,n}$  (complete bipartite graph), 123–124, 180, 188, 242.  
 $\vec{K}_{m,n}$  (oriented complete bigraph), 126, 206.  
 $K_n$  (complete graph), 3, 19, 108, 120–122, 124, 195; *see also* Cliques.  
 $K\mu$ : One kilomem (one thousand memory accesses), 73.  
 Kaluzhnin, Lev Arkadievich (Калужнин, Лев Аркадьевич), 86, 110.  
 Kane, Daniel Mertz, 170.  
 Kaneko, Masanobu (金子昌信), 241.  
 Kaplansky, Irving, 241.  
 Karnaugh, Maurice, map, 188.  
 Karp, Richard Manning, algorithm with Hopcroft, ix–xi, 37–38, 131, 135.  
 Kazda, Alexandr, 252.  
 KC graphs ( $K_n \square C_r$ ), 22, 121, 176.  
 Kearnes, Keith Alan, 255.  
 Keith, Michael, 75.  
 Kenig, Michael Joseph, Jr., 183.  
 Kilomems (thousands of memory accesses), 73.  
 Kingwise adjacent, 222.  
 Kingwise connected, 85, 142, 241.  
 Kirkman, Thomas Penyngton, 191.  
 Kirousis, Lefteris Miltiades (Κυρούσης, Ελευθέριος Μιλτιάδης), 110, 119, 173.  
 Kiss, Emil W., 252.  
 Kissat solver, 43, 211.  
  
 Kite designs, 44.  
 Klavžar, Sandi, 207, 209.  
 Kløve, Torleiv, 175.  
 Knight graph ( $N_n$ ), 133–134, 146, 234, 237.  
   embedded in queen graph, 134.  
 Knight of Holiness, 1.  
 Knight plus queen (superqueen), 157.  
 Knight's grids, 133.  
 Knight's tours, 75–76, 146.  
 Knights and queens, nonattacking, 146–147.  
 Knuth, Donald Ervin (高德纳), i, v, 38, 43, 55, 61, 75, 111, 146, 163, 165, 166, 177, 179–182, 206, 208, 209, 220, 225, 227, 229, 232, 242.  
 Knuth, Nancy Jill Carter (高精兰), iv.  
 Kobouzu17, 220.  
 König, Dénes, 198.  
 Koplowitz, Herbert Philip, 160.  
 Kotěšovec, Václav, 261.  
 Kotov, Viktor Nikolaevich (Котов, Виктор Николаевич), 86, 110.  
 Kotzig, Anton, 24, 175, 178, 179, 182, 183.  
 Kozik, Marcin, 251.  
 KP graphs ( $K_n \square P_r$ ), 20–22, 120, 121, 123, 176.  
 Krasner, Marc Isaacovich (Краснер, Марк Исаакович), 186.  
 Krokhin, Andrei (Крохин, Андрей Анатольевич), 111, 254, 255.  
 Kumudakshi Suvarna (කුමුදාක්ෂි ජුවර්ණ), 189.  
 Kurchan, Rodolfo Marcelo, 223.  
  
 $L(2, 1)$  labelings, *see* Radio colorings.  
 $L_{m,n}$  (lollipop graphs), 180, 189, 204.  
 $L(r, s)$  lattice of partitions, 197.  
 Labeled graph embedding, 130.  
 LAD filtering, 33–37, 131, 132, 200.  
 Ladner, Richard Emil, 97.  
 Lagrange, Joseph Louis (= Giuseppe Lodovico Lagrangia = Giuseppe Ludovico (= Luigi) De la Grange Tournier), 166.  
 Langford, Charles Dudley, pairs, 65.  
 Laplace (= de la Place), Pierre Simon, Marquis de, limit constants, 165.  
 Larrosa Bondia, Francisco Javier, 134.  
 Latin squares, 154, 209.  
 Lattice, 197; *see also* Boolean lattice, Semilattice.  
 Lax Maróti pair, 98, 153.  
 Lax Siggers term, 153, 253.  
 Lax weak near unanimity operation, 97, 152–153.  
 Laxity, 153.  
 Lazy updates, 148.  
 le Clément de Saint-Marcq, Vianney Emmanuel Adrien Martin Marie Joseph, 110.  
 Leader, vii, viii.  
 Leaper, 204, 261.

- Learning, 108.
- Least common multiple, 249.
- Least upper bound, 83, 148.
- Lecoutre, Christophe, 67, 68, 109–111, 214, 216, 223.
- Lee, Chester Chi Yuan (李始元) = Chi Lee (李濟), distance, 27.
- Lee, Jimmy Ho-Man (李浩文), 112.
- LEN fields, 208.
- Lenz, Wilhelm, 4.
- Les Misérables*, 204.
- Level, 144, 227, 228.
- Levels, 66.
- Lexicographic order, 106, 115, 154, 157, 163, 259.
- Li, Hongbo (李宏博), 68.
- Li, Zhanshan (李占山), 68.
- Lichtenstein, David Isaac Joseph, 173.
- LIFE magazine, 263.
- Line graphs, 44, 199, 209.
- Line labeling problem, 9–16, 38, 52, 107–109, 115–119, 139.
- Linear equations and linear inequalities, 119.
- Linear equations mod  $d$ , 79, 86, 149.
- Linear independence, 190, 244.
- Linear programming, 173.
- Linear time, 80–82, 173, 239.
- Linked lists, 17, 26, 82, 142, 145, 147.
- LIPics: Leibniz International Proceedings in Informatics* (2008–).
- List head, 59, 147, 225.
- Literals, 1, 90, 158.
- L0 array, 17, 23, 26, 120, 122, 125, 189.
- Lo Shu magic square, 203.
- Lobjois, Lionel Jean, 56.
- L0C field, 61–64, 143.
- Local constraints versus global constraints, 107, 112.
- Log encoding, 2, 39–42, 44, 46, 136, 137, 207–211.
- Lollipop graphs ( $L_{m,n}$ ), 180, 189, 204.
- Lonesum matrices, 242.
- Longest cycle, 175.
- Longest path, 120.
- Lossless join dependencies, 48.
- Lovász, László, 198.
- Lower bounds for resolution, 136.
- Loyd, Samuel, 237.
- Lucas, François Édouard Anatole, numbers, 118, 171, 208.
- Luks, Eugene Michael, 196.
- LWNU: Lax weak near unanimity, 97, 152–153.
- Lyndon, Roger Conant, words, *see* Prime strings.
- $m$ -fold composition, 248.
- $M\mu$ : One megamem (one million memory accesses), 73.
- $M\nu$ : One meganode (one million nodes), 73.
- $\binom{[m]}{k}$  (combinations as  $k$ -tuples), 99, 256.
- Mackworth, Alan Keith, 48, 109, 158.
- MacMahon, Percy Alexander, 64, 65.
- Magic masks, 90.
- Magic squares, 203.
- Magical  $4 \times 4$  torus, 188.
- Magnetization, 113.
- Mahéo, Maryvonne, 183.
- Maintaining domain consistency, 52–56, 69–73, 110.
- Majority operator, 100, 244.
- Majorization, 196–197.
- Maltsev (= Mal'cev), Anatoly Ivanovich (Мальцев, Анатолий Иванович), 254–255, 257.
- polymorphisms, 103–105, 149, 151, 153–155, 254–255, 257–259.
- stubs, 104, 154–155.
- Marijnissen, Imko Cornelis Wouter Maria, 112.
- Marković, Petar (Марковић, Петар), 255.
- Maróti, Miklós, 111, 251–252.
- pairs, 98, 153, 155.
- pairs, double, 255.
- Matching, 38.
- bipartite, ix, 33–34, 37, 131–132, 135.
- $k$ -dimensional, 159.
- MATE** table, 174.
- Mathematics versus computer science, 239.
- Matoušek, Jiří, 196.
- Matrices of 0s and 1s, 80, 84, 148–149, 239–240.
- Matrix coordinates versus Cartesian coordinates, 162.
- Matrix representation of binary relation, 80, 84, 148–149, 239–240.
- Matula, David William, 38, 134, 135.
- Max-closed constraints, 82–83, 89, 95, 99, 110, 147–149.
- Maximum independent set in bipartite graph, xi.
- MaxSAT: Satisfy as many clauses as possible, 5.
- mc.revise( $c, v$ ), 83.
- MCC problem: multiple covering with colors, 38, 76–79, 114, 146, 147, 179, 234, 238.
- benchmarks for, v, 78–79.
- MCC solver, v, 235–236.
- McCreesh, Ciaran Arthur, iv, 35, 112, 200, 201, 203.
- McGill, John, Jr., 175.
- McGinty, Abigail, 157.
- McGregor, James Jardine, 109, 201.
- McIlree, Matthew John, 112.
- McKenzie, Ralph Nelson Whitfield, 111, 251, 252, 255.
- Mebane, Palmer Croasdale, 221.

- Median ( $\langle xyz \rangle$ ), v, 85, 95, 100, 138, 148, 152, 213, 241, 246, 255.
- Median algebra, 252.
- Median-closed constraints, 148.
- Median labels, 193.
- Megamems (millions of memory accesses), 73.
- Meganodes (millions of nodes), 73.
- Mélot, Hadrien, 179.
- Mems (accesses to 64-bit words), 64, 163, 227.  
per node, 74.
- Mercer, John Herndon, 258.
- Meromorphic function, 166.
- Mézard, Marc Jean Marcel, 5, 159.
- Miguel, Ian James, 198.
- Milton, John, 1.
- Milutinović, Uroš, 209.
- Min-and-max-closed constraints, 149.
- Min-closed constraints, 147, 149, 240.
- Min operation, 99.
- Minimal non-Hamming graphs, 128, 193.
- Minimal polynomial, 249.
- Minimal stub, 100.
- Minimum element of domain, 142.
- Minimum remaining values heuristic,  
*see* MRV heuristic.
- MINIZINC format, 111–112.
- Miracle, 18, 122.
- Mirror image, 7, 117, 160.
- Mitchell, David Geoffrey, 218.
- MIX, 251.
- Mixed-radix representation, 177.
- ML language, 86.
- MMIX computer, ii, 58, 60, 133.
- Mnemonic, 255.
- MNH: Minimal non-Hamming, 128, 193.
- Möbius, August Ferdinand, polynomial, 186.
- Model theory, 2.
- Modstep problem, 55–56, 74, 140.
- Moiseev, Stanislav Vladimirovich (Моисеев, Станислав Владимирович), 252.
- Molecules, 130.
- Monoforkings, 154, 257.
- Monominoes, 221.
- Monotone functions, 83.  
Boolean, 87, 240, 245, 246.  
self-dual Boolean, 95, 254.
- Montanari, Andrea, 5, 159.
- Montanari, Ugo, 107, 155, 213.
- Montgomery, Richard Harford, 28, 127.
- Monus operation ( $x \dot{-} y$ ), 78.
- Moon, John Wesley, 204.
- Moshkovitz, Dana Aaronson (מושקוביץ דנה), 111.
- Motif graph, 36.
- Motley dissections, 79.
- MRV heuristic (minimum remaining values), 54, 63, 65–67, 73, 78, 136, 139, 140, 143, 144, 204, 208, 227, 230, 234.
- Muchnik, Albert Abramovich (Мучник, Альберт Абрамович), 252.
- Multiple covering with colors, *see* MCC problem.
- Multiplicities, 235.
- Multisets, 8, 130, 131.
- Multivalued encoding, 40–42, 44, 207–212.
- Multivalued logic, 110.
- Muñiz, Alexandre Owen, 236.
- Music, 74.
- Mutually intersecting sets, 149.
- Mycielski, Jan, graph  $M_4$ , 65, 135.
- $n$ -cube, 122, 124.
- $n$  queens problem, 48, 109, 138, 146, 157.
- $n$  superqueens problem, 157.
- $n$ -tone rows, 74.
- $(n, k)$ -universal sets, 257.
- Name, choosing a technical, 86.
- NAME array, 17, 26, 120, 125.
- NAND relation, 149.
- Narayana, Tadepalli Venkata (తాడేపల్లి వెంకట నారాయణ), numbers, 243.
- Nārāyaṇa Paṇḍita, son of Nṛsiṃha (नारायण पण्डित, नृसिंहस्य पुत्रः), cows sequence, 161.
- Natural join, 4.
- Near  $\alpha$ -labeling, *see* Ordered graceful labeling.
- Near unanimity polymorphisms, 97–103, 152–154.
- Near vertices, 133.
- Nearest common ancestor, 193.
- Negative table constraints, 135, 146.  
versus positive table constraints, 213.
- Nešetřil, Jaroslav, 159.
- Nested parentheses, 143, 156.
- Networks, 108.
- NEXT array, 26, 125.
- NEXTH and NEXTL arrays, 17, 120, 187.
- Nguyen, Alain, 9.
- Niemelä, Ilkka Niilo Fredrik, 158.
- Nightingale, Peter William, 198.
- Niven, Todd Matthew, 251.
- No-three-in-a-line, 79, 146.
- No-touch puzzle, 160.
- NODE array, 61–64.
- Nodes, 73.
- Nogoods, 1, 46, 108, 112.
- Nonattacking knights and queens, 146–147.
- Nonattacking queens, *see*  $n$  queens problem.
- Noncommuting variables, 169.
- Nondeterministic automaton, 155.
- Nonequality relation, 3, 40–41, 91, 150, 248.
- Nonominoes, 74.
- Nonsharp preference heuristic, 79.



- NOR relation, 245.
- Nordström, Karl Jakob, 112.
- Not all equal, 150.
- Not equality, *see* Nonequality relation.
- Notational conventions, v.
  - $X \nearrow$  and  $Y \swarrow$  (polarities), 92.
  - $G \subseteq H$  ( $G$  is subgraph of  $H$ ), 30.
  - $G \sqsubseteq H$  ( $G$  is induced subgraph of  $H$ ), 30.
  - $r \star s$  (star composition), 255.
  - $t[x, y]$  ( $t$ -of-the-way from  $x$  to  $y$ ), 254.
  - $[x_1 \dots x_k]$  (the value of  $f(x_1, \dots, x_k)$  in a polymorphism), 86–87.
  - $\langle xyz \rangle$  (median), v.
  - $o \parallel o'$  (options are compatible), 54.
  - $x_1 \dots x_k \parallel x'_1 \dots x'_k$  (tuples are unlike), 113.
- Nouguier, Bernard, 216.
- NP-complete, 90–91, 96–98, 111, 119, 130, 159, 253.
- NP-hard, 38.
- NP-intermediate, 97.
- NU: Near unanimity, 97–103, 152–154.
- Null matrix, 241.
- Nullary (constant) operation, 2, 94, 113, 250.
- $O$ , the all-0 matrix, 155, 241.
- O’Beirne, Thomas Hay, 160.
- O’Donnell, Ryan William, 255.
- OA<sub>CTIVE</sub> pointer, 63, 231.
- Octahedron, 9, 178.
- Octal notation, 116, 191.
- Octants, 116.
- Octominoes, 142.
- OEIS®: The On-Line Encyclopedia of Integer Sequences® ([oeis.org](http://oeis.org)), 177, 178, 181, 182, 184, 188, 191, 219, 237, 241, 243, 245, 252.
- Ollis, Matthew Anthony, 175.
- One-in-three satisfiability, 90.
- One-to-one function, 261.
- Online algorithms, 99, 102, 155.
- Online resources, *see* Downloadable programs and data.
- Operation: A function from  $D^k$  to  $D$ , 86, 94.
- Operator: *see* Operation, 94.
- Operator norm, 170.
- Opršal, Jakub, 254.
- $\text{opt}(x, x')$ , 227, 235.
- $\text{opt\_out}(o)$ , 72, 144–145, 234.
- Option list, 61, 62.
- Options, 1, 2, 54, 77.
- OR (bitwise OR), 201.
- OR relation, 86–88, 149.
- Orchard pattern, 237.
- Order encoding, 2, 40–42, 44–46, 207.
- Order-reversing correspondence, 93, 95.
- Ordered graceful labeling, 123, 124.
- Ordered graceful rainbow labeling, 191.
- Ore, Øystein, 93.
- Organ-pipe order, 65, 229.
  - inverse, 170.
- Orientations of edges, 27.
- Oriented complete bipartite graph, 126.
- Oriented cycle, vi, 26, 126, 150, 159.
- Oriented forest, vi.
- Oriented path, 26, 82, 126.
- Origami, 10.
- Orthographic projection, 11.
- Orthohedral polyhedra, 173.
- Östergård, Patric Ralf Johan, 190.
- Otter, Richard Robert, 23.
- Output states, 155.
- Overflow, floating point, 229.
- Overflow of stamps, 145.
- $\varnothing$  (tautology), 211.
- $P \neq \text{NP}$  (?), 97.
- #P-complete, 259.
- $P_n$  (path of order  $n$ ), 19–20, 120, 121, 124, 180.
- $P_n^\rightarrow$  (oriented path of order  $n$ ), 26, 126, 188.
- $(p, q)$ -leaper, 204.
- $(p/q)$ -string, 114.
- Pack a given shape, 146.
- Pahl, Felix Andreas (= [joriki](http://joriki)), 237.
- Pairwise consistency, 216.
- Pairwise ordering trick, 141, 160, 196.
- Palindromes, 161, 260.
- Palindromials, 125.
- Papadimitriou, Christos Harilaos (Παπαδημητρίου, Χρίστος Χαρίλάου), 119, 173, 196.
- Parades, 242.
- Parallomino graph, 123, 243.
- Parameters, tuning, 43–44.
- ParamILS, 43.
- Parent vertex, vii, 128, 134.
- Parity, 22, 104, 126, 183, 184, 213, 221, 230, 234, 258.
- Partial assignment, 84, 260.
- Partial ordering, 83, 148.
  - of multisets, 197.
- Partition function, 5, 113.
- Partitioning into multisets, 130.
- Partitions of integers, 197.
- Partridge puzzle, 78, 79.
- Pasting graphs together, 44.
- Path consistency, 53, 84–85, 107, 139, 148, 241.
- Path of order  $n$  ( $P_n$ ), 19–20, 120, 121, 124, 180.
- Patterns in permutations, 219.
- Paw graph, 23, 180, 204.
- Pearl, Judea (יהודה פֶּרֶל), 215.
- Pegg, Edward Taylor, Jr., 178.
- Penrose, Lionel Sharples, 15.

- Penrose, Roger, 15.  
    $n$ -gon, 119.  
   triangle, 15–16.  
 Pentacubes, 65.  
 Pentominoes, 64–66, 78, 79, 124, 221.  
 Perez, Guillaume Didier, 110.  
 Perfect matching, 198.  
 Perfect Packing puzzle, 79.  
 Permutation matrix, 241.  
 Permutations, 17, 59, 132, 186, 211.  
   of a multiset, 8.  
 Perron, Laurent Frédéric, 110.  
 PERT: Project Evaluation and Review Technique, 83.  
 Petersen, Julius Peter Christian, graph, 129, 132, 135.  
 Petrie, Charles Joseph, Jr., 213.  
 Petrie, Karen Elizabeth Jefferson, 176.  
 Phi ( $\phi$ ), as source of “random” data, 116.  
 Philippsen, Michael, 196.  
 Photographs, 9.  
 Physics (statistical mechanics), 4–5.  
 Pi ( $\pi$ ), as source of “random” data, 18, 59, 60, 74, 76, 90, 91, 100, 116, 121, 124, 141, 150, 164.  
 Pigeonhole principle, 198, 211.  
 Pigeonhole problem, 229.  
 Piles of polyominoes, 78, 236.  
 Pinched Sierpiński gasket graph, xiv, 42–44.  
 Pinched Sierpiński simplex graph, 137.  
 Pinsker, Michael, 254.  
 Pipeline of a computer, 227.  
 Pippenger, Nicholas John, 251.  
 Pixel, 107, 146.  
 Pixley, Alden French, 100, 251, 252.  
 Planar graphs, 11, 117, 172, 173, 177.  
 Plane in three dimensions, 167.  
 Plane partitions, 114.  
 Plummer, Michael David, 198.  
 Pointwise product matrix, 239.  
 Poirot, Hercule, 157.  
 Pokrovskiy, Alexey Alexeevich (Покровский, Алексей Алексеевич), 28, 127.  
 Pol( $\Gamma$ ), 95–97, 110.  
 Polarities, 92–95, 150.  
 Pollak, Henry Otto, 194.  
 Poly-Bernoulli numbers, 241.  
 Polyhedra, 114; *see also* 3VP.  
 Polymorphisms, 86–89, 110, 149–155, 241.  
   exploitation of, 99–106.  
 Polynomial factorization, 124.  
 Polynomial time, 84, 85, 99, 103, 128, 149, 153, 196, 239, 258, 259.  
   delay, 106, 155.  
 Polyominoes, 65, 141, 142, 146, 147, 243.  
   piling, 236.  
 Population count ( $\nu x$ ), sideways sum, v, 58, 161, 193, 247.  
 POS field, 61–64.  
 Pöschel, Reinhard, 110.  
 Positive table constraints, 135.  
   versus negative table constraints, 213.  
 Post, Emil Leon, 95, 251, 256.  
 Potent constraints, 90–92, 95, 246.  
 Preclusion, 109.  
 Preclusion clauses, 46, 212.  
 Predecessors of a vertex, 187.  
 Predicate, *see* Relation, 2.  
 Prefix encoding, 40–41, 45, 135, 136, 207–211, 129.  
 Preorder of a tree, 129, 189.  
 Preprocessing, 55, 65, 74, 75, 211, 216.  
 Prestwich, Steven David, 41, 207.  
 Primary item, 2, 54, 62–64, 77–78, 145, 208–209, 253.  
 Prime factorization, 97, 249.  
 Prime queen attacking problem, 74–76, 146–147.  
 Prime strings, 156.  
 Primitive polynomial modulo  $p$ , 127.  
 Principal ideals, 249.  
 Product of relations ( $R' \otimes R''$ ), 96, 152, 252.  
 Programming languages, 86.  
 Projection from 3D to 2D, 11, 117.  
 Projection function ( $\pi_j$ ), 88, 90, 95, 153, 213, 246–250, 259.  
 Projection of a relation ( $\Pi_\sigma R$ ), 48, 96, 99–101, 152, 241.  
 Projective plane, 127.  
 Proof, computer-generated, 22.  
 Proof logging, 112, 157.  
 Propagation of constraints, 48–56, 108, 112.  
 Prosser, Patrick Thomas, iv, 35, 200.  
 Prover–Delayer game, 218.  
 Puget, Jean-François, 176.  
 Punnim, Narong (ณรงค์ ปันนัณ), 191.  
 Pure literals, 212.  
 Pure vertices, 44–45, 137, 209.  
 Purging, 70–71.  
 Purified items, 231, 233.  
 Puzzles, 6, 17, 64–65, 74, 78–79, 116, 120, 141, 142, 147.  
 Pyramid, 9.  
  
 $q$ -nomial coefficients ( $\binom{n}{k}_q$ ), 197.  
 Quasi-inverse function, 153.  
 Quasi-WNU: LWNNU, 252.  
 Quasigroup, 154.  
 Quaternary (4-ary) constraint, 2, 162, 164.  
 Quaternary discriminator, 251.  
 Queen graph ( $Q_n$ ), 64, 134, 137, 234.  
 Queens, attacking, *see* Prime queen attacking problem.  
 Queens, dominating, *see* 5-queens domination problem.  
 Queens, nonattacking, *see*  $n$  queens problem.

- Queens and knights, nonattacking, 146–147.
- Queens-and-knights problem, 67, 144, 237.
- Queue (FIFO), 52–53, 71, 109, 145, 205–206.  
versus stack (LIFO), 144.
- Quicksort, 224.
- Quimper, Claude-Guy, 201.
- $\mathbb{R}, \overline{\mathbb{R}}$  relations, 245.
- $\langle R \rangle_f$  (completion of  $R$  with respect to  $f$ ), 99, 101.
- $r$ -regular graphs, 121.
- Radio colorings, 65, 135, 159.
- Radio link frequency assignment problem, 56.
- Radix list sort, 233.
- Rainbow graceful labelings, 28, 108, 127–128.
- Rainbow path problem, 39, 205.
- Rampon, Jean-Xavier Fernand, 112.
- Random circuit, 140.
- Random CSP, 55, 140.
- Random free tree, 135.
- Random graceful graph, 121.
- Random graph, 130, 188.
- Random satisfiability, 5.
- Randomization, 154, 179.
- Randomized algorithms, 18, 122.
- Range versus domain, 113.
- Ranulac, Britt (*‘canular’* pseudonym of Frédéric Marty), 186.
- Rational numbers, 151.
- Reachability, 3.
- Real numbers, 125, 249.
- Real-world data, 9.
- Realizable HC networks, 15–16, 116, 117, 168, 173.
- Recurrence, 140, 149, 165, 176, 177, 208, 219.  
binary, 122.
- Recursive description, 240.
- Recursive procedure, 50, 77, 205.
- Reduced encoding, 40–42, 44, 135–136, 207–212.
- Reduced matrix, 148.
- Redundant constraints, 8.
- Reentrant knight’s tour, *see* Closed knight’s tour.
- Reflection of a solution, 7, 117, 156, 157.
- Reflexive binary relation, 148.
- Refutation length, 136.
- Régin, Jean-Charles Pierre Henri, 109, 110, 198.
- Relation: A property that holds for certain tuples of elements, 1–4, 86, 150.  
encoding as SAT clauses, 39–42, 46–48, 135, 137, 207–212.  
encoding as XC, XCC, or MCC options, 38–39.  
table constraints, 135, 146, 213.  
tractable, 79–85.
- Relational clones, 96, 152.
- Relational databases, 4, 96, 216.
- Relational structures, 2, 111.
- Reluctant doubling, 182.
- Removable value, 131.
- Removed option, 71.
- Representation of graphs and digraphs, 17, 26, 27.
- Representation of sets, 56–60, 147.
- Residual supports, 224.
- Resolution, 137.
- Restarts, 112, 182, 203.
- Restricted growth strings, 128, 156, 251.
- Restriction problem, 111.
- Reutersvärd, Oscar Gustav Adolf, 14, 16, 119, 171.
- Reversal of permutation, 20, 219.
- Reverse model, 19.
- Reversible memory, 57.
- Reversible sparse bitsets, 60, 110, 142.
- revise( $c, v$ ), 52–53, 83, 109.
- Revolving-door Gray code for 4-vertex graphs, 134.
- RGS: A restricted growth string, 128, 156, 251.
- Richardson, Samuel, 262.
- Rim representation, 162.
- Riordan, John Francis, 219, 241.
- Rivest, Ronald Linn, clauses  $R'$ , 113.
- RLFAP: Radio link frequency assignment problem, 56.
- Rokicki, Tomas Gerhard, 18, 181, 182, 203, 204.
- Roman numerals, 166.
- Romov, Boris Aronovich (Ромов, Борис Аронович), 86, 110.
- Rook graph, 134.
- Rookwise connected, 141.
- Root of a tree, 134–135.
- Root of unity, 185.
- Rooted graceful labeling, 121, 181, 182.
- Rosa, Alexander, 24, 108, 183.
- Rosenberg, Ivo Jiří (= George), 246.
- Rossi, Francesca, 111, 213.
- Rotating 45°, 221.
- Rounding error, 15.
- Roussel, Olivier Michel Joseph, 111.
- Row convex constraints, 84–85, 110, 148–149.
- Row sums, 242.
- Rows of musical tones, 74.
- Ruler function  $\rho x, v$ , 58, 183, 224.
- Running time, 43–45, 54, 64, 65, 69, 78, 207–211.

- $S_c$  (scope of constraint  $c$ ), 52.  
 $(s, t)$ -caterpillar, 124.  
 Saarinen, Tapio, 220.  
 Sabin, Daniel, 55, 110, 239.  
 SADD (sideways addition), 58.  
 Safra, Shmuel (= Muli; שמואל ספרא), 111.  
 Saggese, Alessia, 200.  
 Saint-Marcq, Vianney Emmanuel  
     Adrien Martin Marie Joseph, le  
     Clément de, 110.  
 Saïs, Lakhdar ( $\odot \cdot \leq \odot \parallel \cdot \times \mathbb{E} \cdot \odot$ , لخضر سايس),  
     67, 68.  
 Sakinaga, Waku (崎永惑), 219.  
 Sansone, Carlo, 200.  
 Sapozhnikov, Artem Vasilievich  
     (Сапожников, Артём Васильевич),  
     217.  
 SAT: The satisfiability problem, 1–2, 137.  
     represented as CSP, 1, 113.  
 SAT solvers, 1, 39–48, 137, 210.  
 Scene analysis, 9.  
 Schaefer, Thomas Jerome, 97.  
 Schaus, Pierre Jean Philippe Daniel, 110.  
 Schiex, Thomas Daniel, 56.  
 Schulte, Christian, 112.  
 Scope of a constraint, 52–53, 239.  
 Search tree, 66–67, 144.  
     size of, 136, 218.  
 Second moment principle, 196.  
 SECOND pointer, 62–64, 73, 143, 226–227.  
 Secondary item, 2, 54, 63, 77, 120, 160, 161,  
     186, 196, 205, 208–209, 213, 217, 219,  
     220, 225, 228–230, 234, 237, 238, 246.  
 Seed of a stub, 99, 104.  
 Selection relation, 96.  
 Self-converse digraphs, 25, 187–190.  
 Self-dual Boolean functions, 95, 150,  
     245, 254.  
 Self-dual ternary functions, 255.  
 Self-transpose latin square, 209.  
 Semilattice, 83–84, 148.  
 Senderov, Valery Anatol'evich (Сендеров,  
     Валерий Анатольевич), 186.  
 Sentinel link, vii.  
 SET array, 61–64.  
 Set inclusion digraph, 24, 125–126.  
 Set partitions, 103, 251, 257.  
 Shadows in scenes, 107.  
 Sharp preference heuristic, 79, 143, 160, 217.  
 Sheppard, David Allen, 23, 184.  
 Shifting, 147.  
 Shortcut relation, 90, 150, 246, 253.  
 Shortest cycles, 108.  
 Shortest distance, 29, 128–129.  
 Shortest paths, 107.  
 Siblings, 62, 228, 236.  
 Sicherman, George Leprechaun, iv.  
 Sideways addition, 58, 161, 193, 247.  
 Sidorov, Konstantin (Сидоров, Константин  
     Сергеевич), 112.  
 Sierpiński, Waclaw Franciszek, 41–42.  
     gasket graph, 41–42, 135–136.  
     gasket graph, pinched, xiv, 42–44,  
         135–136.  
     graphs, proper, 136–137.  
     simplex graphs, 44–45, 136–137.  
     tetrahedron graph, 44–45, 136–137.  
     tetrahedron graph, augmented, 45, 137.  
     triangle graph, 41–42, 135–136.  
 Siggers, Mark Halvor, 255.  
     term, 153.  
 Signed skeleton of a polyhedron, 116.  
 Similar relational structures, 3.  
 Simonis, Helmut Matthias, 8.  
*simplex* graphs, 131.  
 Simultaneous linear equations, 79.  
 Singer, James, 127.  
 Singleton domain consistency, 53, 139.  
 Singly linked lists, 145, 147.  
 Sink component, vii.  
 Sink vertex, 82, 140, 147, 248, 252.  
 SIP: Subgraph isomorphism problem, 3,  
     30–38, 108–109, 128–135, 200.  
     survey, 37.  
**SIZE** field, 61–64, 77–78.  
 Skeleton of a polyhedron, 116, 123, 124.  
**SLACK** field, 77–78.  
 Slim nonominoes, 74.  
 Slot in assembly line, 8, 160–161.  
 Slow growth permutations, 56, 74,  
     75, 140–141.  
 SLUR algorithm, 216.  
 Small-and-slim nonominoes, 74.  
 Smile of order  $n$ , 118.  
 Smith, Barbara Mary, 176.  
 Smith, Henry John Stephen, normal  
     form, 244.  
 Smits, Jeff, 112.  
 Snake path: An induced path, 29, 64.  
 Snevily, Hunter Saint Clair, 184.  
 Snyder, Thomas Marshall, 223.  
 Solid pentominoes, 65.  
 Solnon, Christine Roseline Michèle, iv, 9, 33,  
     37, 61, 110, 134, 144, 195, 196, 203, 204.  
 Solution set, 102, 106, 149, 155, 156.  
 Sorting, 72, 131, 133, 233.  
 Source vertices, 140, 248, 253.  
 Spacer nodes, 61–62, 145, 230.  
 Spanning trees, 122, 221.  
 Sparse bitset, 60, 110, 142.  
 Sparse-set representation, 59–64, 72–73,  
     110, 142, 181, 200, 227.  
     deletion, 59, 227, 234.  
 Spectrum of a clone, 151, 152, 245.  
 Spenser, Edmund, 1.  
 Sphinx, 9–11, 14.  
 Spin, 4.

- Spivak, Alexander Vasilievich (Спивак, Александр Васильевич), 186.
- SPJ-algebra: select, project, join, 96.
- Square dissection puzzle, 74.
- Square torus, 164.
- Squarefree polynomials, 249.
- Squashed boxes, 119, 172, 194.
- Squashing functions, 153, 254.
- Stabilizer of a predicate, 110.
- Stack, 57, 63, 71–73, 144, 214.
- Stage stamp, 233.
- Stages, 66–67, 144, 227, 228.
- Staircase polygons, 243.
- Stallman, Richard Matthew, 108.
- Stamping, 52–53, 57–58, 72, 73, 109, 142, 144, 145, 233.
- Standard line labeling, 13–14, 118–119.
- Stanford GraphBase, ii, 29, 134, 191.
- Stanley, Richard Peter, 219.
- Stappers, Filip Jan Jos, iv, 43, 77, 146, 177, 186, 188, 189, 205, 208, 223, 244, 261.
- Star composition, 255.
- Star graph, 19.
- Star of David pattern, 171.
- State variable: A variable that helps to govern a process, 57.
- States, 155–156.
- Statistical mechanics, 4–5.
- Stirling, James, partition numbers, 242, 251, 253.
- Strachey, Christopher, 86.
- Stravinsky, Igor Fëdorovich (Стравинский, Игорь Фёдорович), 1.
- Strict embedding, 29–30, 128–135, 192, 200, 201.
- Strict inequality, 173.
- String polynomials, 169.
- Strong components, vi–ix, 81–82, 131–132, 152, 198.
- Strong prime queen attacking, 74–76, 146–147.
- Strongly NP-complete, 194.
- Strongly realizable, 15–16, 173.
- Strongly rigid constraints, 246.
- Structured constraint enforcement, 162.
- Stubs, 100–105, 111, 154–155, 259.
- Stuckey, Peter James, iv, 111, 112.
- Subcube labels, 129.
- Subdag isomorphism, 38.
- Subgraph isomorphism, *see* SIP, ISIP.
- subsets* graphs, 129.
- Subtree isomorphism, 38, 134–135.
- Successors of a vertex, 26, 125.
- Sudakov, Benjamin (= Benny; בנימין סודקוב), 28, 127.
- Sudoku, 17, 34, 65, 141.
- Sugihara, Kokichi (杉原厚吉), iv, 171, 173.
- Superachievers, 253, 257.
- Superexponential, 23, 122.
- Superposition of functions, 94.
- Superqueen, 157.
- Supplemental labels, 35–36, 108, 132–133, 196.
- edge labels, 36, 132–133.
- graphs, 37, 132–133.
- pair labels, 36, 132–133.
- Support encoding into SAT, 40–42, 44, 47, 49, 137, 138, 207–212, 215.
- Support matrix, 70–72, 144.
- Support vectors, 139–140.
- Supported words, 192.
- Supporting tuples, 51, 109, 140, 239.
- Suresh Singh, Gnanadas (சுரேஷ் சிங்), 177.
- Surprise, 29, 44, 45, 55, 56, 68, 74, 75, 209, 256.
- not a, 89, 90.
- Sussenguth, Edward Henry, Jr., 108.
- Sussman, Gerald Jay, 108.
- Symmetric binary relation, 3, 249.
- Symmetric Boolean ternary relations, 88–90, 150.
- Symmetric digraphs, 27.
- Symmetrical solutions only, 203.
- Symmetry breaking (removal), 7, 17, 21, 23, 25, 54, 65, 74, 114, 120, 125, 137, 156, 160, 162, 176, 220, 223, 235, 236, 244, 246, 254, 255, 260, 261.
- System of distinct representatives, 197.
- Szeider, Stefan Hans, 205.
- Szilassi, Lajos, polyhedron, 9–11, 14, 114, 116, 117.
- Szymanek, Radosław Włodzimierz, 223.
- $T_0$  (largest possible stage), 72, 227, 233.
- $T_0$  (set of all solutions), 103, 106.
- Table constraint, 46–48, 110, 135, 142.
- Tableau shape, 162.
- Tapestry, 79, 238.
- Tardif, Claude Henri Joseph, 248.
- Tarjan, Robert Endre, v, vi, viii.
- algorithm for strong components, vi–ix, 81, 82.
- Tautology, 158.
- Taylor, Walter Fuller, term, 98, 153, 254.
- Tee tetrominoes, 221.
- Temperature, 5.
- Tensor product of graphs, *see* Direct product of graphs.
- Terms, 94.
- Ternary (3-ary) constraint, 2, 53, 88, 138, 149, 253.
- irreducible, 137–138.
- Ternary (3-ary) domain, 39, 49, 87, 90, 97, 138, 140, 150, 152, 154.
- Ternary Boolean relations, *see* Boolean ternary relations.
- Ternary Hamming graph, 129.
- Ternary-to-binary encodings, 207.

- Tetrahedron, 9–11, 166.
- Tetrominoes, 64, 147, 221.
- Thomas, Robin, 196.
- Thought experiment, 99, 102.
- Three-colorable, 38.
- Three-faced, 9.
- Three-valent polyhedral objects (3VP), 9–16, 115–119.
- Tied-path graphs, 192.
- Tilted squares, 238.
- Time stamps, 52–53, 109.
- Tolerance of weak digracefulness, 189.
- Tone rows, 74.
- Topological order, 81.
- Toroidal patterns, 222.
- Toroidal queens, 216.
- Torto, 74.
- Torus graph, 124.
- Torus polyhedron, 164.
- Totient function ( $\varphi(n)$ ), 187, 188.
- Tournament digraphs, 27, 126.
- Trace of matrix, 118, 169–171.
- Trace theory, 125.
- Tractable families of CSPs, 79–85, 99–106, 110.
- Trail, 57, 60, 142.
- $\text{trail}(x)$ , 224.
- Tramu, Jacques, 75.
- Transcendental number, 249.
- Transition rule, 155.
- Transitive law, 103, 125, 130, 148.
- Transitive tournament, 189.
- Transpose of matrix, 65, 84, 171, 260.
- Trees for algebraic formulas, 94.
- Trees in subcube labels, 129.
- Trees in subgraph isomorphism, 38, 130.
- Treewidth, 79, 196.
- Triangles, 42.
  - colored, 65.
- Triangular grids, 131, 135.
- Tribonacci numbers, 161.
- Trie structure, 104–105, 155, 257.
- Trigger stacks, 71–73, 145.
- Trihedral world, 9–16, 115–119.
- Trimble, James Brian Rory, iv, 35, 205.
- Tripartite digraph, 131–132.
- Trivalent (3-regular) graphs, 44, 121.
- Trominoes, 221.
- Truncated tetrahedron, 15.
- Truth tables, 89, 212.
- Tullis, Ian Dashiell Duckworth, 79, 147.
- Tuple: An ordered sequence  $(x_1, \dots, x_k)$  of elements, often written simply  $x_1 \dots x_k$ , 2, 13, 147, 223.
- Turgeon, Jean Maurice, 179.
- Twain, Mark (= Clemens, Samuel Langhorne), iii.
- Tweaking, 77.
- Twindragon fractal, 118.
- Two-fan, 80, 240, 244.
- Types of identifiers, 86.
- Ullman, Jeffrey David, 4.
- Ullmann, Julian Richard, 108–109, 201.
- Unary (1-ary) constraint, 2, 53, 80, 91, 96, 98, 106, 147, 149, 225, 240, 254.
- Unary encoding, 41, *see* Order encoding.
- Unary operations, 94, 154.
- Unary polymorphisms, 86, 106, 153.
- Undeletion, 59.
- Undoing, 72, 142.
- Uniform hypergraph, 127.
- Uniformly random solutions, 115.
- Union-find algorithm, 177, 178.
- Unique solution, problems that have a, 42, 116, 141, 160, 171, 186, 191, 197, 201, 204, 223, 237–239.
  - graceful labeling, 19, 20, 22, 23, 121, 176–181, 189, 190.
- Unit clauses, 207.
- Unit propagations, 48, 137, 138.
- Unit vector, 150, 259.
- United States graphs, 16–18, 32–35, 132.
- Universal algebra, 2, 94, 111, 153, 255.
- Universal graphs, 134.
- Universal relation, *see* Complete relation.
- Universal set of tuples, 256.
- ‘Unlike’ relation, 113.
- Unmatchable (blank) color, 237.
- Unmatched girl in maximum matching, 205.
- Up-up-or-down-down permutation, 115.
- Updates when dancing, 64.
- USA graph and subgraphs, 16–18, 32–35, 131, 132.
- Valeriotte, Matthew Anthony, 252.
- Valid tuple, 142.
- Validation code, 233.
- Valiente Feruglio, Gabriel Alejandro, 134.
- Valley folds, 10.
- Valley in search tree, 224–225.
- van Beek, Peter George, 110, 111, 168, 216.
- van Dongen, Marc Remko Casper, 111, 211, 217.
- Van Hentenryck, Pascal René Marc, 8, 110.
- Van Kessel, Philippe Thibault, 201.
- Vanden Eynden, Charles Lawrence, 183, 190, 191.
- Vardi, Moshe Ya’akov (משה יעקב ורדי), 4, 111, 239, 254.
- Variable ordering heuristics, dynamic, 65–69, 144.
- Variables, 1, 8, 155.
- Variance, 217.
- Vector spaces, 258.
- Vento, Mario, 200.
- Verhaeghe, Hélène Marie-Jeanne Nelly, 224.
- Vertex covers, 113.
- Vertices, 3, 9.

- Vesztergombi, Katalin, 242.  
 VF2 and VF3 solvers, 200.  
 Viable bindings, 50–52, 181.  
 Video, 205.  
 Vilarem, Marie-Catherine Maurer, 216.  
 Vision, computer, 9–16, 107–108.  
 Visiting all solutions, 63, 72, 82, 106, 148, 155.  
 Volume of a tetrahedron, 166.  
 Vortex in a matrix, 115.
- waerden* ( $j, k, n$ ) problem, 43.  
 Wainwright, Robert Thomas, 79.  
 Walks, 161.  
 WalkSAT, 9, 41.  
 Walsh, Toby, 7, 111, 216.  
 Waltz, David Leigh, 107–109.  
 Warners, Johannes (= Joost) Pieter, 56.  
 Weak near unanimity operators, 97–98, 106, 151–153.  
 Weakened encoding, 40–42, 44, 136, 207–209.  
 Weakly digraceful labelings, 127.  
 Weakly realizable HC pictures, 15, 119.  
 Weakly viable bindings, 50, 55.  
 Webster, Daniel, 1.  
 Weigel, Peter Heinrich, 76, 146, 214, 229, 235, 237, 251.  
 Weight of primary item, *see* WTD heuristic.  
 Wermuth, Udo Wilhelm Emil, iv.  
 Werth, Tobias, 196.  
 Whirlpool permutations, 115.  
 Widom, Jennifer, 4.  
 Wildcards, 156, 224.  
 Willard, Ross David, iv, 250, 252, 254, 255, pairing, 151.  
 Windmill dominoes, 64.  
 Winkler, Peter Mann, 128, 194.  
 Winston, Patrick Henry, 108, 169.  
 Wipe out, 51, 139, 214.  
 Witnesses, 101, 102.  
 WNU operators, 97–98, 106, 152, 153.  
 Word rectangles, 74, 79.  
 Word search puzzles, 65.  
 Word squares, 56.  
**WORDS**( $n$ ), the  $n$  most common five-letter words of English, 29, 74, 128, 138.  
 Wörlein, Marc, 196.  
 Wraparound edges, 22.  
 Wright, Edward Maitland, 188.  
 WTD heuristic, 67–68, 144, 230, 234, 238.  
 WTD<sup>†</sup> heuristic, 230.
- X pentominoes, 221.  
 XC problem: Exact cover problem, 31, 39, 114, 130, 136–138, 141, 146, 190.  
 XCC problem: Exact covering with colors, 1, 17, 38, 76, 117, 120, 125, 138, 140, 144, 146, 149, 152, 166, 182, 205, 247, as a CSP, 2, 54, 62, benchmarks, v, 64.  
 XCC representation of CSP, 2, 113, 135.  
 XCC solvers, v, 63–64, 72–73, 227–228.  
 XCSP format, 111.  
 XOR constraint, 245.
- Y pentominoes, 65.  
 Yablonsky, Sergei Vsevolodovich (Яблонский, Сергей Всеволодович), 110.  
 Yanov, Yuri Ivanovich (Янов, Юрий Иванович), 252.  
 Yap, Roland Hock Chuan (葉福泉), 109.  
 Yin, Minghao (殷明浩), 68.
- Zampelli, Stéphane, 196.  
 ZDD: A zero-suppressed decision diagram, 20, 120, 224.  
 Zemljič, Sara Sabrina, 209.  
 Zhang, Tianwei (张天玮), 205.  
 Zhang, Yuanlin (张元林), 109.  
 Zhu, Zichen (朱子晨), 112.  
 Zhuk, Dmitriy Nikolaevich (Жук, Дмитрий Николаевич), iv, 97, 98, 111, 248, 252, 254.