

Hardness Table Layout Hardness Table

MIT Hardness Grp.¹

Computer Science and Artificial
Intelligence Laboratory, MIT
32 Vassar St., Cambridge, MA 02139

Josh Brunner

Computer Science and Artificial
Intelligence Laboratory, MIT
32 Vassar St., Cambridge, MA 02139
brunnerj@mit.edu

Andy Tockman

Computer Science and Artificial
Intelligence Laboratory, MIT
32 Vassar St., Cambridge, MA 02139
tockman@mit.edu

Frederick Stock

University of Massachusetts, Lowell,
MA 01854
Frederick_Stock@student.uml.edu

Della Hendrickson

Computer Science and Artificial
Intelligence Laboratory, MIT
32 Vassar St., Cambridge, MA 02139
della@mit.edu

Hayashi Layers

Computer Science and Artificial
Intelligence Laboratory, MIT
32 Vassar St., Cambridge, MA 02139
hayastl@mit.edu

Timothy Gomez

Computer Science and Artificial
Intelligence Laboratory, MIT
32 Vassar St., Cambridge, MA 02139
tagomez7@mit.edu

Erik D. Demaine

Computer Science and Artificial
Intelligence Laboratory, MIT
32 Vassar St., Cambridge, MA 02139
edemaine@mit.edu

Jenny Diomidova

Computer Science and Artificial
Intelligence Laboratory, MIT
32 Vassar St., Cambridge, MA 02139
diomidova@mit.edu

Abstract

We analyze the complexity of summarizing given results into a table, organized by various parameters possibly constrained in order, to optimize various objectives. Our results include polynomial-, pseudopolynomial-, constant-, and zero-time algorithms; and NP- and coNP-hardness.

1. Introduction

Complexity theory has a long and storied tradition of summarizing results in large, convoluted tables [1–3, 6, 7, 9, 10, 12]—see Figure 1 for some examples. Of course, this practice isn’t strictly contained to any particular field [2, 4, 5], but complexity theorists seem to exhibit this behavior exceptionally often.

In this paper, we consider the computational problem of laying out a results table. A results table becomes necessary when the authors of a paper consider many variations of a problem, obtaining different results (e.g. P vs. NP-complete) for different variants.

For example, Král et al. [9] consider the game KPlumber, which occurs on a square grid with six different kinds of cell. They investigate the complexity of winning a given instance of KPlumber, when the input is restricted to using some subset of the cell types. Their table of results is shown in Figure 2.

Král et al. cleverly avoid a table with sixty-four rows by listing their results so as to cover several different subsets. For instance, their second result is containment in P when-

ever the set contains S but not D or T, and any combination of O, C, and X.

This style of table concisely and clearly conveys your results, but it is not obvious when you’ve finished, meaning you have results that cover every case—indeed, we show in Theorem 6.1 that this question is coNP-hard. To resolve this issue, we display the same results in a different format in Figure 3: each of the sixty-four versions of the problem (or subsets of the six letters) corresponds to a small cell of the table. We merge adjacent cells that are covered by the same result. It’s now obvious at a glance that the results cover every version of the problem, so the authors know it’s acceptable to stop (though ideally we would resolve the complexity of the green region).

In designing Figure 3, we had to choose which parameters go on each axis, and in what order to nest the parameters. In KPlumber it makes sense to always list the lack of a cell type before the presence of the cell type, but in other situations we may be able to choose from multiple orders of the values of a parameter. These choices greatly affect readability: different choices might result in a table that looks more like Figure 4. You can experiment with different choices for this and other tables from this paper using our open-source web app [11].

1.1 Decision problems

We now define some decision problems inspired by the above example.

We will assume that the variants of a problem under consideration can be described by p parameters, each of which has some finite set of possible values. In the KPlumber example, there are $p = 6$ parameters, each of which has two values indicating whether that cell type is allowed.

¹ Artificial first author to highlight that the other authors (in ascending order of number of vowels) worked as an equal group. Please include all authors (including this one) in your bibliography, and refer to the authors as “MIT Hardness Group” (without “et al.”).

Figure 1. Examples of results tables from the literature: [6], [12], [1], [4], [7], [2], [10], [5].

	P, Thm. 2.2	P, 2.16	NP-c, Thm. 3.9	OPEN	P, Thm. 2.16	P, Thm. 2.16	
	NP-c, Thm. 3.7	P, Thm. 2.2	NP-c, Thm. 3.9	OPEN	NP-c, Thm. 3.10	P, Thm. 2.16	
	NP-c, Thm. 3.1	NP-c, Thm. 3.1	P, Thm. 2.2	NP-c, Thm. 3.11	NP-c, Thm. 3.11	NP-c, Thm. 3.11	NP-c, Thm. 3.11
	NP-c, Thm. 3.2	NP-c, Cor. 3.3	NP-c, Cor. 3.3	P, Thm. 2.2	NP-c, Cor. 3.4	P, Thm. 2.16	
	NP-c, Cor. 3.5	NP-c, Cor. 3.5	NP-c, Cor. 3.5	NP-c, Cor. 3.6	P, Thm. 2.2	P, Thm. 2.16	
	NP-c, Cor. 3.4	NP-c, Cor. 3.4	NP-c, Cor. 3.4	NP-c, Cor. 3.6	NP-c, Cor. 3.4	P, Thm. 2.2	

id	author	year	complexity	ref
1	Wang	1961	NP-complete	[1]
2	Wang	1961	NP-complete	[2]
3	Wang	1961	NP-complete	[3]
4	Wang	1961	NP-complete	[4]
5	Wang	1961	NP-complete	[5]
6	Wang	1961	NP-complete	[6]
7	Wang	1961	NP-complete	[7]
8	Wang	1961	NP-complete	[8]
9	Wang	1961	NP-complete	[9]
10	Wang	1961	NP-complete	[10]
11	Wang	1961	NP-complete	[11]
12	Wang	1961	NP-complete	[12]
13	Wang	1961	NP-complete	[13]
14	Wang	1961	NP-complete	[14]
15	Wang	1961	NP-complete	[15]
16	Wang	1961	NP-complete	[16]
17	Wang	1961	NP-complete	[17]
18	Wang	1961	NP-complete	[18]
19	Wang	1961	NP-complete	[19]
20	Wang	1961	NP-complete	[20]
21	Wang	1961	NP-complete	[21]
22	Wang	1961	NP-complete	[22]
23	Wang	1961	NP-complete	[23]
24	Wang	1961	NP-complete	[24]
25	Wang	1961	NP-complete	[25]
26	Wang	1961	NP-complete	[26]
27	Wang	1961	NP-complete	[27]
28	Wang	1961	NP-complete	[28]
29	Wang	1961	NP-complete	[29]
30	Wang	1961	NP-complete	[30]
31	Wang	1961	NP-complete	[31]
32	Wang	1961	NP-complete	[32]
33	Wang	1961	NP-complete	[33]
34	Wang	1961	NP-complete	[34]
35	Wang	1961	NP-complete	[35]
36	Wang	1961	NP-complete	[36]
37	Wang	1961	NP-complete	[37]
38	Wang	1961	NP-complete	[38]
39	Wang	1961	NP-complete	[39]
40	Wang	1961	NP-complete	[40]
41	Wang	1961	NP-complete	[41]
42	Wang	1961	NP-complete	[42]
43	Wang	1961	NP-complete	[43]
44	Wang	1961	NP-complete	[44]
45	Wang	1961	NP-complete	[45]
46	Wang	1961	NP-complete	[46]
47	Wang	1961	NP-complete	[47]
48	Wang	1961	NP-complete	[48]
49	Wang	1961	NP-complete	[49]
50	Wang	1961	NP-complete	[50]

Model	τ	$n \times n$ Squares		
		Lower	Upper	Theorem
aTAM	1	$\Omega(\frac{\log n}{\log \log n})$	$\mathcal{O}(n)$	[33], [1]
aTAM	2	$\Theta(\frac{\log n}{\log \log n})$		[33], [1]
Flexible Glue aTAM	2	$\Theta(\log^{\frac{1}{2}} n)$		[2]
Seeded TA Det.	1	$\Theta((\frac{\log n}{\log \log n})^{\frac{1}{2}})$		Thms. 3.2, 7.1
Seeded TA ST	1	$\Theta(\log^{\frac{3}{2}} n)$		Thms. 3.4, 7.1
Seeded TA	1	$\Theta(\log^{\frac{1}{4}} n)$		Thms. 3.3, 7.1

AND	OR	MAJ	Edge-Weights	CGS	#CGS
\checkmark	\checkmark	\checkmark	Arbitrary	NP-c [2]	ASP-c (Thm. 15) (open)
\checkmark	\checkmark	\checkmark	Matching	NP-c [3]	#P-c (Thm. 11) (open)
\checkmark	\checkmark	\checkmark	Arbitrary	NP-c [3]	#P-c (Thm. 11) (open)
\checkmark	\checkmark	\checkmark	Matching	NP-c [3]	#P-c (Thm. 11) (open)
\times	\checkmark	\checkmark	Arbitrary	P (Thm. 17)	#P-c (Thm. 19) (open)
\times	\checkmark	\checkmark	Matching	P (Thm. 17)	#P-c (Thm. 19) (open)
\checkmark	\times	\checkmark	Arbitrary	P (Thm. 18)	#P-c (Thm. 11)
\checkmark	\times	\checkmark	Matching	P (Thm. 18)	FP (Thm. 12)
\times	\checkmark	\checkmark	Arbitrary	P (Thm. 17)	#P-c [1]
\times	\checkmark	\checkmark	Matching	P (Thm. 17)	FP (Thm. 13)

Rules	Game	Reachability-F	Reachability
$ABe \rightarrow eBA$	Leap	L-complete (Thm. 2)	L-complete (Thm. 2)
$eAB \rightarrow AeB$	Trivial	L (Cor. 1)	L (Cor. 1)
$AeB \rightarrow eAB$	Trivial	L (Cor. 1)	L (Cor. 1)
$AFe \rightarrow eFA$	Vault	L-complete (Thm. 2)	L (Cor. 1)
$AFB \rightarrow BFA$	Vault Swap	NL-complete (Thm. 2)	NL (Cor. 1)
$ABF \rightarrow BAF$	Push Swap at Fixed	NL (Cor. 1)	NL (Cor. 1)
$BAF \rightarrow ABF$	Pull Swap at Fixed	NL (Cor. 1)	NL (Cor. 1)
$ABe \rightarrow BAe$	Push Swap at Empty	NP-complete (Thm. 3)	NP-complete (Thm. 3)
$ABe \rightarrow BeA$	= Push Swap at Empty	NP-complete (Thm. 3)	NP-complete (Thm. 3)
$ABe \rightarrow eAB$	Push-1	PSPACE-complete [5]	PSPACE-complete [20]
$ABe \rightarrow AeB$	= Push-1	PSPACE-complete [5]	PSPACE-complete [20]
$eAB \rightarrow eBA$	Pull Swap at Empty	PSPACE-complete (Thm. 4)	PSPACE-complete (Thm. 4)
$AeB \rightarrow eBA$	= Pull Swap at Empty	PSPACE-complete (Thm. 4)	PSPACE-complete (Thm. 4)
$eAB \rightarrow ABe$	Pull?-1 [4]	PSPACE-complete [4]	OPEN
$AeB \rightarrow ABe$	= Pull?-1	PSPACE-complete [4]	OPEN
$eAB \rightarrow BAe$	Suplex-1	PSPACE-complete (Thm. 4)	P (Thm. 5)
$eAB \rightarrow BeA$	= Suplex-1	PSPACE-complete (Thm. 4)	P (Thm. 5)
$AeB \rightarrow BAe$	= Suplex-1	PSPACE-complete (Thm. 4)	P (Thm. 5)
$AeB \rightarrow BeA$	= Suplex-1	PSPACE-complete (Thm. 4)	P (Thm. 5)
$ABB \rightarrow BBA$	Swap-2	PSPACE-complete (Thm. 4)	OPEN
$ABB \rightarrow BAB$	Push Swap at Block	PSPACE-complete (Thm. 4)	OPEN
$BAB \rightarrow ABB$	Pull Swap at Block	PSPACE-complete (Thm. 4)	OPEN

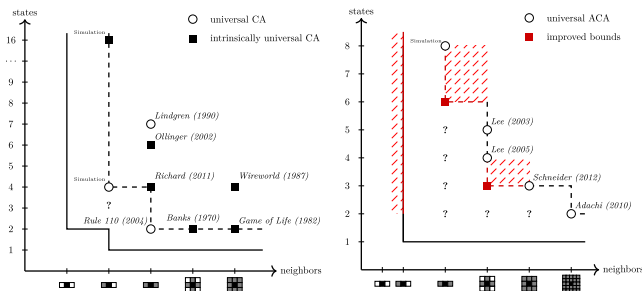


Figure 2. Results for KPlumber, from [9].

0	C	D	S	T	X	Complexity
★	★	★	×	★	★	Polynomial (Theorem 2)
★	★	×	✓	×	★	Polynomial (Theorem 3)
★	✓	✓	✓	×	★	NP-complete (Theorem 8)
★	×	★	✓	✓	★	NP-complete (Theorem 8)
★	✓	×	✓	★	★	Polynomial (Theorem 4)
★	×	✓	✓	★	★	The same (unknown) complexity (Theorem 9)

The sign \checkmark means that the corresponding type of tiles is present, the sign \times means that it is not present and \star means that it does not matter.

Figure 3. The same results for KPlumber as in Figure 2, presented differently.

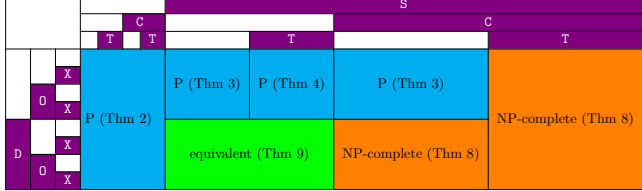
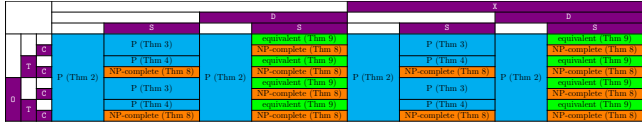


Figure 4. Figure 3 if we make bad choices.



In real life, sometimes a parameter has infinitely many possible values, such as any natural number, but authors must compromise with physical reality by only showing a finite fragment of the true infinite table, or by merging values into finitely many equivalence classes. We consider the layout problem only after this compromise has been made.

Usually our goal is to merge cells as much as possible, so that results aren't repeated several times throughout the table as in Figure 4. We will model this as minimizing the number of connected components of results in our table.

Definition 1.1. In *Readability*, we are given a list of parameters, a list of values for each parameter, a list of results, and a number k . We are asked whether it's possible to choose

- which parameters go on which axis,
- the nesting order of parameters on each axis,
- and the order of the values of each parameter

such that, when we draw the table with those choices and draw edges between adjacent cells with the same result, there are at most k connected components.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, contact the Owner/Author. Request permissions from permissions@acm.org or Publications Dept., ACM, Inc., fax +1 (212) 869-0481. Copyright held by Owner/Author. Publication Rights Licensed to ACM.

In this paper, we will assume that each instance of a parameter has the same nesting order (of the parameters after it) and the same order of values. Relaxing this assumption can make tables harder to parse, but sometimes allows more cells to be merged: for instance, the order of the columns could correspond to a gray code. This generalization is beyond the scope of the present paper, but is an enticing direction for future research.

We are also interested in knowing whether we’ve finished solving every version of the problem.

Definition 1.2. In **Completion**, we are again given parameters, values, and results. We are asked whether every combination of values is covered by a result.

We must now discuss the format in which results are presented. If each parameter has at most m values, there are up to m^p different problems. In particular, the size of the table we construct is exponential in the number of parameters and values.

In some cases, we consider each problem individually, and specify a result for each one. Then, in the input to our decision problems, we should be provided a list containing the result (or open status) of each problem. This list is typically exponentially long in m and p . We will call this the *explicit* encoding.

In other cases, including KPlumber, results are given by specifying the values of some parameters and letting other parameters vary. Visualizing the combinations of parameters as a p -dimensional hypercube, a result of this form describes an axis-aligned subspace. More generally, if parameters have more than two values, we can describe a result using a subset of the values of each parameter; the result applies to all problems in the Cartesian product of these sets. We will call this the *succinct* encoding.

In the succinct encoding, we will also allow results to overlap, with results listed earlier taking precedence. For example, the “open” result is listed last and contains all combinations of values, which means it applies to all the problems that aren’t covered by anything else.

So there are two versions of **Readability** and **Completion**, depending on how results are described.

As defined, **Readability** allows us to choose the order of the values of each parameter. But, as mentioned above for KPlumber, sometimes we don’t want that: maybe the values form a sequence of generalizations so we want them to be listed in order. More generally, there may be a partial order on each set of values, and we are required to list them in a compatible order.

Based on whether each parameter’s values have no structure, a given total order, or a partial order, we have three different versions of **Readability**.

Akitaya et al. [3] present a table of results on the complexity of certain origami problems, shown in Figure 5. One of their parameters (crease assignedness) has three values (assigned, unassigned, mixed) where one is a generalization of the other two. This could be modeled as a partial order, but instead of listing the common generalization last, the authors choose to put it in the middle, so that both special cases are adjacent to it. We can model this by supposing we are given a symmetric relation on the values of each parameter, and related values are required to be adjacent. This gives a fourth version of **Readability**.

In addition to parameter values, results often have a natural structure. There is often a natural structure on results as well as on values. In Figure 3, it is pleasing that

Figure 5. A table of origami results [3].

	ID	⊂	Rectangular	⊂	Arbitrary Paper	
					Infinite	&
One Layer	Assigned	poly	poly [ABD*04]	NP-comp. [ADK17]	&	NP-comp. (§5.3)
	Mixed	poly (§2.2)	poly (§2.3)	NP-comp.	&	NP-comp.
	Unassigned	poly	poly [ABD*04]	NP-comp. [ADK17]	&	NP-comp. (§5.3)
Some Layers	Assigned	poly	poly [ABD*04]	NP-comp. [ADK17]	&	NP-comp. (§5.3)
	Mixed	poly (§2.2)	NP-comp. (§4)	NP-comp.	&	NP-comp.
	Unassigned	poly	poly [ABD*04]	NP-comp. [ADK17]	&	NP-comp. (§5.3)
All Layers	Assigned	poly	poly [ABD*04]	OPEN	&	NP-comp. (§5.3)
	Mixed	poly (§3)	NP-comp. [AAB*20]	NP-comp.	&	NP-comp.
	Unassigned	poly	poly [ABD*04]	poly [AAB*20]	&	NP-comp. (§5.3)

all of the problems in P are in the upper left and all of the NP-complete problems are in the lower right. To model this, we suppose there is a partial order on results—in this case, all three P results are less than both NP-complete results—and we require that, in each row and column, the results are in an order compatible with this partial order. So we have another two versions of **Readability**, for whether results have a prescribed partial order. We could also consider total orders or relations on results, but these seem to appear less in practice and as such are beyond the scope of this paper.

When enough additional constraints are imposed, it may not even be possible to lay out the table.

Definition 1.3. In **Possibility**, we are given parameters, values, results, and possibly some additional structure. We are asked whether there is a layout for the table compatible with all constraints.

If it’s not possible to satisfy all constraints, we will need to make a compromise. The structure on parameter values or results could be a hard *constraint*, meaning we are not willing to compromise on it, or it could be an *objective*, meaning we aren’t required to satisfy it, but would like to come as close as possible to satisfying it.

Definition 1.4. In **Optimization**, we are given the same inputs as for **Possibility**, and a number k . We are asked whether there is a layout for the table that satisfies all constraints and all but k objectives.

Note that **Possibility** is a special case of **Optimization**, when there are only constraints and no objectives.

We need to clarify how we count objectives for each type of structure:

- For total or partial orders, for each a and b with $a < b$, there is an objective that a comes before b . That is, the cost of a layout is the number of pairs that are out of order.
- For relations, for each a and b with $a \sim b$, there is an objective that a and b are adjacent. That is, the cost of a layout is the number of related non-adjacent pairs.

We have defined **Readability** and **Optimization** as different problems. It would be reasonable to consider a combined optimization problem, where we wish to minimize (say) the sum of the number of connected components and the number of out-of-order pairs. Problems like this are beyond the scope of this paper, but are another interesting direction for further research.

We will make one final observation about the origami table in Figure 5. Two of its parameters (paper shape and crease assignedness) have a partial order structure where some values are generalizations of others. As a consequence,

Table 1. Our results for **Completion**.

input	m		$O(1)$	any
	p			
explicit	1	$O(1)$ 3.1	$O(n)$ 3.5	
	$O(1)$			
	any	$O(n)$ 3.5		
succinct	1	$O(1)$ 3.1	$O(n)$ 3.6	polynomial 3.8
	$O(1)$			
	any	coNP-complete 2.1 6.1		

easiness results (such as containment in P) are *downward closed* (easiness for a problem implies easiness for its special cases) and hardness results are *upward closed* (hardness for a problem implies hardness for its generalizations). Akitaya et al. represent this by drawing \subset for containment of problems and \implies for implications of results. This is also true for some of our parameters: a partial order is a generalization of both a total order and no structure. There are natural questions to ask about table layout in this setting, and the implications also complicate **Completion**. However, these considerations are also beyond the scope of this paper.

1.2 Summary of results

As discussed above, we consider several versions of the three decision problems **Completion**, **Readability**, and **Optimization** based on

- results encoding (explicit or succinct),
- value structure (for **Readability**, none, total order, partial order, or relation; for **Optimization**, this can be a constraint or an objective), and
- results structure (similarly none or partial order).

We will also parameterize by

- the number of parameters p (1, $O(1)$, or any),
- the maximum number of values m that each parameter has ($O(1)$ or any), and
- the dimension d of the table we wish to construct (1 or 2, for **Readability** and **Optimization**).

We summarize our results for **Completion**, **Readability**, and **Optimization** in Tables 1, 2, and 3, respectively, including references to the applicable theorems. Throughout, n is the size of the input.

In Section 2, we prove loose upper bounds on the complexity of all of these problems, which are not fully reflected in the tables. The remainder of this paper consists of our algorithms and hardness results, organized by complexity class: Section 3 has containments in P, Section 4 has quasipolynomial-time algorithms, Section 5 has NP-hardness results, and finally Section 6 has our coNP-hardness result.

2. General upper bounds

We now prove general containment results, which depend on the format results are given in.

Theorem 2.1. *Completion is in coNP.*

Proof. A certificate that the listed results don't cover all cases is a case that they don't cover. It is easy to verify such

a certificate in polynomial time, by checking the allegedly uncovered case against every result. \square

For **Readability** and **Optimization**, we need to understand the size of both a table and the space of tables.

Lemma 2.2. *When results are given explicitly, there are at most exponentially many different possible tables, each of which has linear size.*

Proof. It suffices to show that each table has linear size, since there are exponentially many linear-size objects. The size of the table is dominated by the number of cells in it, which is equal to the number of explicit results given in the input. \square

Theorem 2.3. *With explicit results, both Readability and Optimization are in NP.*

Proof. A certificate is a layout of the table, which has linear size by Lemma 2.2. All of our constraints and objectives, including the number of connected components for **Readability**, can be computed in polynomial time. The verifier simply checks that the certificate is a valid table, it satisfies all constraints, and it violates at most k objectives. \square

Lemma 2.4. *When results are given succinctly, there are at most doubly exponentially many different possible tables, each of which has at most exponential size.*

Proof. It suffices to show that each table has at most exponential size. If there are p parameters, each with at most m values, the number of cells in the table is at most m^p , which is exponential in the size of the input. \square

Theorem 2.5. *With succinct results, Readability and Optimization are in NEXP.*

Proof. A certificate is again a layout of the table, which is now exponentially large by Lemma 2.4. An exponential-time verifier can then perform the same steps as in Theorem 2.3. \square

3. Polynomial time

Theorem 3.1. *When there are $p = O(1)$ parameters, each with $m = O(1)$ values, all of our decision problems can be solved in $O(1)$ time.*

Proof. This is a semi-trivial statement. If there are only a constant number of parameters each equipped with only a constant number of possible values, then the input is of constant size. Hence, in this special case, we can brute force an optimal solution in $O(1)$ time. \square

Theorem 3.2. *If values are partially ordered and there is no structure on results, the answer to Possibility is always 'yes', and Optimization can always be solved with zero violations.*

Proof. Use an order of each parameter's values compatible with the partial order. There are no other requirements. \square

Theorem 3.3. *If $p = 1$, results are partially ordered, and there is no structure on values, the answer to Possibility is always 'yes', and Optimization can always be solved with zero violations.*

Table 2. Our results for *Readability*. The black area is when $p = 1$ and $d = 2$, which isn't meaningful (or is equivalent to $d = 1$).

p	d	m input results values	$O(1)$				any			
			explicit		succinct		explicit		succinct	
			set	poset	set	poset	set	poset	set	poset
1	1	set	$O(1)$ 3.1				$O(n \log n)$ 3.7	open	$O(n \log n)$ 3.7	open
		ordered					$O(n)$ 3.9			
		poset					open			
		relation								
	2	set	[black area]							
		ordered								
		poset								
		relation								
$O(1)$	1	set	$O(1)$ 3.1				open			
		ordered					P 3.10			
		poset					open			
		relation					NP-c 2.3, 5.4		NP-h 5.4	
	2	set	$O(1)$ 3.1				P 3.10			
		ordered					NP-c 2.3, 5.4		NP-h 5.4	
		poset					NP-c 2.3, 5.4			
		relation					NP-h 5.4			
any	1	set	QP 4.3				open			
		ordered					open			
		poset								
		relation					open			
	2	set	QP 4.3				NP-c 2.3, 5.4		NP-h 5.4	
		ordered					QP 4.2			
		poset					NP-c 2.3, 5.4		open	
		relation					NP-c 2.3, 5.4		you are here NP-h 5.4	

Proof. Since $p = 1$, the table has a single row, and one column for each value of the parameter. By ordering these values, we can order the columns however we wish. Choose an order compatible with the results' partial order. \square

Theorem 3.4. *When $p = 1$ and we have partial orders on both values and results, Possibility can be solved in linear time.*

Proof. Once again, the table has one row and we can order the columns arbitrarily, but now we need an order compatible with both partial orders. Since $p = 1$, even if given succinctly, each result applies to only one cell of the table. So we can assume there is exactly one result for each cell (if not, we could add them without changing the input size too much), and in particular it doesn't matter in which format results are given.

We assume that each partial order is given as a list of pairs that generate the order; i.e. as a directed acyclic graph that induces a partial order.

Since each value of the parameter corresponds to a single cell, we can consider the two input DAGs V and R to be on the same vertices. Define another directed graph $V \cup R$ on these vertices with the union of the edges. We wish to know whether there is an order of vertices compatible with $V \cup R$. This is exactly when $V \cup R$ is acyclic, which can be tested in linear time. \square

Theorem 3.5. *Completion can be solved in linear time when the results are given explicitly.*

Proof. In the explicit encoding, we are given each combination of parameter values as a separate result. So we simply scan through that list, and see if it contains every possible combination. \square

Theorem 3.6. *Completion can be solved in linear time when there is only one parameter.*

Proof. Because there is only one parameter, the explicit and succinct encoding are the same (if you allow that parameter to vary, then there is only one result and why is it a parameter anyway?). So by Theorem 3.5, it's solved in linear time. \square

Theorem 3.7. *When $p = 1$ and there is no structure on values or results, Readability can be solved in $O(n \log n)$ time.*

Proof. Since there is no structure on the values, we can arrange the table however we like. So we sort the one parameter by the results, which will result in the maximum amount of merging, with each result occupying only one merged cell in the final table. \square

Theorem 3.8. *Completion can be solved in polynomial time when $p = O(1)$.*

Proof. With the explicit encoding, it's done in linear time by Theorem 3.5. With the succinct encoding, we first convert to the explicit encoding and then use the same algorithm as in Theorem 3.5. Each result covers a hypercube containing

Table 3. Our results for **Optimization**, including the special case of **Possibility** when there are only constraints and no objectives.

p	results	m input d values	$O(1)$				any					
			explicit		succinct		explicit		succinct			
			1	2	1	2	1	2	1	2		
1	set	set	always 3.2	always 3.2	always 3.2	always 3.2	always 3.2	always 3.2	always 3.2	always 3.2		
		ordered const.										
		poset const.										
		poset obj.										
	poset const.	relation obj.	$O(1)$ 3.1		$O(1)$ 3.1		NP-c 2.3, 5.3		NP-h 5.3			
		set	always 3.3		always 3.3		always 3.3		always 3.3			
		ordered const.	$O(1)$ 3.1	$O(1)$ 3.1	$O(n)$ 3.4	$O(n)$ 3.4	NP-c 2.3, 5.2	NP-c 2.3, 5.3	NP-h 5.2	NP-h 5.3		
		poset const.										
	poset obj.											
	relation obj.											
	poset obj.	set	always 3.3		always 3.3		always 3.3		always 3.3			
		ordered const.	$O(1)$ 3.1	$O(1)$ 3.1	$O(n^2)$ 3.9	$O(n^2)$ 3.9	NP-c 2.3, 5.2	NP-c 2.3, 5.1	NP-h 5.2	NP-h 5.1		
poset const.												
poset obj.												
relation obj.												
$O(1)$	set	set	always 3.2									
		ordered const.	always 3.2									
		poset const.	always 3.2									
		poset obj.	always 3.2									
	poset const.	relation obj.	$O(1)$ 3.1				NP-c 2.3, 5.3		NP-h 5.3			
		set					open					
		ordered const.					P 3.10					
		poset const.					open					
	poset obj.	poset obj.	$O(1)$ 3.1				NP-c 2.3, 5.2		NP-h 5.2			
		relation obj.					NP-c 2.3, 5.3		NP-h 5.3			
		set					open					
		ordered const.					P 3.10					
any	set	poset const.	always 3.2									
		poset obj.	always 3.2									
		set	always 3.2									
		relation obj.	QP 4.3				NP-c 2.3, 5.3		NP-h 5.3			
	poset const.	set					open					
		ordered const.					QP 4.2		open			
		poset const.					open					
		poset obj.	NP-c 2.3, 5.2		NP-h 5.2							
	poset obj.	relation obj.	NP-c 2.3, 5.3		NP-h 5.3							
		set	open									
		ordered const.	QP 4.2		open							
		poset const.	NP-c 2.3, 5.2		NP-h 5.2							
poset obj.	poset obj.	NP-c 2.3, 5.1		NP-h 5.1								
	relation obj.	NP-c 2.3, 5.3		NP-h 5.3								
	set	open										
	ordered const.	QP 4.2										
poset const.	poset const.	NP-c 2.3, 5.2		NP-h 5.2								
	poset obj.	NP-c 2.3, 5.1		NP-h 5.1								
	relation obj.	NP-c 2.3, 5.3		NP-h 5.3								
	set	open										

up to m^p cells. Since p is constant, this is polynomial in m , so the explicit encoding is still polynomially sized, so the total time is polynomial. \square

Theorem 3.9. *When $p = 1$ and the values of the single parameter are totally ordered (as a constraint), **Readability** can be solved in linear time and **Optimization** can be solved in quadratic time.*

Proof. Since our parameter is totally ordered, we don't actually have any choice in laying out the table: it's just a matter of checking if putting all the results in that order satisfies the conditions. For readability, we can walk through the table and count the number of transitions between different types of results, which will give us the number of merged cells. For optimization, results may have a partial order objective. We need to check each pair of related results to see if they are in the correct order. There are $O(n^2)$ such pairs, so this takes $O(n^2)$ time. \square

Theorem 3.10. *When $p = O(1)$ and the values of each parameter are totally ordered, **Readability** and **Optimization** can be solved in polynomial time.*

Proof. Since the values are totally ordered, the only choice is the nesting structure of the parameters, including which dimension each goes in when $d = 2$. But since there are only a constant number of parameters, there are only a constant number of possible options, so we will just try all of them. Furthermore, since there are only $O(1)$ parameters, the number of cells in the table is polynomial in m , so we can write down the full table in polynomial time even if the results are given succinctly. Checking the number connected components in the table takes time linear in the size of the table, and checking the number of satisfied objectives takes time quadratic in the table size (see Theorem 3.9), so checking each option takes polynomial time. \square

4. Quasipolynomial time

We now consider some cases which have algorithms that aren't quite polynomial time, just by enumerating all the possible ways to lay out the table. We start with a logarithmic upper bound on the number p of parameters. All of our logarithms are base 2.

Lemma 4.1. *When results are given explicitly, the number of nontrivial parameters is at most $\log n$, where n is the size of the input.*

Proof. We consider a parameter with only one value trivial, since it doesn't affect our problems or algorithms. Let p be the number of parameters with at least two values.

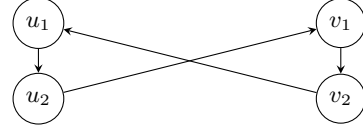
The number of cells in the table is at least 2^p . Since explicit results require the input to list a result for every cell, $n \geq 2^p$, which means $\log n \geq p$. \square

Theorem 4.2. *When results are given explicitly and values are totally ordered, all of our decision problems can be solved in $n^{O(\log \log n)}$ time.*

Proof. We ignore any trivial parameters. If the values are totally ordered, our only choice is the nesting order of the parameters. Thus in one dimension there are $p!$ different table layouts. By Lemma 4.1,

$$p! \leq (\log n)! < (\log n)^{\log n} = n^{\log \log n}.$$

Figure 6. The output of an edge (u, v) in Karp's reduction from Minimum Vertex Cover to Minimum Feedback Arc Set [8].



In polynomial time per table (as in Theorem 3.9), we can construct all possible tables and find the best one, and thus solve the problem in $n^{O(\log \log n)}$ time.

In two dimensions, we also choose which parameters go on each axis. We can count the number of tables by considering an order of the parameters plus a separator which partitions them into the two axes. So the number of layouts is now $(p + 1)!$, and the rest of the analysis is as before. \square

Theorem 4.3. *When results are given explicitly and each parameter has $O(1)$ values, all of our decision problems can be solved in $n^{O(\log \log n)}$ time.*

Proof. Assume each parameter has at most m values, for some constant m . We count the possible table layouts. There is still the choice of parameter order, but now each parameter has at most $m!$ orders for its values.

So the number of tables is at most

$$(p + 1)!(m!)^p = n^{O(\log \log n)} 2^{O(\log n)} = n^{O(\log \log n)}.$$

Once again, we can construct every possible table and check the validity and objective of each, in $n^{O(\log \log n)}$ time. \square

5. NP-hardness

5.1 From feedback arc set

Theorem 5.1. *When both values and results have a partial order objective, **Optimization** is NP-hard, even when $p = 1$.*

Proof. This is similar to the situation in Theorem 3.4, except that now we wish to find an order incompatible with as few ordered pairs as possible.

Minimum Feedback Arc Set (MFAS) is the problem of determining there is a set of at most k edges of a directed graph such that removing those edges leaves an acyclic graph. This is very similar to our present problem, and is known to be NP-complete [8]. But there are two differences:

- In our **Optimization** problem, the relation is the union of two partial orders (given as DAGs), not a general graph. But this doesn't really matter: every directed graph is the union of two DAGs on the same vertices.
- Our problem counts the violations of a partial order, whereas MFAS counts the number of violated edges of a DAG. Removing a single edge of a DAG can remove many related pairs in the induced partial order, so we can't reduce directly from MFAS.

We will see that Karp's simple reduction [8] from **Minimum Vertex Cover** to MFAS still works for us, and analyzing it will also prepare us for the next theorem. Given an undirected graph G and a number k , the reduction constructs a directed graph G' by replacing each vertex v with two vertices connected by an edge $v_1 \rightarrow v_2$, and replacing each edge (u, v) with two edges as in Figure 6.

An edge (u, v) in G becomes a cycle $u_1 \rightarrow u_2 \rightarrow v_1 \rightarrow v_2 \rightarrow u_1$ in G' , so any feedback arc set must contain at least one of these edges. Every cycle that uses $u_2 \rightarrow v_1$ also uses $v_1 \rightarrow v_2$, so replacing $u_2 \rightarrow v_1$ in a feedback arc set with $v_1 \rightarrow v_2$ still gives a feedback arc set. Thus we can consider only sets of edges $v_1 \rightarrow v_2$, which naturally correspond to sets of vertices of G .

Any such feedback arc set must include either $u_1 \rightarrow u_2$ or $v_1 \rightarrow v_2$, so the corresponding set of vertices is a vertex cover of G . Conversely, every cycle in G' must use some edge of the form $u_2 \rightarrow v_1$, meaning it also uses both $u_1 \rightarrow u_2$ and $v_1 \rightarrow v_2$. So a set of edges that contains at least one of these (for every edge of G) is a feedback arc set—removing the edges in the set breaks every cycle in G' . In particular, feedback arc sets of G' containing only edges of the form $v_1 \rightarrow v_2$ correspond precisely to vertex covers of G , and thus G' has a feedback arc set of size at most k exactly when G has a vertex cover of size at most k .

Finally, we adapt this proof to obtain a reduction from **Minimum Vertex Cover** to **Optimization** when $p = 1$ and values and results both have partial order objectives.

The values of the parameter are the vertices of G' . Each cell has a result, which we will also describe as a vertex of G' . The partial order on results is $v_1 \leq v_2$ for all v . The partial order on values is that, for each edge (u, v) of G , $u_2 \leq v_1$ and $v_2 \leq u_1$. Note that both of these partial orders contain only the pairs specified (and reflexivity), and nothing else is generated by transitivity. An order of the values of the parameter violates some number of related pairs of results and some number of related pairs of values. These violations correspond to the edges of G' in the relevant feedback arc set. \square

Theorem 5.2. *When both values and results are partially ordered, one as a constraint and the other as an objective, Optimization is NP-hard, even when $p = 1$.*

Proof. Since $p = 1$, values and results are symmetric, so it suffices to consider the case where values have a poset constraint and results have a poset objective.

We use the same reduction as in the proof of Theorem 5.1. As discussed there, there is always an optimal feedback arc set which uses only edges of the form $v_1 \rightarrow v_2$. An order that violates only these edges, and satisfies all edges of the form $u_2 \rightarrow v_1$, satisfies the constraint imposed by the values partial order. Hence the minimum number of objective violations in the present situation, where values have a constraint and results have an objective, is the same as when both partial orders were objectives. \square

5.2 From Hamiltonian paths

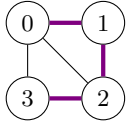
Theorem 5.3. *When the values of parameters have a relation objective, Optimization is NP-hard, even when $p = 1$.*

Proof. We reduce from finding Hamiltonian paths in undirected graphs.

Given a graph $G = (V, E)$, we make the values of the single parameter V , and impose E as a relation objective. Take $k = |E| - (|V| - 1)$. **Optimization** now asks us to find an order of the vertices such that at most k pairs of adjacent (in G) vertices that are not consecutive. Equivalently, we want at least $|V| - 1$ pairs of adjacent vertices that are consecutive.

There are only $|V| - 1$ total pairs of consecutive vertices, so to satisfy **Optimization** we need every consecutive pair to be adjacent. This happens exactly when the order is a

Figure 7. A graph with a highlighted Hamiltonian path, and the corresponding table from the proof of Theorem 5.4 (for $p = d = 2$). Empty cells are all unique results.



		Edge				
		01	02	03	12	23
Vertex	0	01-c	02-c	03-c		
	1				12-c	
	2		02-c			23-c
	3			03-c		

Hamiltonian path, and is possible exactly when G has a Hamiltonian path. \square

Theorem 5.4. *Readability is NP-hard when $p = d = 2$ and there is no structure or values or results.*

Proof. We reduce from finding Hamiltonian paths in undirected graphs.

Let $G = (V, E)$ be the original graph. Our 2 parameters are **Vertex** and **Edge**. For each vertex v in V , we add a parameter value v to **Vertex**. For each edge e we add a parameter value e to **Edge**. For each edge $e = \{u, v\}$, both (u, e) and (v, e) have the result ‘ uv -complete’. All other results are unique. The goal is to make the number of components at most $|V||E| - (|V| - 1)$.

Note that we can set up these results succinctly, by saying the shared results apply for edge $e = \{u, v\}$ and both vertices u and v .

If there’s a Hamiltonian path $(v_1, v_2, \dots, v_{|V|})$, we order the values of **Vertex** in the same order. We put **Vertex** and **Edge** on different axes, and order **Edge** arbitrarily. This makes the entries $(v_i, \{v_i, v_{i+1}\})$ and $(v_{i+1}, \{v_i, v_{i+1}\})$ adjacent. The results of both are $v_i v_{i+1}$ -complete, so these cells can be merged. This merges $|V| - 1$ pairs of cells, and there are $|V||E|$ cells total, so there are now $|V||E| - (|V| - 1)$ regions, satisfying the requirement. An example of such an arrangement is shown in 7.

Now suppose it’s possible to make the number of components at most $|V||E| - (|V| - 1)$. First, if **Vertex** and **Edge** are on the same axis, we can move **Edge** to the other axis without breaking any connected components. The only connected components that would break are between cells with differing values for **Edge** (unless $|E| = 1$, but then $|Edge| = 1$ so it might as well be on the other axis already), which never have the same result. We will assume for convenience that **Vertex** is the vertical axis and **Edge** is the horizontal axis, as in Figure 7.

Let $v_1, v_2, \dots, v_{|V|}$ be the order of **Vertex** in such a table. The only situation in which we can merge adjacent cells is that when there is an edge $e = \{v_i, v_{i+1}\}$, we can merge (v_i, e) with (v_{i+1}, e) . Because there is at most one edge between v_i and v_{i+1} , we can merge at most one pair of cells between each pair of consecutive rows, for a total of at most $|V| - 1$ merged pairs. This is tight, so to achieve the goal there must be a merged cell across each pair of consecutive rows. This requires that the edge $\{v_i, v_{i+1}\}$ exists for all i , which means $(v_1, v_2, \dots, v_{|V|})$ is a Hamiltonian path. \square

6. CoNP-hardness

Theorem 6.1. *Completion is coNP-hard when results are given succinctly, even when each parameter has $m = 2$ values.*

Proof. We consider the case where each parameter has two values, which for convenience we call ‘true’ and ‘false’. A typical result might apply whenever x is true, y is false, and z is true, where x , y , and z are parameters. For convenience, we will write abbreviate this result as $x \wedge \neg y \wedge z$.

In this notation, **Completion** asks: given some clauses which are conjunctions of boolean literals, do they together cover all assignments of true and false to the parameters? Of course, this is precisely the question of whether a given DNF formula is a tautology, where parameters are variables and results are clauses.

Determining whether a DNF formula is a tautology is coNP-complete: it is equivalent to the complement of satisfiability of CNF formulas. \square

References

- [1] Z. Abel, J. Bosboom, M. Coullombe, E. D. Demaine, L. Hamilton, A. Hesterberg, J. Kopinsky, J. Lynch, M. Rudyoy, and C. Thielen. Who witnesses The Witness? finding witnesses in The Witness is hard and sometimes impossible. *Theoretical Computer Science*, 839:41–102, 2020.
- [2] G. Aggarwal, Q. Cheng, M. H. Goldwasser, M.-Y. Kao, P. M. De Espanes, and R. T. Schweller. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34(6):1493–1515, 2005.
- [3] H. Akitaya, J. Brunner, E. D. Demaine, D. Hendrickson, V. Luo, and A. Tockman. Complexity of simple folding of mixed orthogonal crease patterns: Discrete and computational geometry, graphs, and games. *Thai Journal of Mathematics*, 21(4):1025–1046, 2023.
- [4] R. M. Alaniz, D. Caballero, S. C. Cirlos, T. Gomez, E. Grizzell, A. Rodriguez, R. Schweller, A. Tenorio, and T. Wylie. Building squares with optimal state complexity in restricted active self-assembly. *Journal of Computer and System Sciences*, 138:103462, 2023.
- [5] I. Baburin, M. Cook, F. Grötschla, A. Plesner, and R. Wattenhofer. Universality frontier for asynchronous cellular automata. *arXiv preprint arXiv:2502.05989*, 2025.
- [6] J. Brunner, L. Chung, M. Coullombe, E. D. Demaine, T. Gomez, and J. Lynch. Complexity of solo chess with unlimited moves. *arXiv preprint arXiv:2302.01405*, 2023.
- [7] J. Brunner, E. D. Demaine, J. Diomidova, T. Gomez, M. Hecher, F. Stock, Z. Zhou, et al. Easier ways to prove counting hard: A dichotomy for generalized #SAT, applied to constraint graphs. In *35th International Symposium on Algorithms and Computation (ISAAC 2024)*, pages 51–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.
- [8] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations: Proceedings of a Symposium on the Complexity of Computer Computations*, pages 85–103. Springer, 1972. ISBN 978-1-4684-2001-2. doi: [10.1007/978-1-4684-2001-2_9](https://doi.org/10.1007/978-1-4684-2001-2_9).
- [9] D. Král, V. Majerech, J. Sgall, T. Tichý, and G. Woeginger. It is tough to be a plumber. *Theoretical computer science*, 313:473–484, 2004.
- [10] H. Layers, J. Brunner, E. D. Demaine, J. Diomidova, T. Gomez, D. Hendrickson, Y. Kirkpatrick, J. Li, J. Lynch, R. Nag, and F. Stock. Agent motion planning as block asynchronous cellular automata: Pushing, pulling, suplexing, and more. In *International Conference on Unconventional Computation and Natural Computation*, pages 219–236. Springer, 2024.
- [11] MIT Hardness Group. Hyper-table. Interactive web app. <https://65440-2023.github.io/hyper-table/>.
- [12] MIT Hardness Group, J. Brunner, A. Tockman, F. Stock, D. Hendrickson, H. Layers, T. Gomez, E. D. Demaine, and J. Diomidova. Hardness table layout hardness table. In *Proceedings of the 9th Annual Conference on TBD (SIGTBD 2025)*, 2025. You Are Here.