

SAT-Solving the Poset Cover Problem

Chih-Cheng Rex Yuan and Bow-Yaw Wang

Institute of Information Science, Academia Sinica, Taipei, Taiwan
hello@rexyuan.com, bywang@iis.sinica.edu.tw

Abstract. The poset cover problem seeks a minimum set of partial orders whose linear extensions cover a given set of linear orders. Recognizing its NP-completeness, we devised a non-trivial reduction to the Boolean satisfiability problem using a technique we call swap graphs, which avoids the complexity explosion of the naive method. We also explore a special case solvable in polynomial time when the linear orders are linearizations of a single poset. By leveraging modern SAT solvers, we effectively solve instances with reasonable universe sizes. Experimental results using the Z3 theorem prover on randomly generated inputs demonstrate the effectiveness of our method.

1 Introduction

Orders have been in our lives as long as humans can order things. We organize our tasks, prioritize our goals, and rank our preferences. Orders are everywhere in our world.

Partial orders, a generalization of orders, are a mathematical construct that captures the essence of non-linear ordering. For example, we can order both the fields of physics and chemistry under the general category of natural science, but we cannot say one is under the other.

We can find partial orders in various places in the wild. Scheduling tasks and analyzing biological sequences are but a few examples of partial orders that find themselves in[3,2].

Mathematicians have formalized the concept of partial orders as a set of elements with a binary relation that is reflexive, antisymmetric, and transitive. This structure is called a partially ordered set, or *poset* for short.

By a foundational theorem called Szpilrajn extension theorem, every poset has a set of linear extensions. Intuitively, this means we can

“stretch” a poset into a linear order. This is a powerful result that allows us to study posets by looking at their linearizations.

In this paper, we address the poset cover problem. The poset cover problem is a combinatorial problem that involves finding a minimum set of partial orders whose linear extensions cover a given set of linear orders.

The poset cover problem is known to be NP-complete[3]. We hence chose to leverage the power of modern SAT solvers to tackle this problem.

We propose a novel and non-trivial method to reduce the poset cover problem to a Boolean satisfiability problem by utilizing the concept of swap graphs. This approach efficiently handles problem constraints and avoids the exponential growth of naive encodings.

Our method can solve instances of the poset cover problem with a reasonable universe size. We also explore a special case where the problem can be solved in polynomial time when the given set of linear orders constitutes the linearizations of a single poset.

Our experimental results demonstrate the effectiveness of the proposed method. We extensively tested using the Z3[1] theorem prover in Python on randomly generated inputs with varying universe and input sizes. The results show that our method performs well and could be further optimized by a simple divide-and-conquer heuristic.

Future work includes exploring further optimizations to handle larger inputs and investigating potential applications involving partial orders, such as message sequence charts.

2 Preliminaries

A *partial order* is a binary relation that is reflexive, antisymmetric, and transitive. A *partially ordered set* or *poset* is a binary relational structure $\mathcal{P} = (\Omega, \leq)$ where the *universe* Ω is a set and \leq is a partial order on Ω ; we refer to members of Ω as elements of \mathcal{P} and, where specificity is desired, to Ω as $\Omega_{\mathcal{P}}$ and \leq as $\leq_{\mathcal{P}}$.

Example 1. For the following definition, consider the poset \mathcal{P} over $\{a, b, c, d\}$ with $a \leq b$; $a \leq c$; $a \leq d$; $b \leq d$; and $c \leq d$.

The *cover* relation \prec of a poset \mathcal{P} is the transitive reduction of its order relation; it describes the case of immediate successor: for $x, y \in \mathcal{P}$, $x \prec y$ if and only if $x \leq y$ and there is no $z \in \mathcal{P}$ such that $x \leq z$ and $z \leq y$. For Example 1, the covered relation includes $a \prec b$; $a \prec c$; $b \prec d$; and $c \prec d$. Note that (a, d) is absent in \prec .

Notice that a poset is equivalent to an acyclic directed graph. Figure 1 describes \mathcal{P} with a graph $(V, E) = (\Omega, \prec)$.

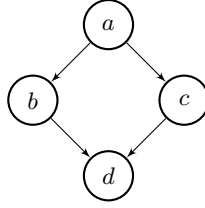


Fig. 1: Graph representation of \mathcal{P} from Example 1.

Example 2. For the following definition, consider the poset \mathcal{L} over $\{a, b, c, d\}$ with $a \leq b$; $a \leq c$; $a \leq d$; $b \leq c$; $b \leq d$; and $c \leq d$. Figure 2 represents \mathcal{L} with a graph $(V, E) = (\Omega, \prec)$.

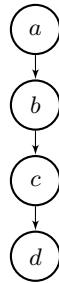


Fig. 2: Graph representation of \mathcal{L} from Example 2.

A partial order where every pair of elements is comparable is called a *linear order*, and a poset \mathcal{L} with such an order is called a *linear*

poset; that is, for $x, y \in \mathcal{L}$, $x \leq y$ or $y \leq x$. Note that the graph describing a linear poset is a path. For simplicity, we represent a linear poset in string form. For Example 2, we write **abcd**.

A linear poset \mathcal{L} that extends a poset \mathcal{P} is called a *linear extension* or *linearization* of \mathcal{P} , denoted $\mathcal{P} \sqsubseteq \mathcal{L}$; that is, for posets \mathcal{P}, \mathcal{L} with $\Omega_{\mathcal{P}} = \Omega_{\mathcal{L}}$, $\mathcal{P} \sqsubseteq \mathcal{L}$ if and only if \mathcal{L} is linear and $\leq_{\mathcal{P}} \subseteq \leq_{\mathcal{L}}$. For example, for \mathcal{P} from Example 1 and \mathcal{L} from Example 2, we have $\mathcal{P} \sqsubseteq \mathcal{L}$. Note that a linearization of a poset is equivalent to a topological sort of the graph describing that poset.

Every poset admits a set of linearizations. The set of all linearizations of a poset \mathcal{P} is denoted $L(\mathcal{P})$. We shall consider it the *language* of \mathcal{P} . For \mathcal{P} from Example 1, we have $L(\mathcal{P}) = \{\text{abcd}, \text{acbd}\}$. For \mathcal{L} from Example 2, $L(\mathcal{L}) = \{\text{abcd}\}$.

Now, we are ready to define the *poset cover problem*.

Definition 1 (Poset Cover Problem). *Given a set of linear posets \mathcal{Y} , find a set of partial orders C , called a cover, such that $|C|$ is minimal and the union of the languages of posets in C is equal to \mathcal{Y} ; that is, $\mathcal{Y} = \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$.*

Example 3. Given the set of linear posets $\{\text{abdce}, \text{badce}, \text{abcde}, \text{abdec}\}$ over $\{a, b, c, d, e\}$, a minimal poset cover of two posets \mathcal{A}, \mathcal{B} is shown in Figure 3a with $L(\mathcal{A}) = \{\text{abdce}, \text{abcde}, \text{abdec}\}$ and $L(\mathcal{B}) = \{\text{badce}\}$.

A minimal poset cover may not be unique, and the languages of the posets in a cover may overlap. For Example 3, there is another minimal poset cover, shown in Figure 3b, with overlapping languages $L(\mathcal{C}) = \{\text{abdce}, \text{abcde}, \text{abdec}\}$ and $L(\mathcal{D}) = \{\text{abdce}, \text{badce}\}$.

3 Reduction to SAT

Since the poset cover problem is proved to be NP-complete [3], we reduce the problem to SAT to utilize the power of modern SAT solvers. However, a naive encoding would easily result in superpolynomial transformation. We shall first show where the naive encoding falls short and then present a more efficient method.

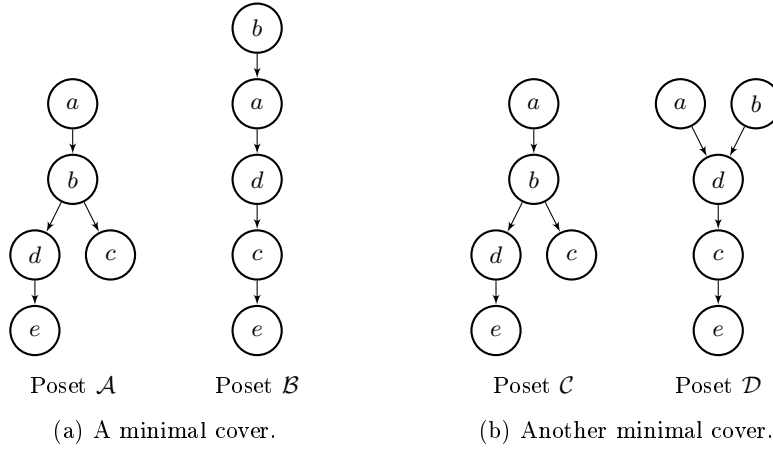


Fig. 3: Minimal covers for the set $\{\mathbf{abdce}, \mathbf{badce}, \mathbf{abcde}, \mathbf{abdec}\}$.

3.1 Basic Constraints

Before getting into constraints given by the problem, here we give the definitions and constraints for the basic poset construction.

Variable Semantics For a poset \mathcal{P} and $x \neq y \in \Omega$, we encode with the propositional variable $\mathbf{x}_{x,y}^{\mathcal{P}}$ the case that $x \leq_{\mathcal{P}} y$.

Poset Axioms Only antisymmetry and transitivity are relevant in our reduction. For a poset \mathcal{P} , we enforce antisymmetry with

$$\bigwedge_{x \neq y \in \Omega} \neg(\mathbf{x}_{x,y}^{\mathcal{P}} \wedge \mathbf{x}_{y,x}^{\mathcal{P}})$$

and transitivity with

$$\bigwedge_{x \neq y \neq z \in \Omega} \mathbf{x}_{x,y}^{\mathcal{P}} \wedge \mathbf{x}_{y,z}^{\mathcal{P}} \rightarrow \mathbf{x}_{x,z}^{\mathcal{P}}$$

Moreover, if \mathcal{P} is linear, we have to add on top of that

$$\bigwedge_{x \neq y \in \Omega} \mathbf{x}_{x,y}^{\mathcal{P}} \vee \mathbf{x}_{y,x}^{\mathcal{P}}$$

Linearization Constraints For posets \mathcal{P}, \mathcal{L} with $\mathcal{P} \sqsubseteq \mathcal{L}$, we encode axioms for both \mathcal{P} and \mathcal{L} and add

$$\bigwedge_{x \neq y \in \Omega} x_{x,y}^{\mathcal{P}} \rightarrow x_{x,y}^{\mathcal{L}}$$

Within logical formulae, we shall abuse the notation $\mathcal{P} \sqsubseteq \mathcal{L}$ as a macro for the above formula.

3.2 Problem Constraints

Our goal is to find a minimal set C of posets such that $\bigcup_{\mathcal{P} \in C} L(\mathcal{P})$ is equal to the input set of linear posets \mathcal{T} . To ensure we find the minimal cover, we encode the case when $|C| = k$, starting from $k = 1$, and incrementally query the SAT solver.

The poset cover problem can now be restated in logical formulae. Given \mathcal{T} and the size of the cover k , we first construct k posets by encoding their axioms. Next, we encode axioms for all the linear posets in \mathcal{T} and then specify their given order relations accordingly.

We shall encode the problem constraints, $\mathcal{T} = \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$, in a twofold manner, by separately constraining $\mathcal{T} \subseteq \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$ and $\mathcal{T} \supseteq \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$.

Naive Method A straightforward way to encode the problem follows immediately from the definition.

For $\mathcal{T} \subseteq \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$, it effectively states that every $\mathcal{L} \in \mathcal{T}$ is a linearization of some $\mathcal{P} \in C$. We encode it as

$$\bigwedge_{\mathcal{L} \in \mathcal{T}} \bigvee_{\mathcal{P} \in C} \mathcal{P} \sqsubseteq \mathcal{L}$$

For $\mathcal{T} \supseteq \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$, conversely, it states that every $\mathcal{L} \notin \mathcal{T}$ is not a linearization of all $\mathcal{P} \in C$. This is encoded as

$$\bigwedge_{\mathcal{L} \in \bar{\mathcal{T}}} \bigwedge_{\mathcal{P} \in C} \mathcal{P} \not\sqsubseteq \mathcal{L}$$

Theorem 1. *The constructed formulae are satisfiable if and only if the given case has a solution.*

Notice that encoding $\mathcal{Y} \supseteq \bigcup_{\mathcal{P} \in \mathcal{C}} L(\mathcal{P})$ this way would result in exponential blow-up from $\overline{\mathcal{Y}}$ as there are $|\Omega|!$ permutations and hence possible linear posets over Ω .

Swap Graph Method Since $\mathcal{Y} \subseteq \bigcup_{\mathcal{P} \in \mathcal{C}} L(\mathcal{P})$ can be efficiently encoded with naive method, we improve on encoding the other direction. We have devised, by building upon the idea of swap graph, a more efficient reduction that requires some preprocessing using notions defined as follows.

The *adjacent transposition*, or *swap*, relation \leftrightarrow describes the case of “off by one swap” between linear posets with shared universe: for linear posets $\mathcal{L}_1, \mathcal{L}_2$, $\mathcal{L}_1 \leftrightarrow \mathcal{L}_2$ if and only if there are $x, y \in \Omega$ such that $\leq_{\mathcal{L}_1} \cap \leq_{\mathcal{L}_2} = (\leq_{\mathcal{L}_1} \cup \leq_{\mathcal{L}_2}) - \{(x, y), (y, x)\}$; to specify which x, y induce the relation, we write $\leftrightarrow_{x,y}$. For example, $abcd \leftrightarrow_{b,c} acbd$. Note that \leftrightarrow is symmetric.

For a set of linear posets \mathcal{Y} , the *swap graph* $G(\mathcal{Y})$ of it is the undirected graph $(V, E) = (\mathcal{Y}, \leftrightarrow)$. Figure 4 shows the swap graph $G(L(\mathcal{P}))$ for \mathcal{P} from Example 1. Conveniently, a swap graph built from the language of a poset is connected [5,4,3]. An iteration of the proof is in Appendix A.

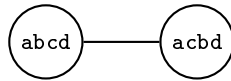


Fig. 4: Swap graph $G(L(\mathcal{P}))$ for \mathcal{P} from Example 1.

We can exploit this property that a swap graph of poset language is connected to avoid exponential blow-up, by “insulating” each closed connected component $v \subseteq G(\mathcal{Y})$ with a set $moat(v) = \{\mathcal{L} \notin v \mid \exists \mathcal{L}' \in v \ \mathcal{L}' \leftrightarrow \mathcal{L}\}$ that encloses it. We denote by $moat(\mathcal{Y})$ the set $\bigcup_{v \in comp(\mathcal{Y})} moat(v)$, where $comp(\mathcal{Y})$ is the family of all closed connected components in \mathcal{Y} . Figure 5 illustrates this idea of moats for two connected components.

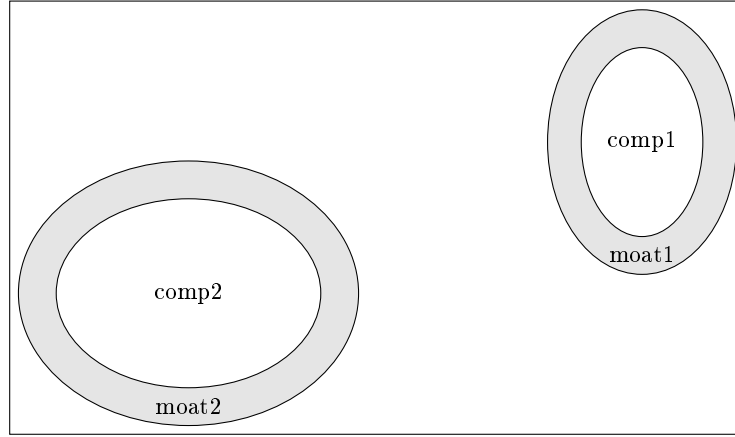


Fig. 5: Illustration of moats around connected components.

An alternative way to encode $\mathcal{Y} \supseteq \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$ is then

$$\bigwedge_{\mathcal{L} \in \text{moat}(\mathcal{Y})} \bigwedge_{\mathcal{P} \in C} \mathcal{P} \not\sqsubseteq \mathcal{L}$$

The construction of $G(\mathcal{Y})$ and $\text{moat}(\mathcal{Y})$ together costs $\mathcal{O}(|\mathcal{Y}| \cdot |\Omega|)$ by going through each $\mathcal{L} \in \mathcal{Y}$ and checking if $\mathcal{L}' \in \mathcal{Y}$ for all $\mathcal{L}' \leftrightarrow_{x,y} \mathcal{L}$ where $x \prec_{\mathcal{L}} y$, assuming constant membership query of \mathcal{Y} .

Proposition 1. *Swap graph method is equivalent to naive method.*

Proof. Suppose the moat constraints hold. If the naive constraints do not hold, then there are some $\mathcal{L} \notin \mathcal{Y}$ and $\mathcal{P} \in C$ such that $\mathcal{P} \sqsubseteq \mathcal{L}$. Per assumption, $\mathcal{L} \notin \text{moat}(\mathcal{Y})$, but then $G(L(\mathcal{P}))$ is not connected, a contradiction. The converse holds by definition, as $\text{moat}(\mathcal{Y}) \subseteq \overline{\mathcal{Y}}$.

3.3 Some More Improvements

Reverting to Naivety When the input linearizations overwhelm the swap graph universe $|\text{moat}(\mathcal{Y})| > |\overline{\mathcal{T}}|$ and the moat method becomes inefficient, we can revert back to the naive method. This is especially useful when the input is extremely dense.

XOR Encoding Recall in section 3.1, when encoding linear posets, we had to enforce total comparability by adding more clauses. We can avoid this requirement since it, combined with antisymmetry, is equivalent to an encoding using $\text{XOR}(\oplus)$.

$$\bigwedge_{x \neq y \in \Omega} \neg(\mathbf{x}_{x,y}^{\mathcal{P}} \wedge \mathbf{x}_{y,x}^{\mathcal{P}}) \wedge \bigwedge_{x \neq y \in \Omega} \mathbf{x}_{x,y}^{\mathcal{P}} \vee \mathbf{x}_{y,x}^{\mathcal{P}} \leftrightarrow \bigwedge_{x \neq y \in \Omega} \mathbf{x}_{x,y}^{\mathcal{P}} \oplus \mathbf{x}_{y,x}^{\mathcal{P}}$$

Skipping \mathcal{L} To encode $\mathcal{P} \sqsubseteq \mathcal{L}$, we had to encode axioms for both posets. However, since the problem input includes complete order relation $\leq_{\mathcal{L}}$ of all given $\mathcal{L} \in \mathcal{T}$, we can skip the encodings for \mathcal{L} and set each variable constrained by \mathcal{L} directly on \mathcal{P} , by taking the contrapositive form of \sqsubseteq . This way, it is possible to lose all the variables for \mathcal{L} and the associated constraints.

$$\bigwedge_{x \neq y \in \Omega} \mathbf{x}_{x,y}^{\mathcal{P}} \rightarrow \mathbf{x}_{x,y}^{\mathcal{L}} \leftrightarrow \bigwedge_{x \neq y \in \Omega} \neg \mathbf{x}_{x,y}^{\mathcal{L}} \rightarrow \neg \mathbf{x}_{x,y}^{\mathcal{P}} \leftrightarrow \bigwedge_{x,y \in \overline{\leq_{\mathcal{L}}}} \neg \mathbf{x}_{x,y}^{\mathcal{P}}$$

Divide and Conquer Finally, with swap graph method, we need not encode the entire problem input in one go. Instead, we can tackle each component $v \in \text{comp}(\mathcal{T})$ individually, which further reduces the number of clauses, with trade-off being the number of queries polynomial in $|\mathcal{T}|$. This is especially efficient when \mathcal{T} is sparse.

4 A Special Case

Although, in general, solving the poset cover problem is hard, in the case that the given set of linearizations constitutes the language of a single poset, the problem can be solved in polynomial time. We can consider this special case as an opening jab at the problem, returning if and once it succeeds.

We know, corollarily from Szpilrajn extension theorem, which states that every poset has some linearizations ($L(\mathcal{P}) \neq \emptyset$), that a poset is exactly the intersection of all its linearizations; that is, $\leq_{\mathcal{P}} = \bigcap_{\mathcal{L} \in L(\mathcal{P})} \leq_{\mathcal{L}}$ for any poset \mathcal{P} .

5 Experimentation

To test the limits of our method, we experimented on distinct randomly generated inputs using the Z3[1] theorem prover from Microsoft Research in Python.

As inputs with a sparse swap graph can be quickly divided and reduced to smaller problems, we restricted our inputs to those with a single connected swap graph.

We ran 100 trials, with solver timeout set to 15 minutes, on each of the different settings of universe size $1 \leq |\Omega| \leq 10$ and input size $1 \leq |\mathcal{R}| \leq 100$. Figure 6 shows the number of timeouts out of 100 trials in each case.

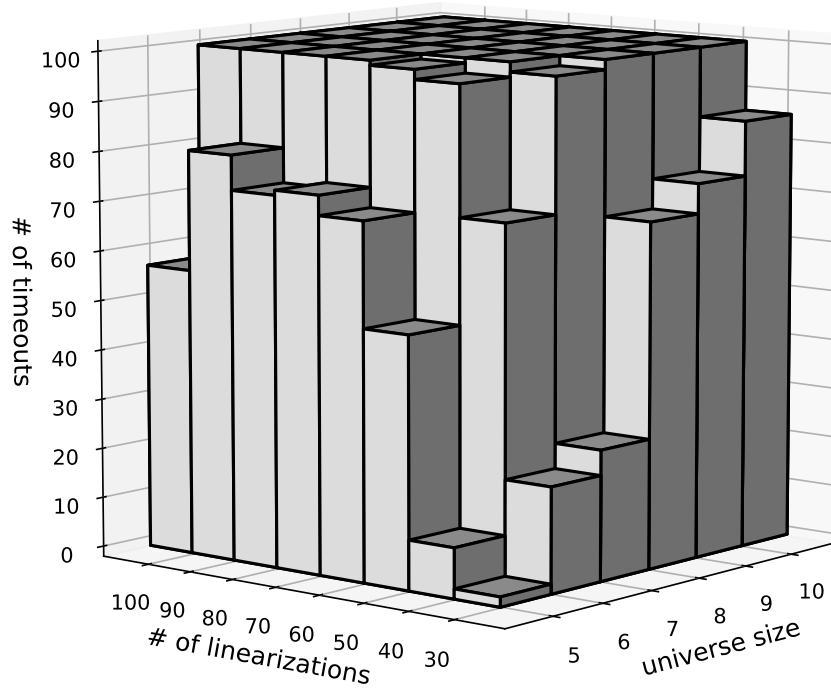


Fig. 6: Experimental results.

Cases where $|\mathcal{R}| \leq 20$ or $|\Omega| \leq 4$ are omitted because they gave no timeout under extensive testing (over 3000 trials) of combinations

with the largest the other parameter: $(|\mathcal{T}| = 100, |\Omega| = 4)$ and $(|\mathcal{T}| = 20, |\Omega| = 10)$.

These results show that our method is effective in practice when the universe size is reasonably small.

In addition, considering the swap graph in practice may be sparse and not a single connected component, the method can be applied to larger inputs.

In the case of swap graph being a single complex connected component, the number of timeouts increases with the size of the input, but the method is still viable for small to medium-sized inputs.

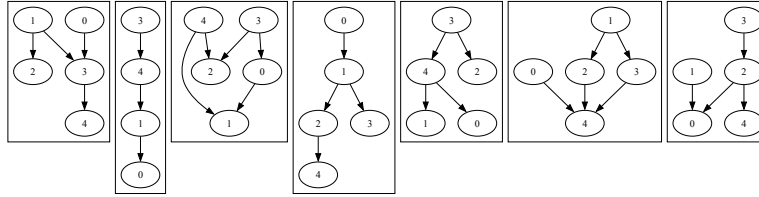


Fig. 7: Poset cover of an example with $(|\mathcal{T}| = 30, |\Omega| = 5)$.

An example poset cover output of a run with $(|\mathcal{T}| = 30, |\Omega| = 5)$ is shown in Figure 7. Its corresponding swap graph is shown in Figure 8.

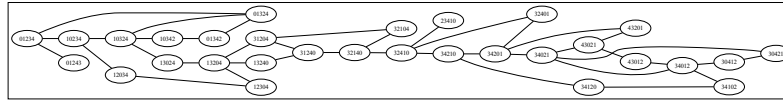


Fig. 8: Swap graph of an example with $(|\mathcal{T}| = 30, |\Omega| = 5)$.

6 Conclusions

We presented a novel and non-trivial method to reduce the poset cover problem to Boolean satisfiability problem. We utilized what

we call the “swap graph” to avoid exponential blow-up in naive encodings.

Our approach efficiently handles problem constraints and solves instances with a reasonably universe size even under the case where the divide-and-conquer heuristic cannot be used. We also considered a special case where the problem can be solved in polynomial time.

Experimental results demonstrate the effectiveness of our method in practice. Future work includes exploring further optimizations to handle larger inputs and potential applications where partial orders are involved, such as message sequence charts.

Appendix A Details

Here we define the *swapping* function. For a linear poset \mathcal{L} and some $x, y \in \Omega$ with $x \prec y$, $\mathcal{L}[x, y]$ is a linear poset \mathcal{L}' such that $\mathcal{L} \leftrightarrow_{x, y} \mathcal{L}'$.

A *swapping sequence* of length n from a linear poset \mathcal{L} to a linear poset \mathcal{L}' is a sequence of linear posets $\mathcal{L} = \mathcal{L}_0, \mathcal{L}_1, \dots, \mathcal{L}_n = \mathcal{L}'$ obtained by collecting the breadcrumbs of recursively swapping \mathcal{L} with some sequence of element pairs $x_1, y_1 ; \dots ; x_n, y_n$ such that $\mathcal{L}[x_1, y_1] \dots [x_n, y_n] = \mathcal{L}'$; i.e., $\mathcal{L}[x_1, y_1] \dots [x_k, y_k] = \mathcal{L}_k$ for $0 < k < n$.

The length of a minimal swapping sequence from \mathcal{L} to \mathcal{L}' is the inversion number $inv(\mathcal{L}, \mathcal{L}')$ sorting \mathcal{L} to \mathcal{L}' , as each swapping in a sequence either increases or decreases the inversion number by 1 [5], and a minimal sequence is where it decreases every step; i.e., $inv(\mathcal{L}_{k+1}, \mathcal{L}') = inv(\mathcal{L}_k, \mathcal{L}') - 1$ for $0 \leq k < n$. It is also known as the Kendall tau distance, or the bubble-sort distance, between them, which counts the number of discordant pairs, i.e., inversions.

In addition, we describe the case of *incomparability* in a poset \mathcal{P} with \parallel relation: for $x, y \in \mathcal{P}$, $x \parallel y$ if and only if $x \not\prec y$ and $y \not\prec x$. For Example 1, there is $b \parallel c$.

Lemma 1. *For posets \mathcal{P}, \mathcal{L} with $\mathcal{P} \sqsubseteq \mathcal{L}$, if $\parallel_{\mathcal{P}} \neq \emptyset$, then there is x, y with $x \parallel_{\mathcal{P}} y$ such that $x \prec_{\mathcal{L}} y$.*

Lemma 2 ([3]). *For posets \mathcal{P}, \mathcal{L} with $\mathcal{P} \sqsubseteq \mathcal{L}$, if $x \parallel_{\mathcal{P}} y$ and $x \prec_{\mathcal{L}} y$, then there is \mathcal{L}' such that $\mathcal{P} \sqsubseteq \mathcal{L}'$ and $y \prec_{\mathcal{L}'} x$.*

Lemma 3. For posets $\mathcal{P}, \mathcal{L}, \mathcal{L}'$ with $\mathcal{L} \neq \mathcal{L}'$ and $\mathcal{P} \sqsubseteq \mathcal{L}, \mathcal{L}'$, $\leq_{\mathcal{L}} \cap \leq_{\mathcal{L}'} \subseteq \parallel_{\mathcal{P}}$ and $|\leq_{\mathcal{L}} \cap \leq_{\mathcal{L}'}| = \text{inv}(\mathcal{L}, \mathcal{L}')$.

Lemma 4. For posets $\mathcal{P}, \mathcal{L}, \mathcal{L}'$ with $\mathcal{L} \neq \mathcal{L}'$ and $\mathcal{P} \sqsubseteq \mathcal{L}, \mathcal{L}'$, their inversion set corresponds to the complement of their intersection.

Lemma 5 ([5]). For posets $\mathcal{P}, \mathcal{L}, \mathcal{L}'$ with $\mathcal{L} \neq \mathcal{L}'$ and $\mathcal{P} \sqsubseteq \mathcal{L}, \mathcal{L}'$, the distance between \mathcal{L} and \mathcal{L}' in $G(L(\mathcal{P}))$ is the length of a minimal swapping sequence from \mathcal{L} to \mathcal{L}' .

Lemma 6. For posets $\mathcal{P}, \mathcal{L}, \mathcal{L}'$ with $\mathcal{L} \neq \mathcal{L}'$ and $\mathcal{P} \sqsubseteq \mathcal{L}, \mathcal{L}'$, for any minimal swapping sequence π from \mathcal{L} to \mathcal{L}' , we have $\mathcal{P} \sqsubseteq \mathcal{L}''$ for all $\mathcal{L}'' \in \pi$.

Theorem 2. For a poset \mathcal{P} , $G(L(\mathcal{P}))$ is connected by exactly all the minimal swapping sequences.

Corollary 1. For a set of linear posets Υ , there is a poset \mathcal{P} with $L(\mathcal{P}) = \Upsilon$ if and only if for all $\mathcal{L}, \mathcal{L}' \in \Upsilon$ and for all minimal swapping sequence π from \mathcal{L} to \mathcal{L}' , $\mathcal{L}'' \in \Upsilon$ for all $\mathcal{L}'' \in \pi$.

Corollary 2. For a set of linear posets Υ , define \mathcal{P} such that $\leq_{\mathcal{P}} = \bigcap_{\mathcal{L} \in \Upsilon} \leq_{\mathcal{L}}$. $L(\mathcal{P}) = \Upsilon$ if and only if for all $\mathcal{L} \in \Upsilon$ where $x \prec_{\mathcal{L}} y$ and $x \parallel_{\mathcal{P}} y$, $\mathcal{L}[x, y] \in \Upsilon$.

References

1. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
2. Fernandez, P.L., Heath, L.S., Ramakrishnan, N., Tan, M., Vergara, J.P.C.: Mining posets from linear orders. Discrete Mathematics, Algorithms and Applications **5**(04), 1350030 (2013)
3. Heath, L.S., Nema, A.K.: The poset cover problem (2013)
4. Pruesse, G., Ruskey, F.: Generating the linear extensions of certain posets by transpositions. SIAM Journal on Discrete Mathematics **4**(3), 413–422 (1991)
5. Ruskey, F.: Generating linear extensions of posets by transpositions. Journal of Combinatorial Theory, Series B **54**(1), 77–101 (1992)