Poset Paper

Bow-yaw Wang[1] and Chih-chen Yuan[2]

[1] Academia Sinica, Taiwan
[2] National Taiwan Normal University, Taiwan

Abstract. hello world

1 Introduction

Intro part here

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 $x \lneq z$ and $z \lneq y$. 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 $(V, E) = (\Omega, \prec)$.

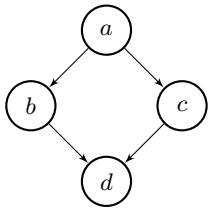


Fig. 1: Graph representation of $\mathcal{P}$ from Example 1.

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 $(V, E) = (\Omega, \prec)$.

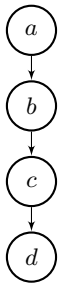


Fig. 2: Graph representation of $\mathcal{L}$ from Example 2.

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`.

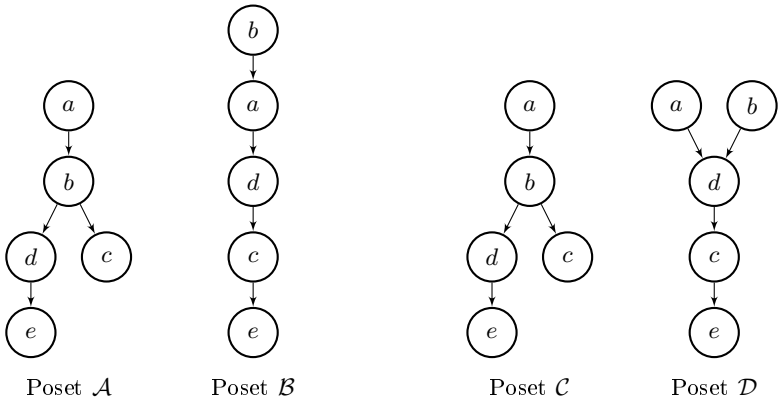
A linear poset $\mathcal{L}$ that extends a poset $\mathcal{P}$ is called a *linear extension* or *linearization* of $\mathcal{P}$, denoted $\mathcal{P} \sqsubseteq \mathcal{L}$; that is, for posets $\mathcal{P}, \mathcal{L}$ with $\Omega_P = \Omega_L$, $\mathcal{P} \sqsubseteq \mathcal{L}$ if and only if $\mathcal{L}$ is linear and $\leq_P \subseteq \leq_L$. 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 $L(\mathcal{P}) = \{\texttt{abcd}, \texttt{acbd}\}$. For $\mathcal{L}$ from Example 2, $L(\mathcal{L}) = \{\texttt{abcd}\}$.

Now, we are ready to define *Poset Cover Problem*.

Definition 1 (Poset Cover Problem). *Given a set of linear posets Υ, 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 Υ; that is, $\Upsilon = \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$.*

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



(a) A minimal cover.

(b) Another minimal cover.

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

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 $L(\mathcal{C}) = \{\texttt{abdce}, \texttt{abcde}, \texttt{abdec}\}$ and $L(\mathcal{D}) = \{\texttt{abdce}, \texttt{badce}\}$.

3 Reduction to SAT

Since the poset cover problem is proved to be NP-complete [1], 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.

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 $\mathtt{X}^{\mathcal{P}}_{x,y}$ 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

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

and transitivity with

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

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

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

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

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

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

COMMENT TO BOW YAW:
I'm not sure if this is okay but it can reduce one quantifier in later formulae

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 Υ. To ensure we find the minimal cover, we encode the case when $|C| = k$, starting from $k = 1$, and incrementally query SAT solver.

1. Let $k = 1$.
2. Encode the case and query solver.
3. If a solution is found, we are done. Else, let $k = k + 1$ and go to step 2.

The poset cover problem can now be restated in logical formulae. Given Υ 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 Υ and then specify their given order relations accordingly.

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

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

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

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

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

$$\bigwedge_{\mathcal{L}\in\overline{\Upsilon}} \bigwedge_{\mathcal{P}\in C} \mathcal{P} \not\sqsubseteq \mathcal{L}$$

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

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

Swap Graph Method Since $\Upsilon \subseteq \bigcup_{\mathcal{P} \in C} L(\mathcal{P})$ 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 Υ, the *swap graph* $G(\Upsilon)$ of it is the undirected graph $(V, E) = (\Upsilon, \leftrightarrow)$. 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 [3,2,1].

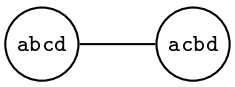
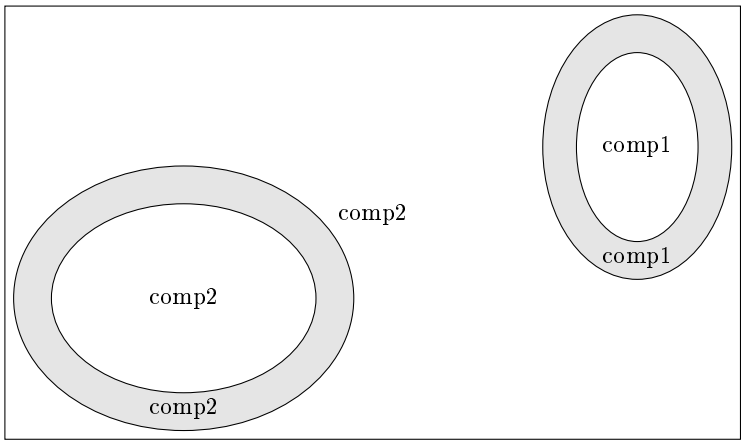


Fig. 4: Swap graph $G(L(\mathcal{P}))$ for $\mathcal{P}$ from Example 1.

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 $\upsilon \subseteq G(\Upsilon)$ with a set $moat(\upsilon) = \{\mathcal{L} \notin \upsilon \mid$ there is $\mathcal{L}' \in \upsilon$ such that $\mathcal{L}' \leftrightarrow \mathcal{L}\}$ that encloses it. We denote* by $moat(\Upsilon)$ the set $\bigcup_{\upsilon \in comp(\Upsilon)} moat(\upsilon)$, where $comp(\Upsilon)$ is the family of all closed connected components in Υ.



COMMENT TO BOW YAW:
(1) Is ‘‘closed’’ connected component an accepted term?
(2) I replaced ‘‘barrier’’ with ‘‘moat’’.
(3) I try $moat(\Upsilon)$ to reduce one quantifer. Is this actually better/clearer?

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

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

COMMENT TO BOW YAW:
Modified complexity justification

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

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 $\mathcal{L} \notin \Upsilon$ and $\mathcal{P} \in C$ such that $\mathcal{P} \sqsubseteq \mathcal{L}$. Per assumption, $\mathcal{L} \notin moat(\Upsilon)$, but then $G(L(\mathcal{P}))$ is not connected, a contradiction. The converse holds by definition, as $moat(\Upsilon) \subseteq \overline{\Upsilon}$.

Despite its complexity, in practice, when $|\Omega|! \approx |\Upsilon|$, it is better off to revert back to the naive method.

3.3 Some More Improvements

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

$$\bigwedge_{x \neq y \in \Omega} \mathtt{X}^{\mathcal{P}}_{x,y} \oplus \mathtt{X}^{\mathcal{P}}_{y,x}$$

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 $\mathcal{L} \in \Upsilon$, we can skip the axioms and set each variable directly.

In fact, it is even possible to skip encoding linearizations altogether. We can disregard the whole $\mathcal{L}$ part and lose all the variables for $\mathcal{L}$, by taking the contrapositive form of $\sqsubseteq$, so we encode $\mathcal{P} \sqsubseteq \mathcal{L}$ with

$$\bigwedge_{x,y \in \overline{\leq_{\mathcal{L}}}} \neg \mathtt{X}^{\mathcal{P}}_{x,y}$$

Finally, with swap graph method, we need not encode the entire problem input in one go. Instead, we can tackle each component $\upsilon \in comp(\Upsilon)$ individually, which could further reduce the number of clauses, with trade-off with the number of queries polynomial in $|\Upsilon|$, especially when Υ 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 $\mathcal{P}$.

```
COMMENT TO BOW YAW:
they've done wrong. (i think?)
```

so lets solve this? i have an idea
(1) Linearizations are connected (known)
(2) Linearizations are connected by exactly the “bubble sort paths (to be defined)”
(3) Algorithm: intersect the input then check all swapped neighbors with ||
(4) All nodes along all bubble sort paths are present iff all swapped neighbors with || of input are present

5 Example

Consider the following program, which creates a colored box inside the terminal screen and cycles three times through the colors red, green, and blue.

```
1  #include <unistd.h>
2  #include <ncurses.h>
3  int main(int argc, char *argv[])
4  {
5      // init options
6      initscr();
7      cbreak();
8      noecho();
9      nonl();
10     intrflush(stdscr, FALSE);
11     keypad(stdscr, TRUE);
12     curs_set(0);
13     // set colors
14     start_color();
15     init_pair(1, COLOR_WHITE, COLOR_RED);
16     init_pair(2, COLOR_WHITE, COLOR_GREEN);
17     init_pair(3, COLOR_WHITE, COLOR_BLUE);
18     // create window and go through colors
19     WINDOW* win = newwin(5, 15, 8, 4);
20     for (int i = 0; i < 9; i++, wrefresh(win), usleep(1000000))
21         wbkgd(win, COLOR_PAIR(i%3+1));
22     // wrap up
23     endwin();
24     return 0;
25 }
```

Lines 6–12 are a common initialization sequence with ncurses. Lines 14–17 initialize and define the colors to be used later. Lines 19–

21 change the color of the box every second. Lines 23–24 end the program.

```
COMMENT TO BOW YAW:
but i'm not so sure if this is apropos now. too many things
can change and change just interleavingly?
```

6 Experimentation

To test the limits of our method, we experimented on distinct randomly generated inputs using the z3 theorem prover from Microsoft Research. Also, 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 $|\Omega|$ and input size $|\Upsilon|$. Figure 5 shows the number of timeouts out of 100 trials in each case.

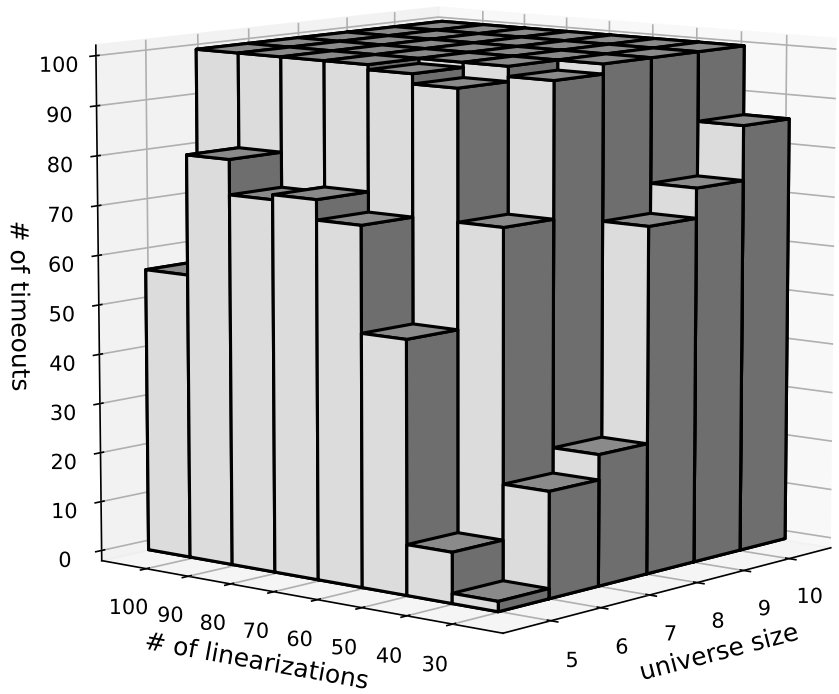


Fig. 5: Experimental results.

Cases where $|\Upsilon| \leq 20$ 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 ($|\Upsilon| = 100$, $|\Omega| = 10$).

7 Conclusions

The results were disappointing. The next step is to find a way to scale this method and to find a practical use.

Appendix A Swap things

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 $\mathcal{L} = \mathcal{L}_0, \mathcal{L}_1, ..., \mathcal{L}_n = \mathcal{L}'$ obtained by collecting the breadcrumbs of recursively swapping $\mathcal{L}$ with some sequence of element pairs x_1, y_1 ; ... ; x_n, y_n such that $\mathcal{L}[x_1, y_1]...[x_n, y_n] = \mathcal{L}'$; i.e., $\mathcal{L}[x_1, y_1]...[x_k, y_k] = \mathcal{L}_k$ 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 [3], 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 $\|$ relation: for $x, y \in \mathcal{P}$, $x \parallel y$ if and only if $x \not\leq y$ and $y \not\leq x$. For Example 1, there is $b \parallel c$.

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

Lemma 2 ([1]). *For posets $\mathcal{P}, \mathcal{L}$ with $\mathcal{P} \sqsubseteq \mathcal{L}$, if $x \|_{\mathcal{P}} y$ 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 $\mathcal{P} \sqsubseteq \mathcal{L}, \mathcal{L}'$, $\leq_{\mathcal{L}} \cap \leq_{\mathcal{L}'} \subseteq \|_{\mathcal{P}}$ and $|\overline{\leq_{\mathcal{L}} \cap \leq_{\mathcal{L}'}}| = inv(\mathcal{L}, \mathcal{L}')$.*

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 ([3]). *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. Heath, L.S., Nema, A.K.: The poset cover problem. Open Journal of Discrete Mathematics **3**(3), 101–111 (2013). https://doi.org/10.4236/ojdm.2013.33020
2. Pruesse, G., Ruskey, F.: Generating the linear extensions of certain posets by transpositions. SIAM Journal on Discrete Mathematics **4**(3), 413–422 (1991). https://doi.org/10.1137/0404037
3. Ruskey, F.: Generating linear extensions of posets by transpositions. Journal of Combinatorial Theory, Series B **54**(1), 77–101 (1992). https://doi.org/10.1016/0095-8956(92)90067-8