Two Heuristics for the Tree Poset Cover Problem

Willie N. Coronel Jr[1,*], Joshua Relucio[1], Ivy D. Ordanel[1], Richelle Ann B. Juayong[2]

[1]Algorithms and Complexity Laboratory, Department of Computer Science, University of the Philippines Diliman, Quezon City, Philippines
[2] Service Science and Software Engineering Laboratory, Department of Computer Science, University of the Philippines Diliman, Quezon City, Philippines
*wncoronel@up.edu.ph

Abstract. The Poset Cover Problem aims to find a minimum set of posets that cover a given input set of linear orders. This problem has practical applications in data mining, particularly in constructing directed networks from sequential data. The decision version of the problem is known to be NP-hard. In this study, we focus on a variant called the Tree Poset Cover Problem, which requires identifying a minimum set of tree posets needed to cover a given input set of linear orders. We propose two polynomial-time heuristics, namely Heuristic 1 and Heuristic 2. Our investigation demonstrates that both heuristics consistently produce feasible solutions and can be classified as approximation algorithms. Furthermore, we empirically evaluate the performance of Heuristic 1 and Heuristic 2 using four datasets.

Keywords: optimization problem, partially ordered sets, heuristics, approximation

1 Introduction

In Computer Science, there exists a well-known problem in generating all topological sortings of a directed acyclic graph (DAG). This problem takes a DAG as input and generates a set of topological sortings as its output. However, in the Poset Cover Problem, the scenario is somewhat reversed. Here, the input comprises a set of topological sortings, referred to as linear orders, and the goal is to find corresponding DAG(s), which are represented as posets, that cover the given input linear orders.

Formally, a (strict) partially ordered set or poset $P = (V, <_P)$ is defined as an ordered pair consisting of a finite set V and a binary relation $<_P \subseteq V \times V$ that is irreflexive, antisymmetric, and transitive. An example of a strict poset is shown in figure 1a. Two distinct elements $u, v \in V$ are said to be comparable in P, written as $u \perp_P v$, if and only if $u <_P v$ or $v <_P u$. Otherwise, they are incomparable, denoted as $u \parallel_P v$. When every pair $u, v \in V$ are comparable in $<_P$, then the poset is a totally ordered set or a linear order. Furthermore, a

J. Caro et al. (eds.), *Proceedings of the Workshop on Computation: Theory and Practice (WCTP 2023)*, Atlantis Highlights in Computer Sciences 20,
https://doi.org/10.2991/978-94-6463-388-7_2

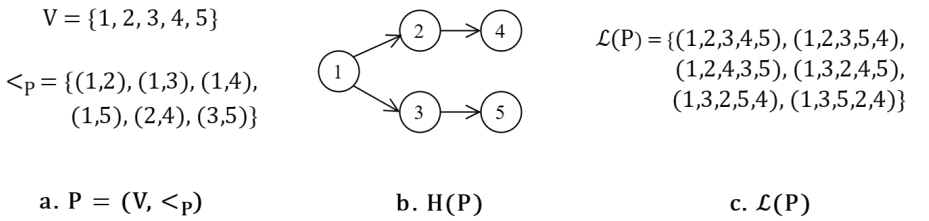

Fig. 1. Poset P, its Hasse diagram and linear extensions

linear order $L = (V, <_L)$ over the same set V is a linear extension of poset P if and only if $<_P \subseteq <_L$. The set of all linear extensions of poset P is denoted as $\mathcal{L}(P)$, as shown in figure 1c.

A cover relation $\prec_P = \{(u, v) | u <_P v$ and there is no $w \in V$ where $u <_P w <_P v\}$ is a subset of $<_P$. This relation defines the edges of the Hasse diagram of poset P, denoted as $H(P) = (V, \prec_P)$, as shown in figure 1b.

The Poset Cover Problem is formally defined as follows:

POSET COVER PROBLEM

INSTANCE: A set $\Upsilon = \{l_1, l_2, ..., l_m\}$ of linear orders over the set $V = \{1, 2, 3, ..., n\}$.

SOLUTION: A set $P^* = \{P_1, P_2, ..., P_k\}$ of posets where $\bigcup_{P_i \in P^*} \mathcal{L}(P_i) = \Upsilon$ and k is minimum.

The Poset Cover problem has been extensively studied, and the decision version of the problem has been proven to be NP-complete [1]. In order to expand the knowledge about the problem, efforts in the past have focused on exploring restricted variations of the problem and classifying which cases are in P and which are NP-hard. One way in which the problem can be restricted is by considering only a specific number of posets, say k. Another way is to consider only a specific class of posets according to their Hasse diagram, such as a tree poset. Moreover, the problem can be constrained by combining the former and the latter ways of restricting the Poset Cover Problem. For instance, the 1-Tree Poset Cover Problem is a restricted version because its output is $k = 1$ tree poset that covers the input linear orders, if it exists. This variation has been proven to be in P [10]. On the other hand, when the output is a set of $k = 2$ tree posets, the variation is called the 2-Tree Poset Cover Problem, and this has been shown to be in P as well [9]. However, the Tree Poset Cover Problem, where the output is a set of minimum k-tree posets, remains relatively unexplored, and its complexity is not yet known. Hence, the focus of this paper is on the Tree Poset Cover Problem Formally, it is defined as follows:

TREE POSET COVER PROBLEM

INSTANCE: A set $\Upsilon = \{l_1, l_2, ..., l_m\}$ of linear orders over the set $V = \{1, 2, 3, ..., n\}$.

SOLUTION: A set $P^* = \{P_1, P_2, ..., P_k\}$ of tree posets where $\bigcup_{P_i \in P^*} \mathcal{L}(P_i) = \Upsilon$ and k is minimum.

In solving the different variations of the Poset Cover Problem such as the Tree Poset Cover Problem, there are essentially three different ways to go about it. The first way is through an exact algorithm where the goal is to find an optimal solution for any given input set of linear orders [13]. The second way is by developing an approximation algorithm where the goal is to only find a feasible solution, albeit not necessarily optimal, for any given input [4]. The third way is through a heuristic algorithm where the objective is to find a good solution that is not necessarily feasible nor optimal. A previous study on the 2-Poset Cover Problem [11] presented three different polynomial-time heuristics. While these heuristics were able to accurately solve a significant majority of the random instances they were tested on, they encountered failures in a small number of cases.

In our study, we developed two heuristics for the Tree Poset Cover Problem, namely Heuristic 1 and Heuristic 2. We also proved that both heuristics can be considered as approximation algorithms as they always return a feasible solution for any given input set of linear orders.

2 Definition of Terms

In this section, we introduce and define the concepts and notations that are utilized throughout the discussion of the study.

Definition 1. ancestors(v,P). [8]
Given a poset $P = (V, <_P)$ and $v \in V$, the $ancestors(v, P)$ is the set of elements in poset P that precedes v, i.e., $ancestors(v, P) = \{a \in V | a <_P v\}$.

Definition 2. cover [8]
The term cover is used in many instances for different objects in the discussion. Given two elements $u, v \in V$ of poset P, we say that u covers v if and only if $u \prec_P v$. In this instance, (u, v) are also said to be cover pairs in P.
Given a set of linear orders Υ and poset P, we say that P covers Υ if and only if $\mathcal{L}(P) = \Upsilon$.
Given a set of linear orders Υ and a set of posets P^*, we say that P^* covers Υ if and only if $\bigcup_{P_i \in P^*} \mathcal{L}(P_i) = \Upsilon$.

Definition 3. Tree Poset [8]
A tree poset $P = (V, <_P)$ is a poset whose Hasse diagram is a rooted directed tree with each non-root node being covered by exactly one node.

Definition 4. tree_node_depth(v, P). [10]
The *tree_node_depth* is the distance of the node v from the root node and is given by the formula $tree\_node\_depth(v, P) = |ancestor(v, P)|$. The root itself has zero *tree_node_depth*.

Definition 5. depth(P). [10]
The $depth(P)$ for a tree poset P is the length of the longest path from the root node or

$$depth(P) = max_{v \in V}\{tree\_node\_depth(v, P)\}.$$

Definition 6. Heuristic Algorithm [2]
A heuristic algorithm is defined as a technique which seeks good (i.e., near optimal) solutions at a reasonable computational cost without being able to guarantee feasibility or optimality, or even in many cases to state how close to optimality a particular solution is.

Definition 7. Approximation Algorithm. [4]
An approximation algorithm for an optimization problem is an algorithm that provides a feasible solution whose quality does not differ too much from the quality of an optimal solution.

Definition 8. Approximation Ratio based on empirical data We define approximation ratio (based on empirical data) as the ratio between approximation solution and optimal solution, i.e.

$$\text{Approximation Ratio} = \frac{|\text{approximation solution}|}{|\text{optimal solution}|}$$

3 Methodology

3.1 Development of Heuristics

In this study, we attempted to develop a polynomial-time algorithm that solves the Tree Poset Cover Problem. In the course of finding a solution, we were able to devise two heuristics that have been proven to be approximation algorithms (to be discussed in section 4)

The first heuristic was developed based on the technique of combining posets [5–7]. On the other hand, the second heuristic was developed by utilizing an existing $O(mn)$-time algorithm for the 1-Tree Poset Cover Problem [10].

3.2 Test Case Generation

To assess the effectiveness of the developed heuristics, various test cases were generated. Input files containing linear extensions of tree posets were created, with the number of vertices (n) ranging from 3 to 6 and a chosen upper bound of the number of posets (k). Due to machine limitations, n and k were restricted to certain values. The input files consisted of random or exhaustive cases.

Two methods were used to generate the input files. The first method was exhaustive, involving all possible sets of tree posets for a specific n and k. The second method combined exhaustive and random approaches. For large values of n and k, all input test cases from k to $k-1$ were exhaustively generated, while the remaining cases were randomly generated.

Four separate input files, each representing a different set of test cases, were used in the study. Figure 2 provides details about these input files, including the number of vertices, the maximum number of tree posets, the generation method, and the total count of generated test cases.

File	Vertices (n)	Max. Tree Posets (k)	Generation Method	Test Cases
File 1	3	3	Exhaustive	63
File 2	4	3	Exhaustive	25,084
File 3	5	4	Random	100,000
File 4	6	2	Random	100,000

Fig. 2. The four input files used in this study

3.3 Examining the Performance of the Heuristics

To determine the accuracy of the heuristics developed, an analysis script was created. This script is responsible for evaluating each test case per input file against a heuristic. For each test case, both the optimal and the heuristic solution were recorded based on the results of the analysis in a text file. Furthermore, the script also logged the feasibility and optimality of the solution provided by the heuristic. This included determining whether the solution was feasible and optimal, feasible but non-optimal, or infeasible.

At the end of each text files, a summary – which includes the total number of inputs, the total number of feasible heuristic solutions, the total number of optimal heuristic solution, and the approximation ratio (based on empirical data) – is recorded.

4 Results

4.1 Theoretical Results

In this section, we discuss the two formulated heuristics - Heuristic 1 and Heuristic 2. We begin by exploring the theoretical outcome for each heuristic, and then proceed to the empirical result in section 4.2.

Heuristic 1. A trivial solution to the Tree Poset Cover Problem is the input set of linear orders itself since every linear order is a tree poset. However, it would be more satisfactory if we find tree posets that cover more linear orders. With this, we can start the solution with the given set of linear orders as tree posets. Then, what is left to do is to iteratively improve the solution by combining tree posets into a single tree poset such that the resulting tree poset generates all the linear extensions of the tree posets being combined. The following existing lemmas and theorem provide the condition as to when we can combine two posets into one poset.

Lemma 1. *[5] Given posets* $P_1 = (V, <_{P_1})$ *and* $P_2 = (V, <_{P_2})$*, if* $<_{P_1} \subseteq <_{P_2}$*, then* $\mathcal{L}(P_2) \subseteq \mathcal{L}(P_1)$*.*

Lemma 2. *[5] Given posets* $P_1 = (V, <_{P_1})$ *and* $P_2 = (V, <_{P_2})$*, if there exist pairs* $(a, b) \in <_{P_1}$ *and* $(b, a) \in <_{P_2}$ *such that* $<_{P_1} \setminus \{(a, b)\} = <_{P_2} \setminus \{(b, a)\}$*, then*

Algorithm 1: First Formulated Heuristic to the Tree Poset Cover Problem (Heuristic 1)

Input : A set $\Upsilon = \{L_1, L_2, ..., L_m\}$ over $V = \{1, 2, ..., n\}$
Output: A set of tree posets where $P = \{P_1, P_2, ...P_k\}$ such that $\mathcal{L}(P) = \mathcal{L}(Y)$ and k is minimum

```
1  P* ← GEN_TREE_POSET(Υ)
2  if P* ≠ null then
3      return P*
4  P* ← {L_1, L_2, ..., L_m}
5  P_tree ← ∅
6  isCombined = true
7  while isCombined do
8      isCombined ← false
9      while |P*| > 0 do
10         currentPoset ← P*[0]
11         Remove P*[0] in P*
12         hasPair ← false
13         for i ← 0 to |P*| do
14             combinedPoset ← CombinePoset(currentPoset, P*[i])
15             if combinedPoset ≠ null and
                L(combinedPoset) = L(currentPoset) ∪ L(P*[i]) then
16                 P_tree ← P_tree ∪ {combinedPoset}
17                 Remove P*[i] in P*
18                 hasPair ← true
19                 isCombined ← true
20                 break
21         if hasPair = false then
22             P_tree ← P_tree ∪ currentPoset
23     P* ← P_tree
24     P_tree ← ∅
25 return P*
```

Fig. 3. First Formulated Heuristic to the Tree Poset Cover Problem

(a, b) and (b, a) are cover pairs in P_1 and P_2, respectively, i.e., $(a, b) \in \prec_{P_1}$ and $(b, a) \in \prec_{P_2}$.

Theorem 1. *[5] Given posets $P_1 = (V, <_{P_1})$ and $P_2 = (V, <_{P_2})$, if there exists at most one pair $\{a, b\}, a \neq b$ such that $(a, b) \in \prec_{P_1}$ and $(b, a) \in \prec_{P_2}$ and $<_{P_1} \setminus \{(a, b)\} =<_{P_2} \setminus \{(b, a)\}$, then there exists a poset $P_3 = (V, <_{P_3})$ where $<_{P_3} =<_{P_1} \setminus \{(a, b)\} =<_{P_2} \setminus \{(b, a)\}$ and $\mathcal{L}(P_3) = \mathcal{L}(P_1) \cup \mathcal{L}(P_2)$.*

Theorem 1 formalizes the first case for which we can combine posets. Additionally, we have observed that there are tree posets that can be combined even if the number of binary relations of the tree posets being combined are unequal.

Algorithm 2: Combine Poset

```
1  <_{P_3} = <_{P_1} − <_{P_2}
2  <_{P_4} = <_{P_2} − <_{P_1}
3  if | <_{P_1} | = | <_{P_2} | then
4      if | <_{P_3} | ≠ 1 or | <_{P_4} | ≠ 1 then
5          return null
6      (a, b) ← <_{P_3}
7      (c, d) ← <_{P_4}
8      if a ≠ d or b ≠ c then
9          return null
10     <_P ← <_{P_1} − <_{P_3}
11 else if | <_{P_3} | > 1 and | <_{P_4} | > 1 then
12     return null
13 else
14     if | <_{P_3} | = 1 then
15         (a, b) ← <_{P_3}
16         if (b, a) ∉ <_{P_4} then
17             return null
18         if <_{P_1} \{(a, b)} ⊆ <_{P_2} \{(b, a)} then
19             <_P = <_{P_1} − <_{P_3}
20     else if | <_{P_4} | = 1 then
21         (a, b) ← <_{P_4}
22         if (b, a) ∉ <_{P_3} then
23             return null
24         if <_{P_2} \{(a, b)} ⊆ <_{P_1} \{(b, a)} then
25             <_P = <_{P_2} − <_{P_4}
26 if <_P is a Tree then
27     return P
```

Fig. 4. CombinePoset subroutine

Consider tree posets $P_4 = (V, <_{P_4})$ and $P_5 = (V, <_{P_5})$ where

$$<_{P_4} = \{(1,2),(1,3),(1,4),(2,3),(2,4)\}$$
$$<_{P_5} = \{(1,2),(1,3),(1,4),(2,3),(4,2),(4,3)\}$$

Let $(a, b) = (2, 4)$. Notice that when we remove $\{(a, b)\}$ from $<_{P_4}$ and $\{(b, a)\}$ from $<_{P_5}$, $<_{P_4}$ becomes a subset of $<_{P_5}$. From Lemma 1, it must be the case that $\mathcal{L}(P_5) \subseteq \mathcal{L}(P_4)$. Therefore, there must be a tree poset, say $P_6 = (V, <_{P_6})$ where $<_{P_6} = <_{P_4} \setminus \{(2,4)\}$ and $\mathcal{L}(P_6) \supseteq \mathcal{L}(P_4) \cup \mathcal{L}(P_5)$. We can also verify that the linear extensions that P_6 generates is equal to the union of $\mathcal{L}(P_4)$ and $\mathcal{L}(P_5)$.

$$\mathcal{L}(P_4) = \{1234, 1243\}$$
$$\mathcal{L}(P_5) = \{1423\}$$
$$\mathcal{L}(P_6) = \{1234, 1243, 1423\}$$

Indeed, $\mathcal{L}(P_6) = \mathcal{L}(P_4) \cup \mathcal{L}(P_5)$.

The caveat here is that there are instances where the resulting combined tree poset generates a linear extension that is not contained in either of the two posets being combined. Take for example the following posets $P_7 = (V, <_{P_7})$ and $P_8 = (V, <_{P_8})$ where

$$<_{P_7} = \{(1,2),(1,3),(1,4),(2,3)\}$$
$$<_{P_8} = \{(1,2),(1,3),(1,4),(3,2),(3,4)\}$$

Let $(a,b) = (2,3)$. When $\{(a,b)\}$ is removed from $<_{P_7}$, and $\{(b,a)\}$ is removed from $<_{P_8}$, the former becomes a subset of the latter. Again, from Lemma 1, it must be the case that $\mathcal{L}(P_8) \subseteq \mathcal{L}(P_7)$. However, these two posets cannot be combined into a single tree poset, say $P_9 = (V, <_{P_9})$ where $<_{P_9} = <_{P_4} \setminus \{(2,3)\}$ because P_9 generates an extra linear extension 1432 that is not contained in either $\mathcal{L}(P_4)$ or $\mathcal{L}(P_5)$. The following corollary formalizes the second case in which there exists an opportunity to combine two tree posets into a single one.

Corollary 1. *Given posets $P_1 = (V, <_{P_1})$ and $P_2 = (V, <_{P_2})$, if there exists at most one pair $\{a,b\}, a \neq b$, such that $(a,b) \in <_{P_1}$ and $(b,a) \in <_{P_2}$, and $<_{P_1} \setminus \{(a,b)\} \subseteq <_{P_2} \setminus \{(b,a)\}$, then there exists a poset $P_3 = (V, <_{P_3})$ where $<_{P_3} = <_{P_1} \setminus \{(a,b)\}$ and $\mathcal{L}(P_3) \supseteq \mathcal{L}(P_1) \cup \mathcal{L}(P_2)$.*

Proof. Since $<_{P_3} \subseteq <_{P_1}$ and $<_{P_3} \subseteq <_{P_2}$, by Lemma 1, it must be the case that $\mathcal{L}(P_1) \subseteq \mathcal{L}(P_3)$ and $\mathcal{L}(P_2) \subseteq \mathcal{L}(P_3)$. Hence, $\mathcal{L}(P_3) \supseteq \mathcal{L}(P_1) \cup \mathcal{L}(P_2)$. □

Theorem 2. *Heuristic 1 always returns a feasible solution to the Tree Poset Cover Problem in $O(m^3n^2)$, and is therefore classified as an approximation algorithm.*

Proof. The output of Heuristic 1 is the set P^*. Lines 1 to 3 verify if the input set of linear orders has a generating tree poset, that is, the aforementioned lines check whether the input can be covered by a single tree poset or not. The `GEN_TREE_POSET` function in line 1 represents the algorithm in [10]. However, if the input does not have a generating tree poset, P^* will contain all the input linear orders as tree posets. From this, P^* is already a feasible solution to the Tree Poset Cover Problem. To improve the solution, the algorithm iteratively combines two tree posets into a single tree poset. Two tree posets can be combined if the resulting poset is a tree poset that generates exactly the linear extensions of the posets being combined. The function *CombinePoset* combines posets P_i and P_j based on two conditions: (1) Theorem 1 [5]; and (2) Corollary 1. However, as explained earlier, there are instances in condition (2) where the resulting poset generates an extra linear extension. To rectify this, we have added an extra condition in line 15 in Algorithm 1 that checks whether the resulting combined poset generates exactly the linear extensions of the two posets being combined or not. Furthermore, the *CombinePoset* subroutine also ensures in line 26 that the resulting poset is a tree poset. Hence, P^* is a feasible solution to the Tree Poset Cover Problem. Furthermore, Heuristic 1 always returns a feasible solution, and is considered to be an approximation algorithm.

Now, we determine the running time complexity of the algorithm. The *CombinePoset* subroutine in line 14 runs in $O(n^2)$ since $| <_{P_1} | = | <_{P_2} |$ is $O(n)$. Determining if a pair is in $<_{P_1}$ or $<_{P_2}$ is just $O(1)$ since the two posets can be stored in a $(0,1)$-Matrix. Moreover, the *CombinePoset* is called in three loops – one for-loop and two outer while-loops. The inner for-loop runs in $O(m)$ since $|P^*| \in O(|Y|)$ and $|Y| = m$. On the one hand, the inner while-loop runs until P^* is empty, thus, in the worst case scenario it runs $O(m)$. On the other hand, the outer while-loop runs until *isCombined* is false. In the worst case, the loop runs in $O(m)$. Therefore, the running time complexity of the algorithm is $O(m^3n^2)$. □

Heuristic 2. Heuristic 2 uses the One Tree Poset Cover Problem algorithm in [10], which is aptly referred to as `OneTreePoset` in this section, where the input is a set of linear orders Υ and the output is a tree poset that covers Υ, if there exists any.

Figure 6 contains the pseudocode of the heuristic algorithm. It works by iterating a modified version of the `OneTreePoset` in lines 2-21 until all the input linear orders in Υ are covered. For each iteration h, we check if linear orders L_1 to L_h in Υ can be covered by a single tree poset. If the said linear orders have a generating tree poset, we push it to P_{tree} and remove the covered linear orders from Υ, so that in the next iteration, the algorithm will find a tree poset that would only cover the remaining linear orders. The outermost loop will only terminate once the cardinality of Υ drops to 0, that is, all linear orders in Υ have already been covered.

To illustrate Algorithm 3, consider the following input:

$$\Upsilon = \{12345, 12354, 12435, 13245, 13254, 13524, 14253, 14523, 14532, 15342, 15423, 15432\}$$

In the first iteration of the outermost for loop, $h = 12$ (see figure 5). With this, the algorithm will try to construct a single tree poset that could possibly cover linear orders L_1 up to L_{12} via lines 9 to 17. Observe that the constructed tree poset would be $P = (V, \prec_P)$ where $\prec_P = \{(1,2),(1,3),(1,4),(1,5)\}$. Line 18 checks whether $\mathcal{L}(P)$ is equal to the set containing L_1 up to L_{12}. Verify that the linear extensions of the poset P is not equal to the set $\{L_1, L_2, \ldots, L_{12}\}$. In the next iteration, h becomes 11. The constructed tree poset for linear orders L_1 to L_{11} would still be P. However, the linear extensions that P generates are still not equal to $\Upsilon[:11]$. The algorithm will continue on iterating and finding a tree poset that would exactly cover a subset of Υ. At $h = 6$, the linear orders that we are trying to construct a tree poset on are $\Upsilon = \{12345, 12354, 12435, 13245, 13254, 13524\}$. Verify that the constructed tree poset for this set of linear orders will be $P_1 = (V, \prec_{P_1})$ where $\prec_{P_1} = \{(1,2),(1,3),(2,4),(3,5)\}$. At this point, the set of linear extensions that P_1 generates is equal to linear orders L_1 to L_6. Hence, the algorithm will append poset P_1 to the variable P_{tree} and remove the linear orders in Υ that were already covered by the poset. The algorithm then breaks out of the outermost for loop.

Since Υ still has linear orders contained in it, the algorithm will continue on finding a tree poset that would cover each linear order. At this point, we have $h = 6$ because there are 6 linear orders left in Υ. Specifically, this iteration will work on $\Upsilon = \{14253, 14523, 14532, 15342, 15423, 15432\}$. The algorithm will eventually construct a tree poset $P_2 = (V, \prec_{P_2})$ where $\prec_{P_2} = \{(1,4), (1,5), (4,2), (5,3)\}$. Verify that P_2 covers the 6 remaining linear orders. In summary, $P_{tree} = \{P_1, P_2\}$ where

$$\prec_{P_1} = \{(1,2), (1,3), (2,4), (3,5)\}$$
$$\prec_{P_2} = \{(1,4), (1,5), (4,2), (5,3)\}$$

Note that $\mathcal{L}(P_1) \cup \mathcal{L}(P_2) = \Upsilon$, hence the output of the algorithm for the given input is a feasible solution, and optimal at that.

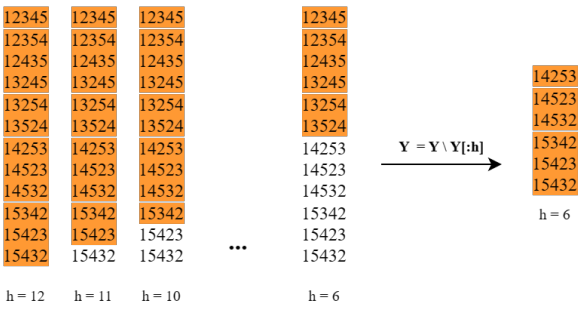

Fig. 5. Sample input for Heuristic 2

The following lemmas and theorem paved a way for devising Heuristic 2.

Lemma 3. *[10] LOWER AND UPPER LIMITS OF AN ELEMENT RANK IN A TREE POSET LINEAR EXTENSION. Given a tree poset $P = (V, <_P)$, and an element $v \in V$, if $i = |ancestor(v, P)|+1$ and $j = i+|incomparable(v, P)|$, then $i \leq rank(v, l) \leq j$, $\forall l \in \mathcal{L}(P)$.*

Lemma 4. *[10] Consider a tree poset $P = (V, <_P)$ and $v \in V$. $\forall k \in \{i, i+1, \ldots, j\}$ where i and j are the minimum and maximum ranks, respectively, of v from Lemma 3, $\exists l \in \mathcal{L}(P)$ such that $rank(v, l) = k$*

Lemma 5. *[10] Consider a tree poset $P = (V, <_P)$, two distinct elements $u, v \in V$, and a linear extension $l \in \mathcal{L}(P)$. Let $i = |ancestor(v, P)| + 1$ (min rank in Lemma 3). If $rank(v, l) = i$ and $rank^{-1}(i-1, l) = u$, then $u \prec_P v$.*

Lemma 6. *[10] The number of cover relations of a tree poset $P = (V, <_P)$ is given by $| \prec_P | = |V| - 1$.*

Algorithm 3: Second Formulated Heuristic to the Tree Poset Cover Problem (Heuristic 2)

Input : A set $\Upsilon = \{L_1, L_2, ..., L_m\}$ over $V = \{1, 2, ..., n\}$
Output: A set of tree posets where $P = \{P_1, P_2, ...P_k\}$ such that $\mathcal{L}(P) = \mathcal{L}(Y)$ and k is minimum

1 $P_{tree} \leftarrow \emptyset$
2 **while** $|\Upsilon| > 0$ **do**
3 **for** $h \leftarrow m$ **to** 1 **do**
4 $minRank \leftarrow$ an array with size n and initialized to 0
5 $numCoverRelation \leftarrow 0$
6 $\prec_P \leftarrow \emptyset$
7 **for** $i \leftarrow 2$ **to** n **do**
8 **for** $j \leftarrow 1$ **to** h **do**
9 $v_2 \leftarrow rank^{-1}(i, l_j)$
10 **if** $minRank[v_2] = 0$ **then**
11 $v_1 \leftarrow rank^{-1}(i-1, l_j)$
12 $\prec_P \leftarrow \prec_P \cup \{(v_1, v_2)\}$
13 $minRank[v_2] \leftarrow i$
14 $minRank[v_1] \leftarrow i - 1$
15 $numCoverRelation \leftarrow numCoverRelation + 1$
16 **if** $numCoverRelation = n - 1$ **then**
17 **break**
18 **if** $VERIFY(P, \Upsilon[:h])$ **then**
19 $P_{tree} \leftarrow P_{tree} \cup P$
20 $\Upsilon \leftarrow \Upsilon[h:]$
21 **break**
22 **return** P_{tree}

Fig. 6. Second Formulated Heuristic to the Tree Poset Cover Problem

Theorem 3. *Heuristic 2 always returns a feasible solution to the Tree Poset Cover Problem in $O(m^3n)$, and is therefore classified as an approximation algorithm.*

Proof. Heuristic 2 is basically just an iteration of a slightly modified version of the `OneTreePoset` algorithm. The `OneTreePoset` algorithm guarantees that it can generate a single tree poset for a set of linear orders that have a corresponding generating tree poset. That is, given an input set of linear orders, say Υ, the aforementioned algorithm can output a tree poset P such that $\mathcal{L}(P) = \Upsilon$.

Now, what is left to show is that the modified version of *OneTreePoset* algorithm in lines 2-21 is correct, i.e, $\mathcal{L}(P^*) = \Upsilon$ where P^* is the output set of tree posets generated by the algorithm.

First, we show that *OneTreePoset* is correct. From Lemma 4, for any $v_2 \in V$, $\exists l \in \mathcal{L}(P)$ such that $rank(v_2, l) = i$ and i is minimum. From Lemma 5, if $rank(v_2, l)$ is minimum, then its preceding adjacent element in l, say $v_1 = rank^{-1}(i-1, l)$, covers v_2, i.e., $(v_1, v_2) \in \prec_P$. Furthermore, by definition of a

tree poset, every non-root element is covered by exactly one distinct element. Thus, all the elements in the cover relation can be determined from the set of linear orders. This is also verified in line 16 with the formula given in Lemma 6.

For every iteration k, P_k is constructed from linear orders L_1 up to L_h of Υ. If $\mathcal{L}(P_k)$ is equal to $\{L_1, L_2, \ldots, L_h\}$, we append P_k to P_{tree} and the linear orders are removed from Υ to ensure that the construction of new tree posets will only be based on the remaining uncovered linear orders. Otherwise, we decrement h by 1 and construct a new tree poset that covers the linear orders L_1 to L_h. This will go on until we arrive at a tree poset that generates exactly the same linear extensions as $\{L_1, L_2, \ldots, L_h\}$. Once we generate a tree poset that covers a subset of Υ, the algorithm proceeds to the next iteration $k + 1$ to find a new tree poset that would cover the remaining linear orders. The outermost loop will then terminate once all linear orders in Υ are covered. Hence, Heuristic 2 always returns a feasible solution, and is considered to be an approximation algorithm.

Finally, we determine the time complexity of Heuristic 2. The initialization of array in line 4 runs in $O(n)$-time. The outer while loop contributes $O(m)$-time to the complexity since $|\Upsilon| = O(|\Upsilon|)$ and $|\Upsilon| = m$. The for loop in line 3 runs in $O(m)$, while the for loops in lines 7 and 8 run in $O(n)$ and $O(m)$, respectively. Lastly, the verification in line 18 runs in $O(mn)$-time [3,12]. Hence, the running time complexity of the entire algorithm is $O(m^3 n)$. □

4.2 Empirical Results

In this section, we discuss the analysis of Heuristic 1 and Heuristic 2 when subjected to testing using the input files described in figure 2. It was demonstrated in section 4.1 that Heuristic 1 and Heuristic 2 have been proven to be approximation algorithms. Hence, it can be expected that for any input set of linear orders, both heuristics can generate a set of tree posets that covers the input.

Figure 7 shows a summary of the approximation ratios of Heuristic 1 and Heuristic 2 when tested against input file 1, file 2, file 3, and file 4. Note that the calculated approximation ratios are based solely on empirical results and do not account for the theoretical ratios.

Heuristic	**File 1**	**File 2**	**File 3**	**File 4**
Heuristic 1	1.0	1.10	1.43	4.18
Heuristic 2	1.0	1.01	1.30	1.63

Fig. 7. Summary of the Approximation Ratio of each heuristic on each input file

Heuristic 1. Heuristic 1, when tested against file 1, was able to achieve an approximation ratio of 1.0. To determine why this is the case, let us investigate the structure of inputs contained in the file. All input linear orders in file 1 contain $n = 3$ vertices, which are sorted in increasing order. In a tree poset

$P = (V, <_P)$ where $|V| = 3$, there can only be two structures depending on the depth of the tree poset: (1) When $depth(P) = 1$, the number of linear extensions that P generates is two. Specifically, the two linear extensions that it generates – $L_1 = (V, <_{L_1})$ and $L_2 = (V, <_{L_2})$ – differ only by a single pair in terms of binary relation, i.e. there exists a pair $(a, b) \in <_{L_1}$ and $(b, a) \in <_{L_2}$ such that $<_{L_1} \setminus \{(a, b)\} = <_{L_2} \setminus \{(b, a)\}$. By Theorem 1, the two linear extensions can be combined into a single tree poset, say $P_3 = (V, <_{P_3})$ where $<_{P_3} = <_{L_1} \setminus \{(a, b)\} = <_{L_2} \setminus \{(b, a)\}$ and $\mathcal{L}(P_3) = L_1 \cup L_2$; and (2) When $depth(P) = 2$, there is only one linear extension that P generates, and can trivially be covered by a tree poset that is a linear order itself. Therefore, Heuristic 1 will always return an optimal solution to the Tree Poset Cover Problem when the input contains $n = 3$ vertices.

For input files 2 and 3, Heuristic 1 was able to achieve an approximation ratio close to 1. The heuristic can successfully cover any input that can be covered by a single tree poset through the use of the `OneTreePoset` algorithm [10]. However, for other inputs that cannot be covered by a single tree poset, there are some traps that the heuristic fall into. In input file 2, one of the cases where the heuristic was not able to return an optimal solution is when multiple tree posets are needed to cover the input and one or more of the tree posets P with $depth(P) = 1$ is needed to cover the input. Generalizing, as the number of vertices increased, it can be observed that when one of the tree posets P is needed to cover the given input and $depth(P) < |V| - 2$, the heuristic always returns a feasible but not optimal solution. It is also important to note that as the number of vertices increase, the structure of inputs also varies greatly. As a result, when $depth(P) \geq |V| - 2$, the heuristic guarantees an optimal solution. However, this is not the case for input files 3 and 4 due to the variation of structure of inputs.

Aside from the depth of the tree posets needed to cover a given input, the ordering of the input linear orders also affects the resulting output. There are instances where some tree posets are better combined with other sets of tree posets instead of the tree poset subsequent to them. However, once these tree posets are already combined with other tree posets based on their ordering, they can no longer be combined to a better set that will return an optimal solution.

Heuristic 2. Similar to Heuristic 1, Heuristic 2 achieved an approximation ratio of 1.0 for File 1. This is due to the structure of inputs contained in the file. As previously discussed, there are only two structures of tree posets when $|V| = 3$: $depth(P)$ is valued 1 or 2. In either case, Heuristic 2 guarantees an optimal solution since both cases can be covered by a single tree poset.

Heuristic 2 was able to achieve better approximation ratio values as compared to Heuristic 1. One of the factors that causes feasible but not optimal solutions for Heuristic 2 is the way linear orders are grouped and represented into one poset. Similar to Heuristic 1, the ordering of the input linear orders affects the output. Looking at a specific input, in figure 8, the linear extensions covered by tree poset P_1 are non-sequential when sorted alphanumerically. The

tree poset P_1 from the optimal solution covers linear extensions 1234, 1243, and 1423. However, the heuristic cannot produce the same tree poset because it considered the subsequent linear extension 1342 instead of 1432. Since the heuristic determined that the linear extension 1342 cannot be grouped with linear extensions 1234 and 1243, the heuristic was not able to produce the same tree poset P_1. As a result, the heuristic produces a feasible but not optimal solution. If the input had been arranged as $\Upsilon = \{1234, 1243, 1423, 1342\}$, the heuristic would have provided an optimal solution. Based on this observation, it can be deduced that certain input sets may have optimal solutions that consist of non-sequential linear extensions. However, Heuristic 2, which constructs tree posets based on the sequential ordering of linear orders, is unable to consider such cases.

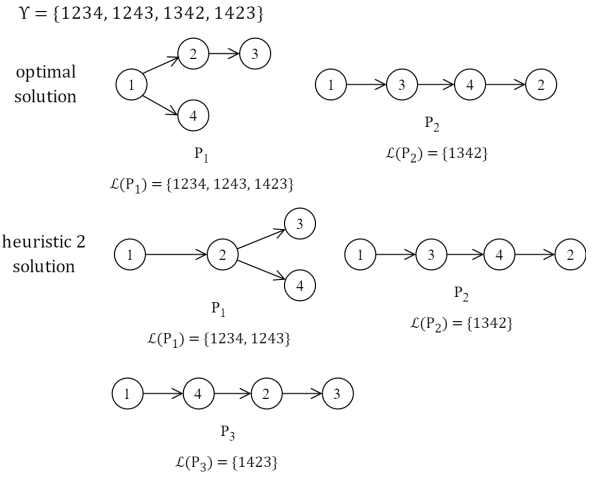

Fig. 8. Tree Posets with non-sequential linear extensions

Additionally, we have observed that there are input sets which have optimal solutions that have intersecting linear extensions. Consider, for instance, the input set of linear orders depicted in Figure 9, where the optimal solution consists of two tree posets with an intersecting linear extension – 1324. Since Heuristic 2 removes linear orders from Υ that have already been covered by a tree poset, the heuristic is guaranteed to always return a feasible, non-optimal solution to input sets characterized by optimal solutions with intersecting linear extensions.

5 Conclusion

In this study, we have formulated two heuristics to the Tree Poset Cover Problem – Heuristic 1 and Heuristic 2. We have also shown that for any input instance of the problem, both algorithms always return a feasible solution in $O(m^3n^2)$

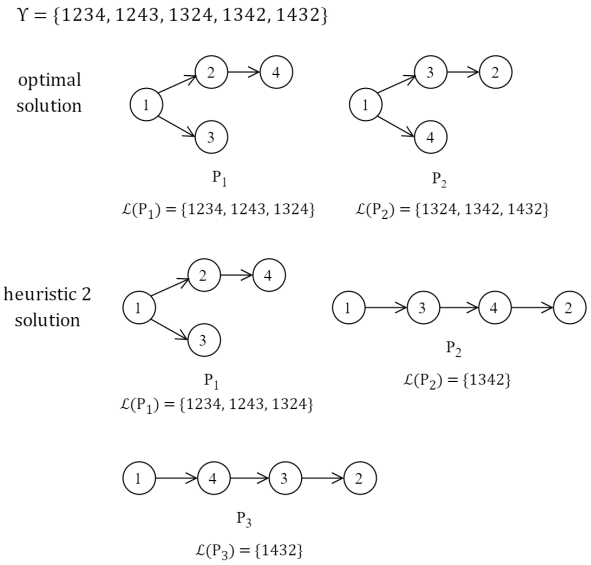

Fig. 9. Tree Posets with intersecting linear extension

and $O(m^3n)$, respectively. Hence, the two heuristics formulated can be classified as approximation algorithms. Based on time complexity, Heuristic 2 is more efficient. The first heuristic was developed based on the technique of combining posets while the second heuristic was developed by utilizing an existing $O(mn)$-time algorithm for the 1-Tree Poset Cover Problem [10].

Furthermore, we have tested each heuristic on four different input files and determined how close the heuristic solution is to the optimal solution by getting the approximation ratio based on the empirical data. Based on empirical results, Heuristic 2 provides a better quality of solution than Heuristic 1 does. This superiority stems from the fact that Heuristic 2 adopts a top-down approach, iteratively constructing a tree poset using all linear orders, and iteratively removing the last linear order if the constructed tree poset does not cover the current input subset, whereas Heuristic 1 focuses only on improving the solution by combining compatible tree posets. In addition to an existing condition for combining posets, we have identified another case where it is possible to combine two tree posets into a single one.

Future studies on this topic can focus on finding the approximation ratio of the two algorithms. The devised approximation algorithms, albeit polynomial, can still be slow in practice. Thus, future work can also focus on enhancing the efficiency of Heuristic 1 by addressing the component that currently requires $O(m^3)$ time. One potential avenue for improvement is exploring the possibility of combining posets in groups larger than two, as opposed to the conventional approach of combining only two posets. This could potentially reduce the time

complexity and enhance the overall performance of the heuristic. The efficiency of Heuristic 2 can also be improved by addressing the same component that incurs $O(m^3)$ time and determining if it can be efficiently be done without iteratively constructing a tree poset that could potentially cover a subset of the input.

6 Acknowledgments

Willie N. Coronel Jr would like to express heartfelt gratitude to the Department of Science and Technology - Science Education Institute (DOST-SEI) for their unwavering support throughout this research study.

References

1. Heath, L. S., Nema, A. K. (2013). The poset cover problem.
2. Rayward-Smith, Vic Osman, Ibrahim Reeves, Colin Simth, G.. (1996). Modern Heuristic Search Methods.
3. Fernandez, P. (2008). On the complexities of the block sorting and poset cover problems (Doctoral dissertation, PhD thesis, Ateneo de Manila University).
4. Hromkovič, J. (2013). Algorithmics for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics. Springer Science & Business Media.
5. Ordanel, I., Adorna, H. (2017). Two Approximation Algorithms for the Poset Cover Problem. In Proceedings of the 17th Philippine Computing Science Congress (PCSC 2017) (pp. 179–184).
6. Ordanel, I., Adorna, H. (2018). Optimal Deterministic Algorithm for Hammock (2, 2)-Poset Cover Problem. Philipp J Sci, 147(7), 733-748.
7. Ordanel, I., Adorna, H., Clemente, J. (2019). Approximation of two simple variations of the Poset Cover Problem. In Theory and Practice of Computation: Proceedings of Workshop on Computation: Theory and Practice WCTP2017 (pp. 1-14).
8. Ordanel, I., Fernandez Jr, P., Adorna, H. (2019). On Finding Two Posets that Cover Given Linear Orders. Algorithms, 12(10), 219.
9. Ordanel, I., Fernandez Jr, P., Adorna, H. (2021). A polynomial time algorithm for the 2-Poset Cover Problem. Information Processing Letters, 169, 106106.
10. Ordanel, I. D., Fernandez, P. L. (2011). Reconstructing a Tree Poset from Linear Extensions. Philippine Information Technology Journal, 4(2).
11. Sanchez, G. A., Fernandez, P. L., Vergara, J. P. (2014). Some heuristics for the 2-poset cover problem. Philippine Computing Journal, 9, 26-32.
12. Tan, M. J. (2010). Polynomial-time solutions to three poset cover problem variations (Doctoral dissertation, Ateneo de Manila University).
13. Tahami, H., Fakhravar, H. (2022). A literature review on combining heuristics and exact algorithms in combinatorial optimization. European Journal of Information Technologies and Computer Science, 2(2), 6-12.