algorithms

MDPI

Article

On Finding Two Posets that Cover Given Linear Orders

Ivy Ordanel [1,*], Proceso Fernandez, Jr. [2] and Henry Adorna [1]

[1] Department of Computer Science, University of the Philippines Diliman, Quezon City 1101, Philippines; hnadorna@up.edu.ph

[2] Department of Information Sytems and Computer Science, Ateneo De Manila University, Quezon City 1108, Philippines; pfernandez@ateneo.edu

* Correspondence: ivyordanel@gmail.com

Received: 13 August 2019; Accepted: 17 October 2019; Published: 19 October 2019

Abstract: The Poset Cover Problem is an optimization problem where the goal is to determine a minimum set of posets that covers a given set of linear orders. This problem is relevant in the field of data mining, specifically in determining directed networks or models that explain the ordering of objects in a large sequential dataset. It is already known that the decision version of the problem is NP-Hard while its variation where the goal is to determine only a single poset that covers the input is in P. In this study, we investigate the variation, which we call the 2-Poset Cover Problem, where the goal is to determine two posets, if they exist, that cover the given linear orders. We derive properties on posets, which leads to an exact solution for the 2-Poset Cover Problem. Although the algorithm runs in exponential-time, it is still significantly faster than a brute-force solution. Moreover, we show that when the posets being considered are tree-posets, the running-time of the algorithm becomes polynomial, which proves that the more restricted variation, which we called the 2-Tree-Poset Cover Problem, is also in P.

Keywords: partial order; poset; linear extensions; algorithm; complexity

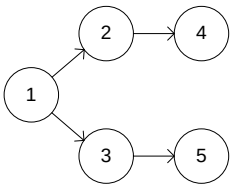
1. Introduction

A poset is a fundamental concept in mathematics that formalizes the notion of ordering in a collection of objects. Formally, a poset $P = (V, \leq_P)$ is a pair consisting of a finite set V and a binary relation $\leq_P \subseteq V \times V$ that is reflexive, antisymmetric and transitive. In this paper, we only consider strict posets, written here as $P = (V, <_P)$, where the binary relation is antisymmetric and transitive but irreflexive. An example of a strict poset is shown in Figure 1a. From here on, all posets being discussed refer to strict posets.

$V = \{1,2,3,4,5\}$

$<_P = \{(1,2),$
$(1,3),(1,4),$
$(1,5),(2,4),$
$(3,5)\}$

a. $P = (V, <_P)$



b. $H(P)$

$\mathcal{L}(P) = \{$
$(1,2,3,4,5),$
$(1,2,3,5,4),$
$(1,2,4,3,5),$
$(1,3,2,4,5),$
$(1,3,2,5,4),$
$(1,3,5,2,4)\}$

c. $\mathcal{L}(P)$

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

Every poset $P = (V, <_P)$ also corresponds to a directed acyclic graph (DAG) $G = (V, A)$ having the vertex set V and the edge set $A = \{(u,v) | (u,v) \in <_P\}$. However, a poset is more commonly

illustrated using a Hasse diagram that corresponds to the transitive reduction of the DAG. Figure 1b shows the Hasse diagram $H(P)$ of the poset P defined in Figure 1a.

The binary relation in a poset is called a partial order since not all pairs need to be related. For the case wherein all pairs are related, the binary relation is called a total order, and the poset is said to be a totally ordered set or a linear order. Formally, a linear order $L = (V, <_L)$ is a poset where for all distinct elements $u, v \in V$, either $u <_L v$ or $v <_L u$. Basically, we can treat a linear order as a permutation of the elements in V. Moreover, a linear order $L = (V, <_L)$ is said to be a linear extension of poset P if and only if $<_P \subseteq <_L$. The set of all linear extensions of P is denoted by $\mathcal{L}(P)$. For example, Figure 1c shows the set of linear extensions of the poset P in Figure 1a.

Generating the linear extension of a poset, which is also equivalent to getting all the topological sorting of a directed acyclic graph, is a well studied problem. Algorithms [1,2] on this show that we can generate each successive linear extension or topological sorting in polynomial-time. The reverse problem where the given is a set of linear extensions, or more technically a set of linear orders, and the goal is to determine the set of posets that generate the given linear orders is the Poset Cover Problem. Formally, the Poset Cover Problem is defined as follows [3]:

POSET COVER PROBLEM
INSTANCE: A set $\mathrm{Y} = \{L_1, L_2, ..., L_m\}$ of linear orders over the set $V = \{1, 2, 3, ..., n\}$.
SOLUTION: A set $P^* = \{P_1, P_2, ..., P_k\}$ of posets where $\bigcup_{P_i \in P^*} \mathcal{L}(P_i) = \mathrm{Y}$ and k is minimum.

The Poset Cover Problem finds application in data mining where there is a massive set of sequential data and the goal is to determine directed networks that explain the ordering of objects in the sequential data. A simple example of this is in marketing, when the business wants to generate a directed process model from the logs of customers' purchases of their products. The logs can be treated as linear orders of the products. Determining directed process models, which explains the purchasing behavior of the customers, can then be represented as getting a set of posets that generate the linear orders (logs). The process model can then be used to predict other customers' future purchases, and the business could develop marketing campaigns from it. Similar to this kind of data mining, the problem is also relevant in other areas such as in neuroscience [4], chemical engineering [5], epidemiology [6], paleontology [7,8], and systems biology [9].

The decision version of the Poset Cover Problem has been shown to be NP-Hard [3]. To deepen the knowledge about the Poset Cover Problem, efforts in the past have focused on studying constrained cases of the problem and trying to draw boundaries of the cases that are in P and cases that are NP-Hard. There are essentially two ways in which the problem can be restricted or constrained. The first is to consider only a specific number of posets, say k. The problem when $k = 1$, or the 1-Poset Cover Problem, is in P [10]. The computational complexity of the problem when $k = 2$, or the 2-Poset Cover Problem, is not yet known. What have been devised are heuristics for the 2-Poset Cover Problem [11]. The other way of restricting the problem is to consider only a specific class of posets according to their Hasse diagram, such as a tree-poset. The poset in Figure 1 is an example of a tree-poset since its Hasse diagram is a tree. The restricted cases of the Poset Cover Problem are similarly important as there may also be instances wherein the class of posets to be reconstructed are known. For example in paleontology, the goal is to construct evolutionary ordering of fossil sites from sequential data about the taxa that occur in each site [7,8]. Evolutionary orderings are usually expressed using trees because evolution starts with an origin and branches out to descendants. Other classes of posets that have been studied are hammock posets and leveled posets [10]. The 2-Tree-Poset Cover Problem then is a contrained variation where the goal is to determine if there exist two tree-posets that cover exactly all of the given set of linear orders.

In our study, we derived properties on posets, which lead to an exact solution for the 2-Poset Cover Problem. The algorithm runs in exponential time. However, if the posets to be considered are

tree-posets, the running time of the algorithm becomes polynomial. This proves that the 2-Tree-Poset Cover Problem is also in P.

2. Definitions

We first define here the terms and notations used in the discussion of results.

Definition 1. ancestors(v,P)

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. descendants(v,P)

Given a poset $P = (V, <_P)$ and $v \in V$, the $descendants(v, P)$ is the set of elements in poset P that succeeds v, i.e., $descendants(v, P) = \{d \in V | v <_P d\}$.

Definition 3. cover relation $\prec_P$

Given a poset $P = (V, <_P)$, its cover relation is $\prec_P = \{(u, v) | u <_P v$ and there is no $w \in V$ where $u <_P w <_P v\}$.

Definition 4. cover

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 5. Hasse Diagram of Poset $H(P)$

A poset $P = (V, <_P)$ corresponds to a Hasse diagram $H(P)$, which is a directed acyclic graph G with the elements in V as nodes and pairs in the cover relation of P as edges, i.e., $G = (V, \prec_P)$.

Definition 6. Tree Poset

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 7. parent(v,P)

Given a tree-poset $P = (V, <_P)$ and $v \in V$ such that v is not the root node, $parent(v, P)$ is the element u that covers v, i.e., $parent(v, P) = u$ if and only if $u \prec_P v$.

Definition 8. 2-Poset Cover Problem

Instance: A set $\Upsilon = \{L_1, L_2, ..., L_m\}$ of linear orders over the set $V = \{1, 2, 3, ..., n\}$.

Question: Does there exist a pair of distinct posets $P_1 = (V, <_{P_1})$ and $P_2 = (V, <_{P_2})$ such that $\mathcal{L}(P_1) \cup \mathcal{L}(P_2) = \Upsilon$ and $\mathcal{L}(P_1)$ is neither a subset nor a superset of $\mathcal{L}(P_2)$?

Definition 9. 2-Tree-Poset Cover Problem

Instance: A set $\Upsilon = \{L_1, L_2, ..., L_m\}$ of linear orders over the set $V = \{1, 2, 3, ..., n\}$.

Question: Does there exist a pair of distinct tree-posets $P_1 = (V, <_{P_1})$ and $P_2 = (V, <_{P_2})$ such that $\mathcal{L}(P_1) \cup \mathcal{L}(P_2) = \Upsilon$ and $\mathcal{L}(P_1)$ is neither a subset nor a superset of $\mathcal{L}(P_2)$?

Definition 10. comparable and incomparable

Given a poset $P = (V, <_P)$ and elements $u, v \in V$, u and v are comparable, denoted by $u \perp v$ if $u <_P v$ or $v <_P u$. Otherwise, u and v are incomparable, denoted by $u ||_P v$.

3. Theoretical Bases

Before we discuss the algorithms, we present the following lemmas and theorems that serve as bases in devising the algorithms.

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

Proof. Let the linear order L be a linear extension of P_2, i.e., $L \in \mathcal{L}(P_2)$. By definition of linear extension, $<_{P_2} \subseteq <_L$. Since $<_{P_1} \subseteq <_{P_2}$, then by transitivity, $<_{P_1} \subseteq <_L$. This implies that $L \in \mathcal{L}(P_1)$. Hence, $\mathcal{L}(P_2) \subseteq \mathcal{L}(P_1)$. □

Given a pair that is incomparable in P, say $a \parallel_P b$, we know that there are some linear extensions where $a < b$ while $b < a$ in the remaining ones. In other words, we can partition Y into $\mathrm{Y}_1 = \{L \in \mathcal{L}(P) | a <_L b\}$ and $\mathrm{Y}_2 = \{L \in \mathcal{L}(P) | b <_L a\}$. The next theorem shows that there exists posets $P_1 = (V, <_{P_1})$, $P_2 = (V, <_{P_2})$ that cover Y_1 and Y_2, respectively. Moreover, we can derive the relationship of $<_{P_1}$ and $<_{P_2}$ from $<_P$, which is also given in the following theorem.

Theorem 1. *Consider a poset* $P = (V, <_P)$ *and distinct elements* $a, b \in V$ *where* $a || _P b$. *Let* $\mathrm{Y}_1 = \{L \in \mathcal{L}(P) | a <_L b\}$ *and* $\mathrm{Y}_2 = \{L \in \mathcal{L}(P) | b <_L a\}$. *Moreover, let*

$$<_A = \{(x, y) | x \in \{a\} \cup ancestors(a, P), y \in \{b\} \cup descendants(b, P)\} \quad (1)$$

$$<_B = \{(x, y) | x \in \{b\} \cup ancestors(b, P), y \in \{a\} \cup descendants(a, P)\} \quad (2)$$

Then, there exist posets $P_1 = (V, <_{P_1})$ *and* $P_2 = (V, <_{P_2})$ *where* $<_{P_1} = <_P \cup <_A$ *and* $<_{P_2} = <_P \cup <_B$ *such that* $\mathcal{L}(P_1) = \mathrm{Y}_1$ *and* $\mathcal{L}(P_2) = \mathrm{Y}_2$. *Moreover, in terms of cover relation,*

$$\prec_{P_1} = \{(a, b)\} \cup \prec_P \setminus (\{(u, b) \in \prec_P | u <_P a\} \cup \{(a, v) \in \prec_P | b <_P v\}) \quad (3)$$

$$\prec_{P_2} = \{(b, a)\} \cup \prec_P \setminus (\{(u, a) \in \prec_P | u <_P b\} \cup \{(b, v) \in \prec_P | a <_P v\}). \quad (4)$$

To illustrate the theorem, consider the posets P, P_1 and P_2 with Hasse diagrams in Figure 2. Since $3 || _P 4$, let us take elements 3 and 4 as a and b in the theorem, respectively. If we generate $\mathcal{L}(P)$, we have the following linear extensions.

(1, 2, 3, 4, 5, 6)	(1, 2, 4, 3, 5, 6)
(1, 2, 3, 4, 6, 5)	(1, 2, 4, 3, 6, 5)
(1, 2, 3, 5, 4, 6)	(2, 1, 4, 3, 5, 6)
(2, 1, 3, 4, 5, 6)	(2, 1, 4, 3, 6, 5)
(2, 1, 3, 4, 6, 5)	(2, 4, 1, 3, 5, 6)
(2, 1, 3, 5, 4, 6)	(2, 4, 1, 3, 6, 5)

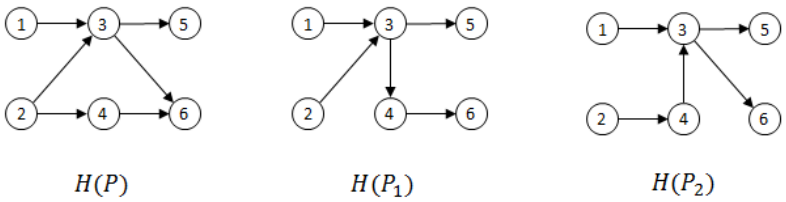


Figure 2. Example of Theorem 1.

We can partition $\mathcal{L}(P)$ into two - $\mathrm{Y}_1 = \{L \in \mathcal{L}(P) | 3 <_L 4\}$ and $\mathrm{Y}_2 = \{L \in \mathcal{L}(P) | 4 <_L 3\}$. Clearly, Y_1 contains all the linear extensions in the left column while Y_2 contains those in the right

column. From the theorem, there exist posets P_1 and P_2 that cover Y_1 and Y_2, respectively. In our example, they are the posets $P_1 = (V, <_{P_1})$ and $P_2 = (V, <_{P_2})$ in Figure 2. Verify that $\mathcal{L}(P_1) = Y_1$ and $\mathcal{L}(P_2) = Y_2$. Moreover,

$$\begin{aligned}
<_{P_1} = & \quad <_P \cup \quad \{(x,y) | x \in \{3\} \cup ancestors(3,P), y \in \{4\} \cup descendants(4,P)\} \\
= & \quad \{(1,3),(2,3),(2,4),(3,5),(3,6),(4,6),(1,5),(1,6),(2,5),(2,6)\} \quad \cup \\
& \quad \{(3,4),(3,6),(1,4),(1,6),(2,4),(2,6)\} \\
= & \quad \{(1,3),(2,3),(2,4),(3,5),(3,6),(4,6),(1,5),(1,6),(2,5),(2,6),(3,4),(1,4)\}
\end{aligned}$$

$$\begin{aligned}
<_{P_2} = & \quad <_P \cup \quad \{(x,y) | x \in \{4\} \cup ancestors(4,P), y \in \{3\} \cup descendants(3,P)\} \\
= & \quad \{(1,3),(2,3),(2,4),(3,5),(3,6),(4,6),(1,5),(1,6),(2,5),(2,6)\} \quad \cup \\
& \quad \{(4,3),(4,5),(4,6),(2,3),(2,5),(2,6)\} \\
= & \quad \{(1,3),(2,3),(2,4),(3,5),(3,6),(4,6),(1,5),(1,6),(2,5),(2,6),(4,3),(4,5)\}
\end{aligned}$$

In terms of cover relation, which is the transitive reduction of $<_{P_1}$ and $<_{P_2}$, we have the following:

$$\begin{aligned}
\prec_{P_1} = & \quad \{(3,4)\} \quad \cup \prec_P \setminus \quad (\{(u,4) \in \prec_P | u <_P 3\} \cup \{(3,v) \in \prec_P | 4 <_P v\}) \\
= & \quad \{(3,4)\} \quad \cup \{(1,3),(2,3),(2,4),(3,5),(3,6),(4,6)\} \setminus (\{(2,4)\} \cup \{(3,6)\}) \\
= & \quad \{(3,4),(1,3),(2,3),(3,5),(4,6)\}
\end{aligned}$$

$$\begin{aligned}
\prec_{P_2} = & \quad \{(4,3)\} \quad \cup \prec_P \setminus \quad (\{(u,3) \in \prec_P | u <_P 4\} \setminus \{(4,v) \in \prec_P | 3 <_P v\}) \\
= & \quad \{(4,3)\} \quad \cup \{(1,3),(2,3),(2,4),(3,5),(3,6),(4,6)\} \setminus (\{(2,3)\} \cup \{(4,6)\}) \\
= & \quad \{(4,3),(1,3),(2,4),(3,5),(3,6)\}
\end{aligned}$$

Proof. We first show that $P_1 = (V, <_{P_1})$ where $<_{P_1} = <_P \cup <_A$ is a poset. Since P is a poset, then we know that $<_P$ is irreflexive, antisymmetric and transitive. We also know that a and b are distinct elements. An element x cannot be both in $ancestors(a,P)$ and $descendants(b,P)$, otherwise, a and b are related in $<_P$. Hence, we can say that $<_A$ is also irreflexive and antisymmetric. To show that $<_{P_1}$ is transitive, suppose $(x,y), (y,z) \in <_{P_1}$, then we have the following cases:

- $(x,y) \in <_P$ and $(y,z) \in <_P$
  By transitive property of poset P, $(x,z) \in <_P$. Hence, $(x,z) \in <_{P_1}$
- $(x,y) \in A$ and $(y,z) \in A$
  By the definition of A, $y \in \{a\} \cup ancestors(a,P)$ and $y \in \{b\} \cup descendants(b,P)$. It cannot be that $y = a$ or $y = b$ because of the assumption that a and b are distinct. It cannot also be that $y \in ancestors(a,P)$ and $y \in descendants(b,P)$, otherwise the assumption that $a||_P b$ is violated. Hence, we can say that this case is not possible.
- $(x,y) \in <_P$ and $(y,z) \in A$
  By the definition of A, $y \in \{a\} \cup ancestors(a,P)$ and $z \in \{b\} \cup descendants(b,P)$. If $y = a$, then $x \in ancestors(a,P)$. Hence $(x,z) \in A$ and thus, $(x,z) \in <_{P_1}$. On the other hand, if $y \in ancestors(a,P)$, then $(y,a) \in <_P$. By the transitive property of $<_P$, $(x,a) \in <_P$. In other words, $x \in ancestors(a,P)$. Hence, $(x,z) \in A$ and thus, $(x,z) \in <_{P_1}$.
- $(x,y) \in <_A$ and $(y,z) \in <_P$
  By the definition of A, $x \in \{a\} \cup ancestors(a,P)$ and $y \in \{b\} \cup descendants(b,P)$. If $y = b$, then $z \in descendants(b,P)$. Hence $(x,z) \in A$ and thus, $(x,z) \in <_{P_1}$. On the other hand, if $y \in descendants(b,P)$, then $(b,y) \in <_P$. By the transitive property of $<_P$, $(b,z) \in <_P$. In other words, $z \in descendants(b,P)$. Hence $(x,z) \in A$ and thus, $(x,z) \in <_{P_1}$.

From these cases, we can say that $<_{P_1}$ is transitive. Hence, $<_{P_1}$ is a poset. In a similar way, we can also show that $<_{P_2}$ is a poset.

Next, we show that $\mathcal{L}(P_1) = \mathrm{Y}_1$ by showing that $\mathcal{L}(P_1) \subseteq \mathrm{Y}_1$ and $\mathrm{Y}_1 \subseteq \mathcal{L}(P_1)$.

To prove the first direction, suppose $L \in \mathcal{L}(P_1)$. Then $<_{P_1} \subseteq <_L$. This implies that $a <_L b$. Since $<_P \subseteq <_{P_1}$, then we can also say that $<_P \subseteq <_L$. This means, $L \in \mathcal{L}(P)$. Hence, $L \in \mathrm{Y}_1$.

To prove the other direction, suppose $L \in \mathrm{Y}_1$. Then, $L \in \mathcal{L}(P)$ and $a <_L b$. Now, let $x <_{P_1} y$. It suffices to show that $x <_L y$. This clearly holds if $x <_P y$, since $L \in \mathcal{L}(P)$. Otherwise, $x <_A y$, which implies that ($x = a$ or $x <_P a$) and ($b = y$ or $b <_P y$), whence ($x = a$ or $x <_L a$) and ($b = y$ or $b <_L y$). In any case, since $a <_L b$, we get $x <_L y$ by transitivity.

In a similar way, it can also be shown that $\mathcal{L}(P_2) = \mathrm{Y}_2$.

Now, let us determine the cover relation $\prec_{P_1} \subseteq <_{P_1}$. We first determine which pairs $(x, y) \in <_A$ are cover pairs in P_1. If $(x, y) = (a, b)$, we know that by definition of $<_A$, there exists no $w \in V$ such that $(a, w) \in <_A$ and $(w, b) \in <_A$, otherwise $a <_P b$. Since $a \parallel_P b$, then we also know that there exists no $w \in V$ such that $(a, w) \in <_A$ and $(w, b) \in <_A$. Hence, $(a, b) \in \prec_{P_1}$. On the other hand, any other value of (x, y) in $<_A$ will lead to ($x <_{P_1} a <_{P_1} b <_{P_1} y$), ($x = a <_{P_1} b <_{P_1} y$) or ($x <_{P_1} a <_{P_1} b = y$), which all imply that $(x, y) \notin \prec_{P_1}$. Hence, the only pair in $<_A$ that is covered in P_1 is (a, b). Now, we know that $<_P$ corresponds to $\prec_P$. Hence, $\prec_{P_1}$ is the transitive reduction of $\{(a, b)\} \cup \prec_P$. To get the transitive reduction, we have to elimate pairs in $\prec_P$ that are not in $\prec_{P_1}$ because the addition of (a, b) no longer makes them covered. These are the following:

- $u \prec_P b$ where $u <_P a$
  In this case $u <_{P_1} a <_{P_1} b$, hence, $(u, b) \notin \prec_{P_1}$.
- $(a \prec_P v)$ where $b <_P v$
  In this case $a <_{P_1} b <_{P_1} v$, hence, $(a, v) \notin \prec_{P_1}$

Hence, $\prec_{P_1} = \{(a, b)\} \cup \prec_P \setminus \{\{(u, b) \in \prec_P | u <_P a\} \cup \{(a, v) \in \prec_P | b <_P v\}\}$. In a similar way, we can derive the formula for $\prec_{P_2}$. □

Corollary 1. *If the poset $P = (V, <_P)$ in Theorem 1 is a tree-poset, then Equations (3) and (4) become Equations (5) and (6), respectively.*

$$\prec_{P_1} = \begin{cases} \{(a, b)\} \cup \prec_P, & \text{if } parent(b, P) \ ||_P \ a \\ \{(a, b)\} \cup \prec_P \setminus \{(parent(b, P), b)\}, & \text{if } parent(b, P) \ <_P \ a \end{cases} \tag{5}$$

$$\prec_{P_2} = \begin{cases} \{(b, a)\} \cup \prec_P, & \text{if } parent(a, P) \ ||_P \ b \\ \{(b, a)\} \cup \prec_P \setminus \{(parent(a, P), a)\}, & \text{if } parent(a, P) \ <_P \ b \end{cases} \tag{6}$$

Proof. From Theorem 1, $\prec_{P_1} = \{(a, b)\} \cup \prec_P \setminus (\{(u, b) \in \prec_P | u <_P a\} \cup \{(a, v) \in \prec_P | b <_P v\})$. First, we show that $\{(a, v) \in \prec_P | b <_P v\} = \emptyset$ if P is a tree-poset. Suppose there exists such v where $a \prec_P v$ and $b <_P v$. Since, every non-root node in a tree-poset is covered by only one element, then only a covers v. Then it must be that $b <_P a$, so that $b <_P v$. This is a contradiction to the assumption that $a || _P b$. Hence, $\{(a, v) \in \prec_P | b <_P v\} = \emptyset$.

Next, we examine the set $\{(u, b) \in \prec_P | u <_P a\}$. If P is a tree-poset, then there exists only one possible u that covers b, which is $parent(b, P)$. Hence, we have two possibilities. If $parent(b, P) <_P a$ then $\prec_{P_1} = \{(a, b)\} \cup \prec_P \setminus \{(parent(b, P), b)\}$. Note that $a \not<_P parent(b, P)$, otherwise $a <_P b$. Hence, the only other condition is when $parent(b, P) || _P u$. With this condition, Equation (3) then becomes $\prec_{P_1} = \prec_P \cup \{(a, b)\}$.

We can prove Equation (6) in a similar way. □

4. Algorithm for the 2-Poset Cover Problem

As mentioned earlier, there is already a polynomial-time solution for the 1-Poset Cover Problem [10]. We can determine in $O(mn^2)$ time, where m is the number of linear orders over a base set of n elements, the poset that covers a given set of linear orders. Let us denote the algorithm for the 1-Poset Cover Problem as GeneratePoset. The input to GeneratePoset is a set of linear orders and it returns a poset, if there exists such, that covers the input; otherwise it returns null. Now, for the 2-Poset Cover Problem with input Y, a brute-force algorithm is then to determine Y_1, $Y_2 \subseteq Y$ such that $Y_1 \cup Y_2 = Y$ and then use GeneratePoset to determine if there is a poset that covers Y_1 and a poset that covers Y_2. Since there are 3^m possibilities for values of Y_1 and Y_2, the brute-force algorithm entails a running time of $O(3^m mn^2)$.

Theorem 1 suggests a strategy of partitioning Y. Given a pair $(a, b) \in V \times V$, we can partition Y into $Y_1 = \{L \in Y | a <_L b\}$ and $Y_2 = \{L \in Y | b <_L a\}$. Suppose there exist two posets, say $P = (V, <_P)$ and $P' = (V, <_{P'})$ that cover Y. If $(a, b) \in <_P$ and $(b, a) \in <_{P'}$, then we just have partitioned Y perfectly because $Y_1 = \mathcal{L}(P)$ and $Y_2 = \mathcal{L}(P')$. Hence, we can determine P and P' using GeneratePoset with Y_1 and Y_2 as inputs, respectively. This is executed in Lines 7 and 8 of Algorithm 1. However, such a pair does not always exist. There can be also instances where $a \perp_P b$ and $a||_{P'} b$ and vice versa—there also exists a different pair, say (c, d) where $c||_P d$ and $c \perp_{P'} d$. Without lost of generality on $a \perp_P b$, suppose $a <_P b$. Then, if we apply the same partitioning strategy, $Y_1 = \mathcal{L}(P) \cup \{L \in \mathcal{L}(P') | a <_L b\}$ and $Y_2 = \{L \in \mathcal{L}(P') | b <_L a\}$. We are not sure if there exists a poset that covers Y_1, however, we are sure from Theorem 1 that there exists a poset that covers Y_2, say P_2. Moreover, with the use of the equations in the Theorem, we can determine or reconstruct the relation of P' from P_2. These are executed in Lines 22–32. We can also determine P in a similar way with the pair (c, d). This is executed in Lines 10–20 in the Algorithm.

Theorem 2. *Algorithm 1 produces a solution to the 2-Poset Cover Problem in* $O(2^n mn + mn^4 + m^2)$ *time.*

Proof. Let P and P' be the two different posets that cover Y, i.e., $\mathcal{L}(P) \cup \mathcal{L}(P') = Y$. In the following, we want to show that P and P', if there exist such, can be determined and generated by the 2-Poset Cover Algorithm.

With the assumption that $P \neq P'$ means also that $<_P \neq <_{P'}$, then there must exist, without loss of generality, at least one (a, b) such that $(a, b) \in <_P$ but $(a, b) \notin <_{P'}$. On the other hand, there must also exist (c, d) such that $(c, d) \in <_{P'}$ but $(c, d) \notin <_P$, otherwise, $<_{P'} \subset <_P$ and by Lemma 1, Y can be covered by a single poset P'.

Let us first take the case where $(a, b) \in <_P$ but $(a, b) \notin <_{P'}$. With this, we have two further cases.

1. $(b, a) \in <_{P'}$
   In this case, $Y_1 = \mathcal{L}(P)$ and $Y_2 = \mathcal{L}(P')$. Hence, posets $P_1 = P$ and $P_2 = P'$ can be generated in Lines 5 and 6, respectively and are returned in Line 8.
2. $(b, a) \notin <_{P'}$
   In this case, $a||_{P'} b$. Hence, there exist linear extensions of P' where $a < b$ and linear extensions where $b > a$. Let $E = \{L \in \mathcal{L}(P') | a <_L b\}$ and $F = \{L \in \mathcal{L}(P') | b <_L a\}$. This implies that $Y_1 = \mathcal{L}(P) \cup E$ and $Y_2 = F$. Let P_1 and P_2 be the return of GeneratePoset for Y_1 and Y_2 in Lines 4 and 5, respectively.

   We know that P_2 is not null because from Theorem 1 there exists a poset that covers Y_2. Moreover, we can also reconstruct P' from P_2 using Equation (4), i.e., $\prec_{P'} = \prec_{P_2} \setminus \{(b, a)\} \cup \{C \cup D\}$ where $C = \{(u, a) \in \prec_{P'} | u <_{P'} b\}$ and $D = \{(b, v) \in \prec_{P'} | a <_{P'} v\}$.

   In reconstructing P' from P_2, let P_3 be our working poset. First, we let $\prec_{P_3} = \prec_{P'} \setminus \{(b, a)\}$ and if $\mathcal{L}(P_3) \subset Y$, then we can already disregard the components $\{C \cup D\}$ of $\prec_{P'}$. This is because $\prec_{P_2} \setminus \{(b, a)\} \subset \prec_{P_2} \setminus \{(b, a)\} \cup \{C \cup D\}$. With Lemma 1, whatever set of linear orders that are covered by the poset with the later cover relation can be already covered by the poset with the

former cover relation. These are executed with the "If" condition in Lines 23–24. Otherwise (the else part), we have to determine C and D. However, in the Algorithm, what we have only generated and know is the relation of P_2. We do not know P' yet hence we also do not know C and D. However, we know that $C \subseteq \{(u,b) \in <_{P'}\}$ and $D \subseteq \{(a,v) \in <_{P'}\}$, and with Equation (2), $(u,b) \in <_{P_2}$ and $(a,v) \in <_{P_2}$. Hence, $C \subseteq A$ where $A = \{(u,b) \in <_{P_2}\}$ and $D \subseteq B$ where $B = \{(a,v) \in <_{P_2}\}$. Hence, we can try all possible subsets of A and B for $\prec_{P_3}$. This is what we have done in Lines 26–32. Every possible poset from each combination is a candidate poset and added to P^*.

Algorithm 1: 2-Poset Cover Algorithm

Input: A set $\mathrm{Y} = \{L_1, L_2, ..., L_m\}$ of linear orders on $V = \{1, 2, ..., n\}$
Output: A pair of posets $\{P, P'\}$, $P \neq P'$ where $\mathcal{L}(P) \cup \mathcal{L}(P') = \mathrm{Y}$ and $\mathcal{L}(P)$ is neither a subset nor a superset of $\mathcal{L}(P')$, if there exist such.

1 $P^* := \emptyset$
2 **for** *each pair* $(a,b) \in V \times V$ **do**
3 $\quad \mathrm{Y}_1 := \{L \in \mathrm{Y} | a <_L b\}$
4 $\quad \mathrm{Y}_2 := \{L \in \mathrm{Y} | b <_L a\}$
5 $\quad P_1 :=$ GeneratePoset(Y_1)
6 $\quad P_2 :=$ GeneratePoset(Y_2)
7 $\quad$ **if** $P_1 \neq$ *null and* $P_2 \neq$ *null* **then**
8 $\quad\quad$ **return** $\{P_1, P_2\}$
9 $\quad$ **else if** $P_1 \neq$ *null* **then**
10 $\quad\quad P_3 := (V, <_{P_3})$ where $\prec_{P_3} = \prec_{P_1} \setminus \{(a,b)\}$
11 $\quad\quad$ **if** $\mathcal{L}(P_3) \subset \mathrm{Y}$ **then**
12 $\quad\quad\quad P^* := P^* \cup \{P_3\}$
13 $\quad\quad$ **else**
14 $\quad\quad\quad A := \{u \in V | u <_{P_1} a\}$
15 $\quad\quad\quad B := \{v \in V | b <_{P_1} v\}$
16 $\quad\quad\quad$ **for** *each* $C \subseteq A$ **do**
17 $\quad\quad\quad\quad$ **for** *each* $D \subseteq B$ **do**
18 $\quad\quad\quad\quad\quad P_3 := (V, <_{P_3})$ where $\prec_{P_3} = \prec_{P_1} \setminus \{(a,b)\} \cup \{\{(c,a) | c \in C\} \cup \{(b,d) | d \in D\}\}$
19 $\quad\quad\quad\quad\quad$ **if** $\mathcal{L}(P_3) \subset \mathrm{Y}$ **then**
20 $\quad\quad\quad\quad\quad\quad P^* := P^* \cup \{P_3\}$
21 $\quad$ **else if** $P_2 \neq$ *null* **then**
22 $\quad\quad P_3 := (V, <_{P_3})$ where $\prec_{P_3} = \prec_{P_2} \setminus \{(b,a)\}$
23 $\quad\quad$ **if** $\mathcal{L}(P_3) \subset \mathrm{Y}$ **then**
24 $\quad\quad\quad P^* := P^* \cup \{P_3\}$
25 $\quad\quad$ **else**
26 $\quad\quad\quad A := \{u \in V | u <_{P_2} b\}$
27 $\quad\quad\quad B := \{v \in V | a <_{P_2} v\}$
28 $\quad\quad\quad$ **for** *each* $C \subseteq A$ **do**
29 $\quad\quad\quad\quad$ **for** *each* $D \subseteq B$ **do**
30 $\quad\quad\quad\quad\quad P_3 := (V, <_{P_3})$ where $\prec_{P_3} = \prec_{P_2} \setminus \{(b,a)\} \cup \{\{(c,a) | c \in C\} \cup \{(b,d) | d \in D\}\}$
31 $\quad\quad\quad\quad\quad$ **if** $\mathcal{L}(P_3) \subset \mathrm{Y}$ **then**
32 $\quad\quad\quad\quad\quad\quad P^* := P^* \cup \{P_3\}$
33 **for** *each* $P_i, P_j \in P^*$, $P_i \neq P_j$ **do**
34 $\quad$ **if** $\mathcal{L}(P_i) \cup \mathcal{L}(P_j) = \mathrm{Y}$ **then**
35 $\quad\quad$ **return** $\{P_i, P_j\}$
36 **return** null

We can also do the same in determining P when the iteration evaluates the pair (c,d) where $(c,d) \notin <_P$ but $(c,d) \in <_{P'}$.

Lastly, to get P and P' in P^*, we can try all possible pairs of posets that exactly cover Y in Lines 33–35.

Now, we determine the running time complexity of the algorithm.

The first and outermost for-loop iterates in $O(n^2)$. One dominating execution inside it is the call to GeneratePoset, which runs in $O(mn^2)$ [10]. Another dominating part is the two inner for-loops, in Lines 16–20 or Lines 28–32, that iterates through all the pairwise combinations of subsets A and subsets of B. Note that an element cannot be both an ancestor and descendant of a or b. Hence $|A + B| < n$. If $|A| = t$, then $|B|$ is at most $n - t$. Hence, there are $2^t \times 2^{n-t} = 2^n$ pairs of their subsets. Inside the two inner for-loops is a statement that gets all linear extensions of poset, i.e., $\mathcal{L}(P_3)$. This can be done in $O(mn)$ by using the algorithm of Pruesse and Ruskey [1]. Hence, the total running time of the first outermost for-loop is $O(mn^4 + mn2^n)$.

For the second outermost for-loop, $|P^*| \in O(m)$ since the smallest posets (with respect to the number of linear extensions) are the linear orders themselves. Thus, the second for-loop iterates in $O(m^2)$. Inside it is a set equality testing that can be performed in constant time [12]. Hence, the second for-loop runs in $O(m^2)$.

Thus, the total running time of the algorithm is $O(2^n mn + mn^4 + m^2)$. □

The running time of the solution for the 2-Poset Cover Problem is still exponential. But there is now an improvement from the running time of a brute-force solution, i.e. from a base of 3 to a base of 2 and from exponents in terms of $m = |\mathrm{Y}|$ to exponent in terms of $n = |V|$. Hence, we can say that for a large m, Algorithm 1 is much more efficient than a brute-force solution.

In the following, we show that when the posets are tree-posets, the running time of the algorithm becomes polynomial.

5. Algorithm for the 2-Tree-Poset Cover Problem

From Theorem 2, the part of Algorithm 1 that makes its running-time exponential is determining the latter components of Equations (3) or (4). However, with Corollary 1, the said components become simpler if the posets are tree-posets. Hence, we can then modify Algorithm 1 to specifically consider only tree-posets. The resulting algorithm is Algorithm 2. Consequently, the running time of the algorithm becomes polynomial. With these results, we can then say that the 2-Tree-Poset Cover Problem is in P.

Corollary 2. *Algorithm 2 produces a solution to the 2-Tree-Poset Cover Problem in* $O(n^5 + mn^4 + m^2)$.

Proof. The theoretical bases of Algorithm 2 is the same as Algorithm 1 for the 2-Poset Cover Problem except that we use Equations (5) and (6) for tree-posets as given in Corollary 1 instead of Equations (3) and (4) for general posets. Consequently, the running time then of the Algorithm differs on the following points.

1. B is empty and there is only one element in A that could cover v or u. This follows from Corollary 1. Hence, $|A| = O(n)$
2. There is a need to check if the poset is a tree-poset. This can be done in $O(n^2)$

Hence, the inner for-loops runs in $O(n(mn + n^2)) = O(mn^2 + n^3)$. The first outermost for-loop then run in $O(n^2(mn^2 + n^3)) = O(mn^4 + n^5)$. The running time of the second-outermost for loop remains the same, i.e., $O(m^2)$. The running time then of the algorithm is $O(n^5 + mn^4 + m^2)$. □

Algorithm 2: 2-Tree-Poset Cover Algorithm

Input: A set $\mathrm{Y} = \{L_1, L_2, ..., L_m\}$ of linear orders on $V = \{1, 2, ..., n\}$

Output: A pair of tree-posets $\{P, P'\}$, $P \neq P'$ where $\mathcal{L}(P) \cup \mathcal{L}(P') = \mathrm{Y}$ and $\mathcal{L}(P)$ is neither a subset nor a superset of $\mathcal{L}(P')$, if there exist such.

$P^* := \emptyset$
for *each pair* $(a, b) \in V \times V$ **do**
  $\mathrm{Y}_1 := \{L \in \mathrm{Y} | a <_L b\}$
  $\mathrm{Y}_2 := \{L \in \mathrm{Y} | b <_L a\}$
  $P_1 := \mathrm{GeneratePoset}(\mathrm{Y}_1)$
  $P_2 := \mathrm{GeneratePoset}(\mathrm{Y}_2)$
  if $P_1 \neq null$ *and* $P_2 \neq null$ **then**
    if P_1 *is a tree and* P_2 *is a tree* **then**
      return $\{P_1, P_2\}$
  else if $P_1 \neq null$ **then**
    $P_3 := (V, <_{P_3})$ where $\prec_{P_3} = \prec_{P_1} \setminus \{(a, b)\}$
    if $\mathcal{L}(P_3) \subset \mathrm{Y}$ *and* P_3 *is a tree* **then**
      $P^* := P^* \cup \{P_3\}$
    else
      $A := \{u \in V | u <_{P_1} a\}$
      for *each* $u \in A$ **do**
        $P_3 := (V, <_{P_3})$ $\prec_{P_3} = \prec_{P_1} \setminus \{(a, b)\} \cup \{(u, b)\}$
        if $\mathcal{L}(P_3) \subset \mathrm{Y}$ *and* P_3 *is a tree* **then**
          $P^* := P^* \cup \{P_3\}$
  else if $P_2 \neq null$ **then**
    $P_3 := (V, <_{P_3})$ where $\prec_{P_3} = \prec_{P_2} \setminus \{(b, a)\}$
    if $\mathcal{L}(P_3) \subset \mathrm{Y}$ *and* P_3 *is a tree* **then**
      $P^* := P^* \cup \{P_3\}$
    else
      $A := \{u \in V | u <_{P_2} b\}$
      for *each each* $u \in A$ **do**
        $P_3 := (V, <_{P_3})$ where $\prec_{P_3} = \prec_{P_2} \setminus \{(b, a)\} \cup \{(u, a)\}$
        if $\mathcal{L}(P_3) \subset \mathrm{Y}$ *and* P_3 *is a tree* **then**
          $P^* := P^* \cup \{P_3\}$
for *each* $P_i, P_j \in P^*$, $P_i \neq P_j$ **do**
  if $\mathcal{L}(P_i) \cup \mathcal{L}(P_j) = \mathrm{Y}$ **then**
    return $\{P_i, P_j\}$
return null

6. Conclusions

In this study, we explored the Poset Cover Problem, a hard problem that is relevant in the field of data mining. It is already known that we can determine a single poset that covers a given set of linear orders in polynomial time. In this paper, we extended the knowledge on the problem by investigating the hardness of the case where we want to determine two posets, if they exist, that cover a set of linear orders, which we called the 2-Poset Cover Problem.

We discovered properties on posets that lead to an exact solution for the 2-Poset Cover Problem. The algorithm runs in $O(2^n mn + mn^4 + m^2)$ where m is the number of linear orders over a based set of n elements. When $m > n$, the algorithm is significantly faster than a brute-force solution, which runs in $O(3^m mn^2)$. Since the algorithm runs in exponential time, the complexity class that the 2-Poset Cover Problem belongs to remains unknown. However, when the posets to be considered are tree posets, which can be treated as combinatorial models for evolutionary orderings, the running time of the algorithm becomes polynomial. Hence, this proves that the more restricted case, which we called 2-Tree-Poset Cover Problem, belongs to the computational complexity class P.

Author Contributions: Conceptualization, I.O. and P.F.J.; Formal analysis, I.O.; Investigation, I.O., P.F.J. and H.A.; Supervision, H.A.; Writing—original draft, I.O.; Writing—review and editing, P.F.J. and H.A.

Funding: This research was supported by Department of Science and Technology - Engineering Research and Development for Technology (DOST-ERDT).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Pruesse, G.; Ruskey, F. Generating linear extensions fast. *SIAM J. Comput.* **1994**, *23*, 373–386. [CrossRef]
2. Kalvin, A.D.; Varol, Y.L. On the generation of all topological sortings. *J. Algorithms* **1983**, *4*, 150–162. [CrossRef]
3. Heath, L.S.; Nema, A.K. The poset cover problem. *Open J. Discret. Math.* **2013**, *3*, 101–111. [CrossRef]
4. Lee, A.; Wilson, M. A combinatorial method for analyzing sequential firing patterns involving an arbitrary number of neurons based on relative time order. *J. Neurophysiol.* **2005**, *92*, 2555–2573. [CrossRef] [PubMed]
5. Wiggins, C.; Nemenman, I. Process pathway via time series analysis. *Exp. Mech.* **2003**, *43*, 361–370. [CrossRef]
6. Unnikrishnan, K.; Ramakrishnan, N.; Sastry, P.; Uthurusamy, R. Network reconstruction from dynamic data. *ACM Sigkdd Explor. Newsl.* **2006**, *8*, 90–91. [CrossRef]
7. Puolamäki, K.; Fortelius, M.; Mannila, H. Seriation in paleontological data: Using markov chain monte carlo methods. *PLoS Comput. Biol.* **2006**, *2*, e6. [CrossRef] [PubMed]
8. Mannila, H. Finding total and partial orders from data for seriation. *Lect. Notes Comput. Sci.* **2008**, *5255*, 16–25.
9. Arkin, A.; Sheng, P.; Ross, J. A test case of correlation metric construction of a reaction pathway from measurements. *Science* **1997**, *277*, 1275–1279. [CrossRef]
10. Fernandez, P.L.; Heath, L.S.; Ramakrishnan, N.; Tan, M.; Vergara, J.P. Mining posets from linear orders. *Discret. Math. Algorithms Appl.* **2013**, *5*, 1350030. [CrossRef]
11. Sanchez, G.A.; Fernandez, P.L.; Vergara, J.P. Some heuristics for the 2-poset cover problem. *Philipp. Comput. J.* **2014**, *9*, 26–32.
12. Yellin, D. Representing sets with constant time equality testing. *J. Algorithms* **1992**, *13*, 353–373. [CrossRef]