Philippine Journal of Science
147 (4): 725-740, December 2018
ISSN 0031 - 7683
Date Received: 02 May 2018

Optimal Deterministic Algorithm for the Hammock(2,2)-Poset Cover Problem

Ivy D. Ordanel* and **Henry N. Adorna**

Department of Computer Science, University of the Philippines Diliman, Quezon City, Philippines

The Poset Cover Problem is a hard optimization problem that has application to problems in data mining where the goal is to come up with directed acyclic networks that characterize the ordering pattern of a given set of sequential data. There have been variations to the problem, one of which is the Hammock($\frac{2,2,\ldots,2}{k}$)-Poset Cover Problem. It has been shown that the decision version of Hammock($\frac{2,2,\ldots,2}{k}$)-Poset Cover Problem is NP-complete when $k \geq 3$ but in P when $k = 1$, the complexity of the problem when $k = 2$ was left open. In this study, we focus on the variation where $k = 2$, that is the Hammock(2,2)-Poset Cover Problem, where the input is a collection of linear orders over a set and the output is a minimum set of Hammock(2,2)-Posets that covers or generates the given linear orders. We present an optimal deterministic algorithm for the problem that runs in $O(n^2m^3 + m^3M)$ where n is the size of the set, m is the number of linear orders, and M is the number of all maximum matchings of a subgraph of the transposition graph of linear orders. We also present properties that show relationships of Hammock(2,2)-Poset Cover Problem to the Set Cover Problem, Cycle Cover Problem, and Edge Cover Problem.

Key words: algorithm, partial order, poset

INTRODUCTION

Consider the ordering of different tasks in Figure 1. Suppose a teacher gives those tasks to students and the students need to finish all of them. From the graph, there are some tasks that are dependent on other tasks. For example, Tasks 2, 3, and 4 need to be accomplished first before proceeding on Task 5. There are also some tasks that do not have dependencies. For example, Task 6 is not dependent on Task 7, so it does not matter which one between them will be started first. With this ordering, one student can do all the tasks in the following order: Task 1 →Task 2 → Task 3 → Task 4 → Task 5 → Task 6 → Task 7 → Task 8. Another student can accomplish all the task as follows: Task 1→ Task 3 → Task 2 → Task 4 → Task 5 → Task 7 → Task 6 → Task 8. There are actually 12 possible ways on which a student can finish all the tasks. This is a typical scenario – an ordering is given and the flow of events must follow from them.

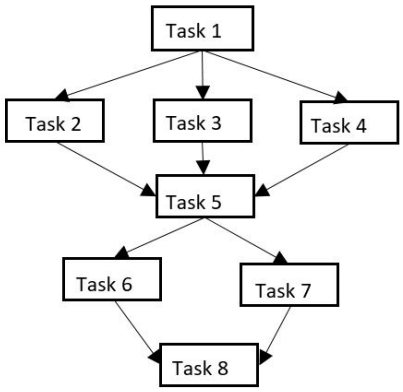

Figure 1. Ordering of tasks.

*Corresponding author: ivyordanel@gmail.com

In data mining, a relevant and more difficult problem is the reverse. What is given or available is a massive set of sequential data and the goal is to come up with networks that show the ordering pattern of the sequential data. We can see such data in many areas. In neuroscience, neuronal networks are constructed from the sequences of individual neuron firings (Lee & Wilson 2005). In chemical engineering, chemical process networks or pathways are constructed from studying sequential data of chemical concentrations (Wiggins & Nemenman 2003). In epidemiology, structures of disease spread are constructed from the sequential data of people movement from one location to another (Unnikrishnan *et al.* 2006). In paleontology, evolutionary ordering of fossil sites is constructed from sequential data about the taxa that occur in each site (Puolamäki *et al.* 2006, Mannila 2008). In systems biology, reaction systems or pathways are constructed from a sequence of concentration – measured from time to time – of the species composing the system (Arkin *et al.* 1997). In business, process models are constructed from logs of events in a workflow that are recorded sequentially (Agrawal *et al.* 1998, Der Aalst *et al.* 2004). A simple example of this is the case when the business wants to generate a process model from the logs of purchases of their products. The process model can be used to predict other customers' future purchases, and the business could develop marketing campaigns from it.

Ordering or ranking of a collection of objects or events is formalized in mathematics using the notion of a poset (partially ordered set). A poset $P = (V,<_p)$ 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 poset – 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 2a. Note that not all pairs in a poset 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 ∈ V, either $u <_L v$ or $v <_L u$. Moreover, a linear order $L = (V,<_L)$ is said to be a linear extension of poset $P = (V,<_{P_3})$ if and only if $<_p \subseteq <_L$. In this paper, we denote the set of all linear extensions of P as $\mathcal{L}(P)$. For example, Figure 2c shows the set of linear extensions of the poset P in Figure 2a.

We can now abstract the data mining problem with the concepts just discussed. A sequential data of events, which is basically a permutation of events, can be treated as a linear order. Constructing a directed network from sequential data can then be abstracted as constructing a poset that covers the input set of linear orders. If the goal is to find the minimum number of posets, then the problem becomes the Poset Cover Problem. Formally, the Poset Cover Problem is defined as follows (Heath & Nema 2013).

Poset Cover Problem

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

Solution: A set $P^* = \{P_1, P_2, \ldots, P_q\}$ of posets over the set V where $\bigcup_{P_i \in P^*} \mathcal{L}(P_i) = \Upsilon$ and q is minimum.

The decision version of the Poset Cover Problem has been shown to be NP-complete (Heath & Nema 2013). The problem was also further explored by defining some simple variations and investigating their complexity (Fernandez *et al.* 2013, Ordanel & Fernandez 2011, Sanchez *et al.* 2014). One of the variations is the Hammock($\frac{2,2,\ldots,2}{k}$)-Poset Cover Problem. This variation has also been shown to be NP-complete for $k \geq 3$ but in P if $k = 1$ (Fernandez *et al.* 2013), the complexity of the problem when $k = 2$ was left open. In this study, we present an optimal deterministic algorithm for the problem where $k = 2$ *i.e.*, the Hammock(2,2)-Poset Cover Problem. Moreover, we also present properties that show relationships of Hammock(2,2)-Poset Cover Problem to a restricted case of Set Cover Problem, Cycle Cover Problem, and Edge Cover Problem.

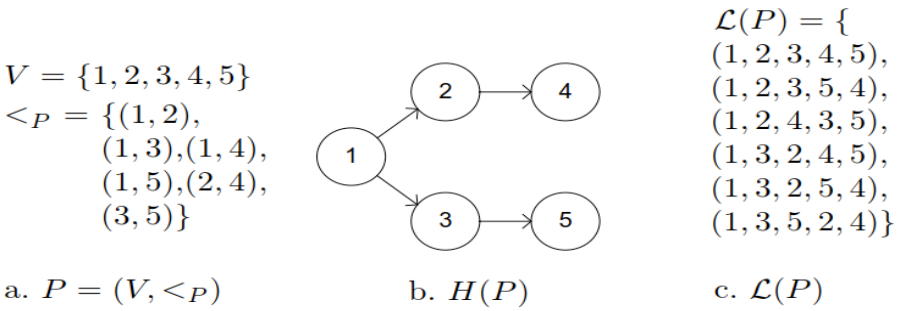

Figure 2. Poset P, its Hasse diagram, and linear extensions.

This paper is an extended version of our previous works (Ordanel *et al.* 2017, Ordanel & Adorna 2017). We present the results into three sections, the second of which contains the base results from previous works. These base results are used to build the main result of this paper, which is on the Hammock(2,2)-Poset Cover Problem.

Definitions

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 $u <_p w <_p v\}$.

For example, in the poset in Figure 2a, $(2,4) \in \prec_p$ which can also be written as $2 \prec_p 4$. Moreover, $(1,4) \notin \prec_p$.

Hasse Diagram of Poset. A poset $P = (V, <_p)$ corresponds to Hasse diagram $H(P)$, which is a directed acyclic graph $G = (V, \prec_p)$.

Figure 2b shows the corresponding Hasse diagram of poset P in Figure 2a.

Hammock Poset (Fernandez et al. 2013). A Hammock Poset $P = (V, <_p)$ is a poset where the set V can be partitioned into disjoint subsets $V_1, V_2, \ldots, V_k$ such that for $u \in V_i$ and $v \in V_j$, we have $u <_p v$ if and only if $i < j$, $|V_1| = |V_k| = 1$ and either $|V_i| = 1$ or $|V_{i+1}| = 1$ for $2 \leq i \leq k - 1$.

A non-singleton V_i in a hammock poset is called a hammock, while a singleton V_i is called a link. The sequence of the sizes of the hammocks are used to name the hammock poset. The Hammock$(a_1, a_2, \ldots, a_n)$- Poset is the hammock poset where a_j is the size of the jth hammock.

For example, a Hammock(2,3)-Poset is a hammock poset that has two hammocks – the first hammock has size of 2 and the second hammock has size of 3. The Hasse diagram of a Hammock(2,3)-Poset is shown on Figure 3.

***C*-Poset Cover Problem.** Let C be a property of a poset (e.g., C characterizes the Hasse diagram of a poset). A poset on $\boldsymbol{V}$ that satisfies C is called a ***C*-Poset**. The ***C*-Poset Cover Problem** is the minimization problem where:

Instance: A set $\Upsilon = \{L_1, L_2, \ldots, L_m\}$ of linear orders over the set $V = \{1,2,3, \ldots, n\}$ and an integer $K \leq m$.

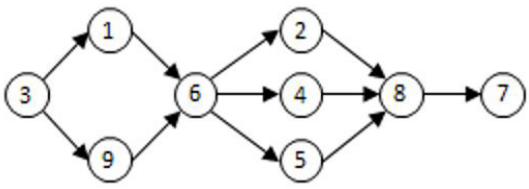

Figure 3. Hasse diagram of a hammock poset.

Solution: A poset cover $P^* = \{P_1, P_2, \ldots, P_k\}$ of Υ such that $k \leq K$ and P_i is a $\boldsymbol{C}$-poset for every $P_i \in P^*$.

For example, the Hammock(2,2)-Poset Cover Problem is the problem where the solution is a set of Hammock(2,2)-Posets that cover the input set of linear orders. Figure 4 shows an instance and a solution to a Hammock(2,2)-Poset Cover Problem.

Transposition Graph. A transposition is an exchange of two elements of an ordered list with all others staying the same. The transposition graph $G = (V, E)$ is a graph where V is a set of linear orders over a same set and $E = \{\{L_1, L_2\} | L_1$ and L_2 differ by a single transposition$\}$.

In this paper, we denote the transposition graph on the linear extension of poset $\boldsymbol{P}$ as $G(P)$, which is transposition graph with $\mathcal{L}(P)$ as the vertex set. We also refer to the two

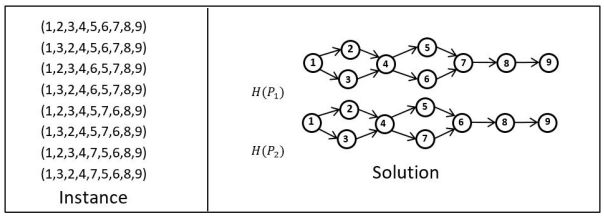

Figure 4. Example of Hammock(2,2)-Poset Cover Problem.

elements being swapped in a transposition as swapPair. For example, let $L = (x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, x_{i+2}, \ldots, x_n$ and $L' = (x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, x_{i+2}, \ldots, x_n$ be linear orders. Then swapPair $(L; L') = \{x_i, x_{i+1}\}$.

Siblings. Given $P = (V, <_p)$ and $a, b, c \in V$, a and b are siblings if for all $c \in V - \{a, b\}$, $c <_p a$ if and only if $c <_p b$, and $a <_p c$ if and only if $b <_p c$.

Comparable. Given a poset $P = (V, <_p)$ and elements u, $v \in V$, u and v are comparable if $u <_p v$ or $v <_p u$. On the other hand, u and v are incomparable, if $u \not<_p v$ and $v \not<_p u$.

The elements of a hammock are siblings and incomparable.

Minimum Edge Cover Problem. The Minimum Edge Cover Problem is an optimization problem that is in P (Garey & Johnson 1979):

Instance: A nonempty undirected graph $G = (V, E)$.

Solution: A minimum subset $E' \subseteq E$ where every vertex in V is incident on at least one edge in E'

Similar Posets. Given two posets $P = (V, <_p)$ and $P' = (V', <_{p'})$, we say that P and P' are similar if and only if $V = V'$ and $<_P = <_{P'}$.

RESULTS

Base Result

The first lemma tells us a condition when can two posets be combined into one new poset that covers the same set of linear orders as the two posets. For example, consider the posets P_1, P_2, and P_3 in Figure 5. Posets P_1 and P_2 satisfy the condition in Lemma 2.1 *i.e.*, there exists at most one pair {6,7} where $<_{P_1} \setminus \{(6,7)\} = <_{P_2} \setminus \{(6,7)\}$. From Lemma 2.1, we can combine P_1 and P_2 into new poset P_3 where $<_{P_3} = <_{P_1} \setminus \{(6,7)\} = <_{P_2} \setminus \{(6,7)\}$ and $\mathcal{L}(P_3) = \mathcal{L}(P_1) \cup (P_2)$ = {(1,2,3,4,5,6,7,8,9), (1,2,3,4,5,6,7,8,9), (1,2,3,4,5,6,7,8,9), (1,2,3,4,5,6,7,8,9) }

Lemma 2.1. 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 $(a, b) \in <_{P_1}$, $(b, a) \in <_{P_2}$ 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 $\mathcal{L}(P_3) = \mathcal{L}(P_1) \cup \mathcal{L}(P_2)$.

Proof. Suppose there exists one such pair $\{a, b\}$ where $(a, b) \in <_{P_1}$, $(b, a) \in <_{P_2}$ and $<_{P_1} \setminus (\{a, b\}) = <_{P_2} \setminus \{(b, a)\}$. We first show that $P_3 = (V, <_{P_3})$ where $<_{P_3} = <_{P_1} \setminus (\{a, b\})$ is a poset. Removing a pair from a poset does not affect the irreflexive and antisymmetric properties of the remaining pairs. This means $<_{P_3}$ is irreflexive and antisymmetric. The only way that $<_{P_3}$ is not transitive is if there exist at least one element x such that (a, x), $(x, b) \in V <_{P_3}$ or (b, x), $(x, a) \in <_{P_3}$. Now, suppose there exists such element $x \in V$ such that (a, x), $(x, b) \in <_{P_3}$. This implies that (a, x), (x, b) are also in $<_{P_1}$ and $<_{P_2}$. By transitive property of posets, $(a, b) \in <_{P_2}$. This contradicts the antisymmetric property of $<_{P_2}$ since (a, b) is also in $<_P$. In a similar way, we can also show that there exists no x such that (b, x), $(x, a) \in <_{P_3}$.

Next, we show that $\mathcal{L}(P_1) \cup \mathcal{L}(P_2) \subseteq \mathcal{L}(P_3)$ and $\mathcal{L}(P_3) \subseteq \mathcal{L}(P_1) \cup \mathcal{L}(P_2)$.

To show the first part, let the linear order $L_1 \in \mathcal{L}(P_1) \cup \mathcal{L}(P_2)$. Then, it is either $L_1 \in \mathcal{L}(P_1)$ or $L_1 \in \mathcal{L}(P_2)$. Suppose $L_1 \in \mathcal{L}(P_1)$. This means that $<_{P_1} \subseteq <_{L_1}$. Since $<_{P_3} \subseteq <_{P_1}$, then $<_{P_3} \subseteq <_{L_1}$. This implies that $L_1 \in \mathcal{L}(P_3)$. In a similar way, we can show that if $L_1 \in \mathcal{L}(P_2)$, then $L_1 \in \mathcal{L}(P_3)$. Thus, $\mathcal{L}(P_1) \cup \mathcal{L}(P_2) \subseteq \mathcal{L}(P_3)$.

To show the second part, let the linear order $L_2 \in \mathcal{L}(P_3)$. By definition, $<_{P_3} \subseteq <_{L_2}$. Since a is incomparable to b in P_3 and every pair of distinct elements in L_2 are comparable, then it is either $(a, b) \in <_{L_2}$ or $(b, a) \in <_{L_2}$. If $(a, b) \in <_{L_2}$, then $<_{P_3} \cup \{(a, b)\} \subseteq_{L_2}$. This implies $<_{P_1} \subseteq <_{L_2}$. Thus, $L_2 \in \mathcal{L}(P_1)$. On the other hand, if $(b, a) \in <_{L_2}$, then $<_{P_3} \cup \{(a, b)\} \subseteq_{L_2}$. This means $<_{P_2} \subseteq <_{L_2}$. Thus, $L_2 \in \mathcal{L}(P_2)$. In either case, $L_2 \in \mathcal{L}(P_1) \cup \mathcal{L}(P_2)$. Hence, $\mathcal{L}(P_3) \subseteq \mathcal{L}(P_1) \cup \mathcal{L}(P_2)$. ■

The following lemma is somewhat the reverse of the previous one. If a poset has a pair of elements that are siblings, then we can decompose it into two new posets such that the union of the linear extensions of the two posets is the same as the set of linear extensions of the original poset. In the same example in Figure 5, {6,7} are siblings in poset P_3, hence we can decompose poset P_3 into posets P_1 and P_2 such that they cover the same set of linear orders as P_3.

Lemma 2.2. Given a poset $P = (V, <_P)$ and $x_i, x_j \in V$, if x_i and x_j are siblings in P, then there exist posets $P_1 = (V, <_{P_1})$ and $P_2 = (V, <_{P_2})$ where $<_{P_1} = <_P \cup \{(x_i, x_j)\}$ and $<_{P_2} = <_P \cup \{(x_j, x_i)\}$ such that $\mathcal{L}(P_1) = \{L \in \mathcal{L}(P) | x_i <_L x_j\}$ and $\mathcal{L}(P_2) = \{L \in \mathcal{L}(P) | x_j <_L x_i\}$. Moreover, $\mathcal{L}(P_1) \cap \mathcal{L}(P_2) = \emptyset$ and $\mathcal{L}(P_1) \cup \mathcal{L}(P_2) = \mathcal{L}(P)$.

Proof. We first show that P_1 and P_2 are posets. Since $x_i \neq x_j$, then $<_{P_1}$ and $<_{P_2}$ are irreflexive. x_i and x_j are siblings; hence, they are incomparable. This means there is no (x_i, x_j) nor (x_j, x_i) in $<_{P_1}$ and $<_{P_2}$. Hence, adding (x_i, x_j) or (x_j, x_i) to $<_P$ makes $<_{P_1}$ and $<_{P_2}$ still antisymmetric. Lastly, to show that $<_{P_1}$ is transitive, we just need to show that the addition of pair (x_j, x_i) to transitive relation $<_P$ makes $<_{P_1}$ still transitive. Let $a, b \in V$ where $a <_{P_1} x_i$ and $x_j <_{P_1} b$. Since x_i and x_j are siblings in P, then $a <_P x_j$ and $x_i <_P b$. This implies $a <_{P_1} x_j$ and $x_i <_{P_1} b$. Hence, $<_{P_1}$ is transitive. We can also show that $<_{P_2}$ is transitive in similar way.

Now, let $\boldsymbol{A} = \{L \in \mathcal{L}(P) | x_i <_L x_j\}$ and $\boldsymbol{B} = \{L \in \mathcal{L}(P) | x_i <_L x_j\}$. Next, we show that $\mathcal{L}(P_1) = A$ and $\mathcal{L}(P_2) = B$. Let $L_1 \in \mathcal{L}(P_1)$. Then $<_P \subset <_{P_1}$. Since $<_P \subset <_{P_1}$, then $<_P \subset <_{L_1}$. Hence $L_1 \in \mathcal{L}(P)$. Also, since $(x_i, x_j) \in <_{P_1}$ then $x_i <_{L_1} x_j$. Hence, $L_1 \in A$. On the other hand, let $L_2 \in A$. This means $<_P \subset <_{L_2}$. Also, since $x_i <_{L_2} x_j$, then $x_i, x_j \in <_{L_2}$. This implies that $<_P \cup \{(x_i, x_j)\} \subset <_{L_2}$. This means $<_{P_1} \subset <_{L_2}$. Hence, $L_2 \in \mathcal{L}(P_1)$. Thus, $A = \{L \in \mathcal{L}(P) | x_i <_L x_j\}$. In the same way, we can also show that $B = \{L \in \mathcal{L}(P) | x_j <_L x_i\}$.

Next, suppose there exists $L \in A \cap B$. This means that $x_i <_L x_j$ and $x_j <_L x_i$. This contradicts that antisymmetric property of $<_L$. Hence, $A \cap B = \emptyset$.

Lastly, since every pair in $L \in P$ is related, then it is either $x_i <_L x_j$ or $x_j <_L x_i$. Hence, it is either $L \in A$ or $L \in B$. Hence, $A \cup B = \mathcal{L}(P)$. ■

A Hammock($\underbrace{2,2,\ldots,2}_{k}$)-Poset has k pairs of siblings. From Lemma 2.2, we can then decompose a Hammock($\underbrace{2,2,\ldots,2}_{k}$) -Poset and the resulting posets are Hammock($\underbrace{2,2,\ldots,2}_{k-1}$)-Posets that cover the same linear orders as the Hammock($\underbrace{2,2,\ldots,2}_{k}$) -Poset. We can iteratively reapply Lemma 2.2 until there are no more hammocks or, in other words, until the resulting posets are linear orders. However, the problem we are trying to solve is the reverse – the given set is a set of linear orders and the goal is to get a set of Hammock

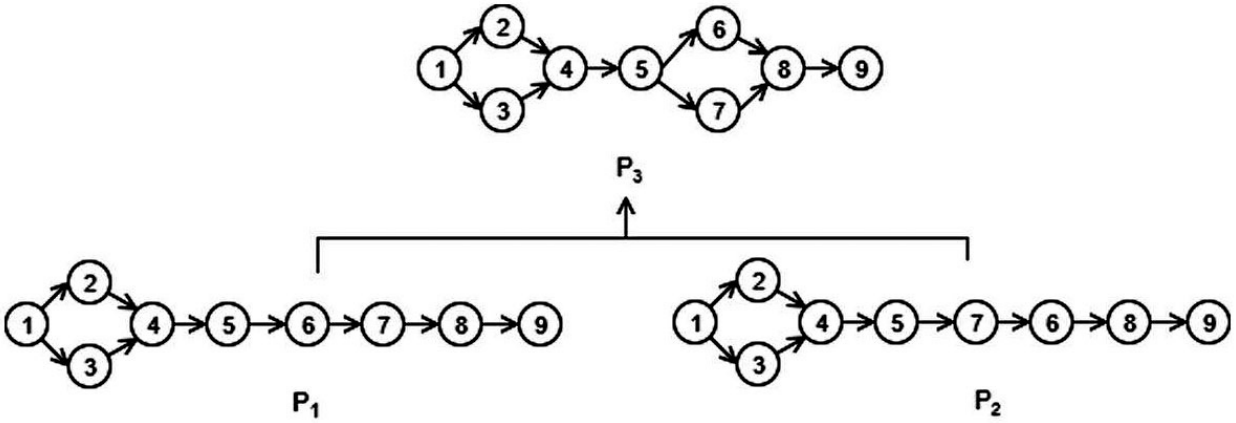

Figure 5. Example of Lemma 2.1.

($\underbrace{2,2,\dots,2}_{k}$)-Posets that exactly cover them. But with Lemma 2.2, we can do the reverse process. We can start with the linear orders – combine them using Lemma 2.1 to make Hammock(2)-Posets until we get to the Hammock($\underbrace{2,2,\dots,2}_{k}$) -Posets. This is basically what the following Algorithm 1 does to get the set $P_r^* = \{P|\ P$ is Hammock ($\underbrace{2,2,\dots,2}_{k}$)-Posets and $\mathcal{L}(P) \subseteq \Upsilon\}$ where $\bigcup_{P\in P_r^*} \mathcal{L}(P) = \Upsilon$.

Theorem 2.3. The Algorithm 1 returns $P_r^* = \{P|P$ is Hammock ($\underbrace{2,2,\dots,2}_{k}$)-Posets and $\mathcal{L}(P) \subseteq \Upsilon\}$ such that $\bigcup_{P\in P_r^*} \mathcal{L}(P) = \Upsilon$ in $O(kn^2m^2)$ time.

Proof. We first show by induction that the output in each rth iteration of the outermost for-loop in Algorithm 1 is $P_r^* = \{P|\ P$ is Hammock($\underbrace{2,2,\dots,2}_{r}$) such that $\mathcal{L}(P) \subseteq \Upsilon\}$.

For $r = 1$, let P_1^* be the output of the first iteration. Let $P = (V, <_P)$ be a Hammock(2)- Poset such that $\mathcal{L}(P) \subseteq \Upsilon$. Let x_1 and x_2 be the elements of the hammock of P. Since P has only one hammock, x_1 and x_2 are the only incomparable pairs in $<_P$. Moreover, x_1 and x_2 are also siblings. Hence, from Lemma 2.2, P is a combination of posets $P_1 = (V, <_{P_1})$ and $P_2 = (V, <_{P_2})$ where $<_{P_1} = <_P \cup \{(x_1, x_2)\}$ and $<_{P_2} = <_P \cup \{(x_2, x_1)\}$. This implies that every pair of elements in V are related in $<_{P_1}$ and $<_{P_2}$. Hence, P_1 and P_2 are linear orders. The first iteration of Algorithm 1 checks on every pair of linear orders, say L_1, $L_2 \in \Upsilon$ that can be combined according to the condition in Lemma 2.1. Since P_1 and P_2 are linear orders in Υ and $<_{P_1} - \{(x_1, x_2)\} = <_{P_2} - \{(x_2, x_1)\}$, then they must be identified by Algorithm 1 as linear orders that can be combined. This is performed in the CombinePoset subroutine of Algorithm 1. Moreover, the CombinePoset subroutine also ensures in lines 19–30 that $\{x_1, x_2\}$ form a hammock by verifying if there exists only one $c \in V$ where (c, x_1), $(c, x_2) \in \prec_P$ and only one $d \in V$ where (x_1, d), $(x_2, d) \in \prec_P$. Hence, $P \in P_1^*$.

Now, suppose the claim is true for $r = t$ that is the output is P_r^* in iteration t contains all possible Hammock($\underbrace{2,2,\dots,2}_{t}$)-Posets whose set of linear extensions are in Υ. Let $P_3^t = (V, <_{P_3})$ be a Hammock($\underbrace{2,2,\dots,2}_{t+1}$)-Poset such that $\mathcal{L}(P_3) \subseteq \Upsilon$. Let the ith hammock of P_3 be $\{x_{i_1}, x_{i_2}\}$. Since x_{i_1} and x_{i_2} are siblings, then P_3 is a combination of posets $P_{i_1} = (V, <_{P_{i_1}})$ and $P_{i_2} = (V, <_{P_{i_2}})$ where $<_{P_{i_1}} = <_{P_3} \cup \{(x_{i_1}, x_{i_2})\}$, $<_{P_{i_2}} = <_{P_3} \cup \{(x_{i_2}, x_{i_1})\}$. Since there are $t+1$ pairs of siblings in P_3, then there are also $t+1$ pairs of P_{i_1} and P_{i_2}, $i = 1, \dots, t+1$. Note that P_{i_1} and P_{i_2} are hammock posets with t hammocks of size 2. Since $\mathcal{L}(P_{i_1}) \subseteq \Upsilon$ and $\mathcal{L}(P_{i_2}) \subseteq \Upsilon$ then $P_{i_1}, P_{i_2} \in P_t^*$. The iteration $r = t+1$ of Algorithm 1 checks and combines all possible pairs of posets in P_t^* using Lemma 2.1. Note that $<_{P_{i_1}}$ and $<_{P_{i_2}}$ satisfy the condition of Lemma 2.1. Hence, combining them by Lemma 2.2, we get P_3. Hence, $P_3 \in P_{t+1}^*$. In other words, the claim is true for iteration $r = t+1$.

Since r goes from 1 to k, then at the end of the outermost for-loop (line 14) the value of $P_r^* = \{P|\ P$ is Hammock ($\underbrace{2,2,\dots,2}_{k}$) such that $\mathcal{L}(P) \subseteq \Upsilon\}$. After which, the algorithm can easily verify in line 16 if all the linear orders in Υ are covered. Hence, the algorithm returns $P_r^* = \{P|\ P$ is Hammock($\underbrace{2,2,\dots,2}_{k}$) and $\mathcal{L}(P) \subseteq \Upsilon\}$ such that $\bigcup_{P\in P_r^*} \mathcal{L}(P) = \Upsilon$.

Now, we determine the running time of the algorithm. It is easy to see that the call to CombinePoset in line 6 dominates the time complexity of the GetAllHammock $2k$ Posets. The CombinePoset runs in $O(n^2)$ since $|<_{P_1}| = |<_{P_2}| \in O(n^2)$. The $<_{P_1}$ and $<_{P_2}$ can be stored in a binary matrix, hence determining if a pair is in $<_{P_1}$ or $<_{P_2}$ is just $O(1)$. The CombinePoset is called in three loops – the outer while-loop and the two inner for-loops.

Each inner for-loop runs in $O(m)$ since $|P_{r^*-1}| \in O(m)$. The outer while-loop iterates k times. Hence, the time complexity of the entire Algorithm 1 is $O(kn^2m^2)$.■

Figure 8 shows an example of the algorithm when $k = 2$. The first column contains the input set of linear orders

and the second column contains the Hammock(2)-Posets derived by combining the linear orders. Then, the last column contains the Hammock(2,2)-Posets derived by combining the Hammock(2)-Posets. You may notice that the last column contains posets that are similar in terms of relation and set of linear extensions. However, the algorithm stores attributes P_{e_x} and P_{e_y} of poset P, which are the posets combined to get P – thus, as data, the redundant posets differ. The attributes P_{e_x} and P_{e_y} of the posets are stored for use of Algorithm 2. However, if the use of Algorithm 1 is just to get a cover of unique Hammock$^{(\underbrace{2,2,\ldots,2}_{k})}$-Posets, we can just remove easily the redundant posets in the final ouput.

Optimal Deterministic Algorithm for Hammock(2,2)-Poset Cover Problem

The output of Algorithm 1 when $k = 2$ is $P_r^* = \{P |\ P$ is Hammock (2,2)-Posets and $\mathcal{L}(P) \subseteq \Upsilon\}$ such that $\bigcup_{P \in P_r^*} \mathcal{L}(P) = \Upsilon$. Hence, for Hammock(2,2)-Poset Cover Problem, what remains is to get the minimum subset of P_r^* that cover exactly Υ. This can be simply done using a deterministic algorithm for k-Set Cover Problem, a variant of Set Cover Problem where the subsets in the input have size ≤ k. The following outlines the steps in getting the minimum Hammock(2,2)-Poset using Set Cover Problem.

$P_a^* = \text{GetAllHammock2k-Posets}(\Upsilon, 2)$
$\tau \leftarrow \{\mathcal{L}(P) |\ P \in P_a^*\}$
$\omega \leftarrow \text{MinSetCover}(\Upsilon, \tau)$
$P^* = \{P \mid \mathcal{L}(P) \in \omega\}$

$|t| = 4$ for all subsets $t \in \tau$ in the input to a Minimum Set Cover since every Hammock(2,2)-Poset has 4 linear extensions. However, the k-Set Cover Problem is NP-hard for $k > 2$ and the solution above could have a running time of $O(2^m)$ in the worst case.

In the following set of results, instead of working on the linear extensions of poset as subsets, we work on the transposition graph of the linear extensions of the posets.

Lemma 2.4. Let $G(P)$ be the transposition graph of Hammock(2,2) Poset P. Then,

(1) $G(P)$ is a C_4 (cycle graph of length 4)

(2) For every edge $\{L, L'\}$ of $G(P)$, there exists a Hammock(2)-Poset P' that generates exactly the linear orders L and L', *i.e.*, $\mathcal{L}(P') = \{L, L'\}$

(3) Let $\{L_a, L_b\}$ and $\{L_c, L_d\}$ be alternating edges in $G(P)$ and P_r and P_s be the Hammock(2)-Posets that generate the alternating edges, respectively. Then P_r and P_s can

Algorithm 1 Get All Hammock2kPosets

INPUT: $\Upsilon = \{L_1, L_2, \ldots, L_m\}$ of linear orders on $V = \{1,2,\ldots,n\}$ and k -the number of hammocks of size 2.
OUTPUT: A set $P_r^* = \{P |\ P$ is Hammock $(\underbrace{2,2,\ldots,2}_{k})$-Posets and $\mathcal{L}(P) \subseteq \Upsilon\}$ such that $\bigcup_{P \in P_r^*} \mathcal{L}(P) = \Upsilon$

```
P_0* ← {P_0_1, P_0_2, ..., P_0_m} where P_0_1 = L_1, P_0_2 = L_2, ..., P_0_m = L_m
for r ← 1 to k
    s ← 0
    for i ← 1 to |P*_{r-1}|
        for j ← i + 1 to |P*_{r-1}|
            P_temp = CombinePoset(P_{r-1_i}, P_{r-1_j})
            if P_temp ≠ null
                s ← s + 1
                P_{r_s} ← P_temp
                P_r* ← P_r* ∪ {P_{r_s}}
            end if
        end for
    end for
end for
if ⋃_{P∈P_r*} L(P) = Υ
        return P_r*
else
        return null
end if
```

Figure 6. Algorithm 1 pseudocode.

INPUT: Posets P_1 and P_2
OUTPUT: Poset P
1: **function** CombinePoset
2: /* Check if the condition in Lemma is satisfied */
3: **if** $|<_{P_1}| \neq |<_{P_2}|$ **then return** *null*
4: **end if**
5: $<_{P_3} = <_{P_1} - <_{P_2}$
6: $<_{P_4} = <_{P_2} - <_{P_1}$
7: **if** $|<_{P_3}| \neq 1$ or $|<_{P_4}| \neq 1$ **then return** *null*
8: **end if**
9: $(a, b) \leftarrow <_{P_3}$
10: $(c, d) \leftarrow <_{P_4}$
11: **if** $a \neq d$ *or* $b \neq c$ then **return** *null*
12: **end if**
13: /* Combine the two posets according to Lemmma */
14: $<_P \leftarrow <_{P_1} - <_{P_3}$
15: /* Store the Hammock Posets that were combined to become P as P_{e_x} and P_{e_y}. This will be used in succeeding algorithms */
16: $P_{e_x} \leftarrow P_1$
17: $P_{e_y} \leftarrow P_2$
18: $\mathcal{L}(P) \leftarrow \mathcal{L}(P_1) \cup \mathcal{L}(P_2)$
19: /* Check if the combined poset P is a Hammock Poset */
20: $noPrecCover \leftarrow 0$
21: $noSucCover \leftarrow 0$
22: **for all** $i \in V - \{a, b\}$
23: **if** $(i, a) \prec_P$ and $(i, b) \prec_P$
24: $noPrecCover \leftarrow noPrecCover + 1$
25: **else if** $(a, i) \prec_P$ and $(b, i) \prec_P$
26: $noSucCover \leftarrow noSucCover + 1$
27: **end if**
28: **end for**
29: **if** $noPrecCover \neq 1$ or $noSucCover \neq 1$ **then return** *null*
30: **end if**
31: **return** P
32: **end function**

Figure 7. CombinePoset subroutine.

be combined to get P *i.e.*, $<_P = <_{P_r} \setminus \{(v_x, v_y)\} = <_{P_s} \setminus \{(v_y, v_x)\}$ where $\{v_x, v_y\}$ is a hammock in P and $\mathcal{L}(P) = \{L_a, L_b, L_c, L_d\}$.

Proof. Let $P = (V, <_P)$ be a Hammock(2,2)-Poset where $V = \bigcup_{i=1}^{q} V_i$, $V_i \cap V_j = \emptyset$ for every $i, j = 1$ and $i \neq j$. Let V_r and V_s where $r < s$ be the two hammocks of size 2. Without loss of generality, let $V_r = \{v_{r_1}, v_{r_2}\}$, $V_s = \{v_{s_1}, v_{s_2}\}$, and $V_t = \{v_t\}$ for $t = 1, 2, \dots, r-1, r+1, \dots, s-1, s+1\ 1, \dots, q$.

$$L_1 = (v_1, v_2, v_3, \dots, v_{r-1}, v_{r_1}, v_{r_2}, v_{r+1}, \dots, v_{s-1}, v_{s_1}, v_{s_2}, v_{s+1}, \dots, v_q)$$

$$L_2 = (v_1, v_2, v_3, \dots, v_{r-1}, v_{r_2}, v_{r_1}, v_{r+1}, \dots, v_{s-1}, v_{s_1}, v_{s_2}, v_{s+1}, \dots, v_q)$$

$$L_3 = (v_1, v_2, v_3, \dots, v_{r-1}, v_{r_2}, v_{r_1}, v_{r+1}, \dots, v_{s-1}, v_{s_2}, v_{s_1}, v_{s+1}, \dots, v_q)$$

$$L_4 = (v_1, v_2, v_3, \dots, v_{r-1}, v_{r_1}, v_{r_2}, v_{r+1}, \dots, v_{s-1}, v_{s_2}, v_{s_1}, v_{s+1}, \dots, v_q)$$

If we build the transposition graph of P, we have $G(P) = \mathcal{L}(P), E)$ where $E = \{\{L_1, L_2\}, \{L_2, L_3\}, \{L_3, L_4\}, \{L_1, L_4\}\}$. Note that $G(P)$ is a cycle graph of length 4.

By definition of Hammock posets, for $v_i \in V_i$, $v_j \in V_j$, $v_i < v_j$ whenever $i < j$. Since all the V_i's are singleton except for the hammocks, then in the linear extensions of hammock poset, only those hammock vertices can be transposed. Since the hammocks in Hammock(2,2) are of size 2, then the transposition is always adjacent. That means every edge $\{L, L'\}$ in $G(P)$ differ by a single adjacent transposition of elements in a hammock. We can then arbitrarily define an edge $\{L, L'\} \in G(P)$ where $L = (v_1, v_2, \dots, v_x, v_{x+1}, \dots, v_q)$ and $L' = (v_1, v_2, \dots, v_{x+1}, v_x, \dots, v_q)$ where $\{v_x, v_{x+1}\}$ is a hammock in P. Note that $<_L \setminus \{(v_x, v_{x+1})\} = <_{L'} \setminus \{(v_{x+1}, x)\}$. Then by Lemma 2.1, there

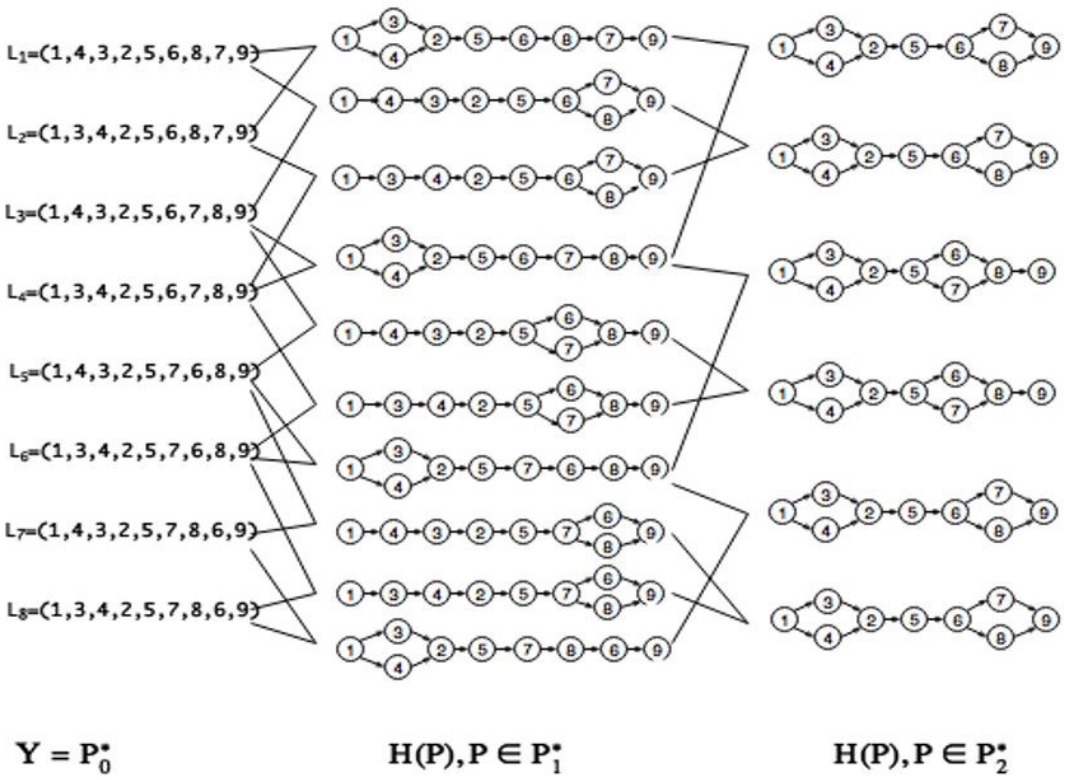

Figure 8. Algorithm 1 example.

exists a poset $P'=(V,<_{P'})$ where $<_{P'}=<_L\setminus\{(v_x,v_{x+1})\}=<_{L'}\setminus\{(v_{x+1},v_x)\}$ and $\mathcal{L}(P')=\{L,L'\}$. Specifically, (P' is a Hammock(2)-Poset where V can be partitioned into $V=\{v_1\}\cup\{v_2\}\cup\ldots\cup\{v_{x-1}\}\cup\{v_x,v_{x+1}\}\cup\{v_{x+2}\}\cup\ldots\cup\{v_q\}$.

Let $\{L_a, L_b\}$ and $\{L_c, L_d\}$ be alternating edges in $G(P)$ and P_r and P_s be the Hammock(2)-Posets that generate the alternating edges, respectively. Being alternating edges, $\{L_a, L_b\}\cap\{L_c, L_d\}=\emptyset$. This implies that there exists a pair, say $\{v_x, v_y\}$, where $v_x <_{L_a} v_y$ and $v_x <_{L_b} v_y$ while $v_y <_{L_a} v_x$ and $v_y <_{L_b} v_x$. Since $L_a, L_b, L_c, L_d \in \mathcal{L}(P)$, then$<_p \subseteq L_a$, $<_p \subseteq L_b$, $<_p \subseteq L_c$ and $<_p \subseteq L_d$. This means that $\{v_x, v_y\}$ must be incomparable in P. Hence, $\{v_x, v_y\}$ is one of the hammocks of P. But $\{v_x, v_y\}$ is not a hammock in P_r and P_s since $(v_x, v_y)\in<_{P_r}$ and $(v_y, v_x)\in<_{P_s}$. Let $\{v_i, v_j\}$ be the other hammock of P. Then $\{v_i, v_j\}$ must be the hammock of P_r and P_s. Since P_r and P_s are Hammock(2)-Posets, all the other pair in $<_{P_s}$ and $<_{P_r}$ must be the same except for (v_x, v_y) or (v_y, v_x), otherwise P would have more incomparable pairs other than elements in the two hammocks. Hence, $<_{P_r}=<_{P_r}\setminus\{(v_x, v_y)\}=<_{P_s}\setminus\{(v_y, v_x)\}$. By Lemma 2.1, we can combine P_r and P_s to get a poset P. ■

Lemma 2.5. Given two Hammock(2,2)-Posets $P_1=(V, <_{P_1})$ and $P_2=(V, <_{P_2})$, $P_1\neq P_2$, if $\mathcal{L}(P_1)\cap\mathcal{L}(P_2)\neq\emptyset$, then $\mathcal{L}(P_1)\cap\mathcal{L}(P_2)=\{L_1\}$ xor $\mathcal{L}(P_1)\cap\mathcal{L}(P_2)=\{L_1, L_2\}$ where $\{L_1, L_2\}$ is an edge in $G(P_1)$ and $G(P_2)$.

Proof. First, we show that $|\mathcal{L}(P_1)\cap\mathcal{L}(P_2)|$ cannot be greater than 2. We know that $|\mathcal{L}(P_1)|=|\mathcal{L}(P_2)|=4$. Hence, $|\mathcal{L}(P_1)\cap\mathcal{L}(P_2)|$ can be 0,1,2,3 or 4. However, since $P_1\neq P_2$, then $|\mathcal{L}(P_1)\cap\mathcal{L}(P_2)|$ cannot be 4. Now, suppose $|\mathcal{L}(P_1)\cap\mathcal{L}(P_2)|=3$. Let $\mathcal{L}(P_1)=\{L_1, L_2, L_3, L_4\}$ and $\mathcal{L}(P_2)=\{L_1, L_2, L_3, L_4'\}$, $L_4\neq L_4'$. From Lemma 2.4, $G(P_1)$and $G(P_2)$ are cycle graphs. Since the cycle graphs are of length 4 only, then L_1, L_2, L_3 must form a path in $G(P_1)$ and $G(P_2)$. Without loss of generality, let the path be (L_1, L_2, L_3) in $G(P_1)$ and $G(P_2)$. Since $G(P_1)$ is a cycle, then there must be edges $\{L_4,L_1\}, \{L_4,L_3\}\in G(P_1)$. Similarly, $\{L_4,L_1\}, \{L_4,L_3\}\in G(P_2)$. Let swapPair($\{L_4,L_1\}$)=$\{a, b\}$ and swapPair($\{L_4, L_3\}$)=$\{c, d\}$. The only pair that can be swapped in the linear extensions of a Hammock(2,2)-Poset are the elements of hammocks. This implies that $\{a, b\}$ and $\{c, d\}$ are the 2 hammocks of P_1. Similarly, since $\{L_2, L_1\}$ and $\{L_2, L_3\}$ are edges in $G(P_1)$, then swapPair($\{L_2, L_1\}$) =$\{a, b\}$ and swapPair($\{L_2, L_3\}$) =$\{c, d\}$ or *vice versa*. Note that $\{L_2, L_1\}$ and $\{L_2, L_3\}$ are also edges in $G(P_2)$. Hence, the two hammocks of P_2 are also $\{a, b\}$ and $\{c, d\}$. This implies that swapPair($\{L_4',L_1\}$) = $\{a, b\}$ and swapPair($\{L_4,L_3\}$) =$\{c, d\}$, or *vice versa*. Either case means $L_4=L_4'$. Hence, $P_1=P_2$, which is a contradiction to our assumption.

Moreover, Figure 9 shows Hasse diagrams of hammocks where the size of the intersection of Hammock(2,2)-Posets is 0, 1, or 2. $|\mathcal{L}(P_1) \cap \mathcal{L}(P_2)| = |\mathcal{L}(P_1) \cap \mathcal{L}(P_3) = 2$. $|\mathcal{L}(P_2) \cap \mathcal{L}(P_3)| = 1$. $|\mathcal{L}(P_4) \cap \mathcal{L}(P_1)| = |\mathcal{L}(P_4) \cap \mathcal{L}(P_2)| = |\mathcal{L}(P_4) \cap | \mathcal{L}(P_3)| = 0$.

It is known that a transposition graph is bipartite (Chase 1979). Now, if $|\mathcal{L}(P_1) \cap \mathcal{L}(P_3)| = 2$, say L_1 and L_2, then $\{L_1, L_2\}$ must be an edge in $G(P_1)$ and $G(P_3)$; otherwise, the bipartite property of transposition graph is violated.■

From previous lemmas, the transposition graph of the union of more than one Hammock(2,2)-Posets consist of

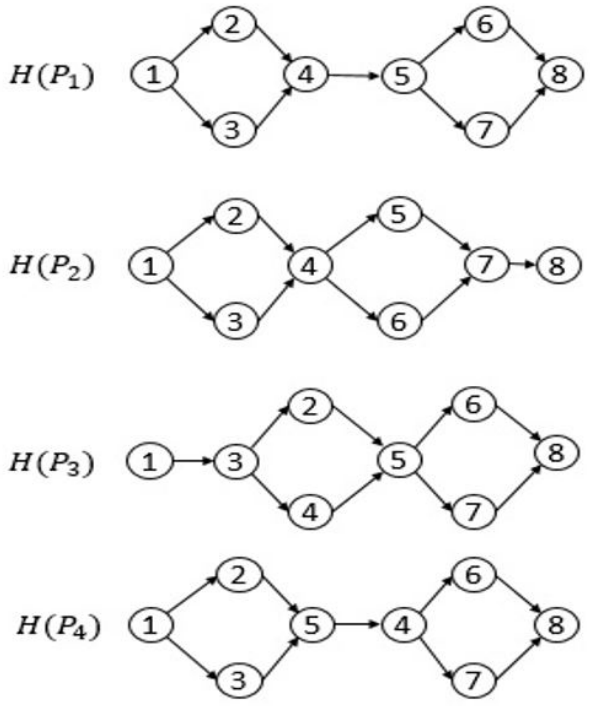

Figure 9. Hasse diagram of Hammock(2,2) Posets P_1, P_2, P_3, P_4.

cycles of length 4. This gives us another perspective in getting the minimum number of Hammock(2,2)-Posets cover. We can treat the problem as a special case of the Cycle Cover Problem where we only want to get a minimum number of C_4 that cover the vertices. However, the Cycle Cover Problem is NP-complete when cycles are of length greater than 2 (Bläser & Manthey 2002). However, the transposition graph of Hammock(2,2)-Posets have other properties derived in previous lemmas aside from being a cycle of length 4. We used these properties in devising Algorithm 2 to get the minimum Hammock(2,2)-Poset cover.

We discuss Algorithm 2 using the example shown in Figure 10. The example uses the same input as that of Algorithm 1 in Figure 8. Algorithm 2 starts off by calling Algorithm 1 to get all possible Hammock(2,2)-Posets covers, which is stored as P_a^*. Hence, P_a^* is the set of posets in the third column in Figure 8. To determine the minimum subset of P_a^* that covers the input, we make use of the transposition graph $G(\Upsilon)$ (line 5). Note that at this point we are already sure that there exists a set of Hammock(2,2)-Posets that exactly covers Υ; otherwise, Algorithm 2 should have been terminated from line 3. This means that from Lemma 2.4, G is composed of C_4's where the cycles can either intersect in a node or in an edge. From the previous lemma, we can treat a C_4 as a Hammock(2,2)-Posets – say P – and its alternating edges as the Hammock(2)-Posets combined to get P, which are the P_{e_x} and the P_{e_y}. That is why the notation are subscripts e_x and e_y because they can be treated as alternating edges in the transposition graph of Hammock(2,2)-Posets. From now on, we can interchangeably refer to the alternating edges of transposition graph of P as the Hammock(2)-Posets P_{e_x} and P_{e_y}.

The next step is to remove redundant posets in P_a^*. We assume without loss of generality that the linear orders in input Υ are unique. With this, we expect that a poset in P_a^* can have only one other poset similar to it. For each pair of similar posets, we need to get only one. We don't just pick one from the pair of similar posets. We need to select the posets such that the set union of the P_{e_x} and P_{e_y} from these posets produces a smallest set. For instance, Hammock(2,2)-Posets P_1 and P_2 are a pair of similar posets in P_a^*. Let us say that P_3 and P_4 are another pair of similar posets in P_a^*. Suppose we select P_1 in the first pair and suppose P_3 has the same Hammock(2)-Poset component, say, $P_{1_{e_x}} = P_{3_{e_x}}$. Then, we select P_3 instead of P_4. In the perspective of $G(\Upsilon)$, we select a pair of alternating edges from each cycle such that the total number of selected edges is minimum. This is what the TrimSimilarPoset subroutine does. Since we remove redundant posets, the output of TrimSimilarPoset which is $P_b^* = \{P \in P_a^* |\ P$ is unique with respect to the relation i.e., there exists no other poset $P' \in P_b^*$ where $<_P = <_{P'}\}$ such that $\bigcup_{P \in P_b^*} \mathcal{L}(P) = \Upsilon$ and $G' = (\Upsilon, E_2)$ where E_2 contains the selected alternating edges.

Next, we want to get the minimum edges in E_2 that covers Υ. We do this by getting the minimum edge cover of $G' = (\Upsilon, E_2)$. There are many possible minimum edges covering and we can generate all of them using the algorithm of Uno (1997). What we want is the minimum edge cover that can be associated to a minimum number of posets in P_b^*. Let $E_{3,d}$ be that minimum edge cover of $G' = (\Upsilon, E_2)$. Then, a minimum set of Hammock(2,2)-Posets that covers Υ is the minimum set of Hammock(2,2)-Poset that contain $E_{3,d}$ as alternating edges. We obtain this by constructing a graph $G'' = (V', E_4)$

where the vertex set V' consist of edges in $E_{3,d}$ and there is an edge – say $\{e_i, e_j\} \in E_4$ – if and only if there is a poset P that has them as alternating edges *i.e.*, $\mathcal{L}(P_{e_x}) = e_i$ and $\mathcal{L}(P_{e_y}) = e_j$. If there is an edge $e_p \in E_{3,d}$ that is unpaired, we select one poset $P \in P_b^*$ that has e_p as an edge *i.e.*, $P_{e_x} = e_p$, and add the alternate edge of e_p – which is $\mathcal{L}(P_{e_y})$ – to V' and the edge $\{e_p, \mathcal{L}(P_{e_y})\}$ to E_4. In the next theorem, we show that the minimum edge cover of E_4, which is E_5, is the minimum number of Hammock(2,2)-Posets that cover Υ.

Theorem 2.6. Algorithm 2 returns the minimum Hammock(2,2)-Poset Cover Problem in $O(n^2m^3 + m^3M)$ time where M is the number of all maximum matchings of the subgraph of the transposition graph of input linear orders .

Proof. Without loss of generality, we assume that the linear orders in Υ are all unique. We show that the following claims hold in the respective step in the algorithm.

1) In Step 1,

   a. $P_a^* = \{P | P \text{ is Hammock}(2,2)\text{-Poset where } \mathcal{L}(P) \subseteq \Upsilon \text{ and } \bigcup_{P \in P_a^*} \mathcal{L}(P) = \Upsilon$.

      This follows directly from Theorem 2.3.

   b. Each $P \in P_a^*$, there exists exactly one $P' \in P_a^*$ where $<_p = <_{p'}$ but $P_{e_x} \neq P'_{e_x}$ and $P_{e_y} \neq P'_{e_y}$.

      There are two pairs of siblings in P and P'. From Lemma 2.2, we can decompose $<_P = <'_P$ in two ways. Using the first pair of sibling, we can have Hammock(2)-Posets P_{e_x} and P_{e_y}. Using the second pair of sibling, we can have the Hammock(2)-Posets P'_{e_x} and P'_{e_y}. From the proof of Theorem 2.3, P_{e_x}, P_{e_y}, P'_{e_x}, P'_{e_y} must be in P_1^*. Since Algorithm 1 combines every possible pair in P_1^*, then $<_P$ is obtained by combining P_{e_x} and P_{e_y} while $<'_P$ by combing P'_{e_x} and P'_{e_y}.

2) In Step 5,

   a. $G = (\Upsilon, E_1) = \bigcup_{i=1}^{\frac{|P_a^*|}{2}} C_i$ where C_i is a cycle of length 4. Let the set of cycles be $\mathbb{C} = \{C_1, C_2, \ldots, C_{\frac{|P_a^*|}{2}}\}$. Let $C_i, C_j \in \mathbb{C}$, then C_i and C_j can have no intersection, intersect in a

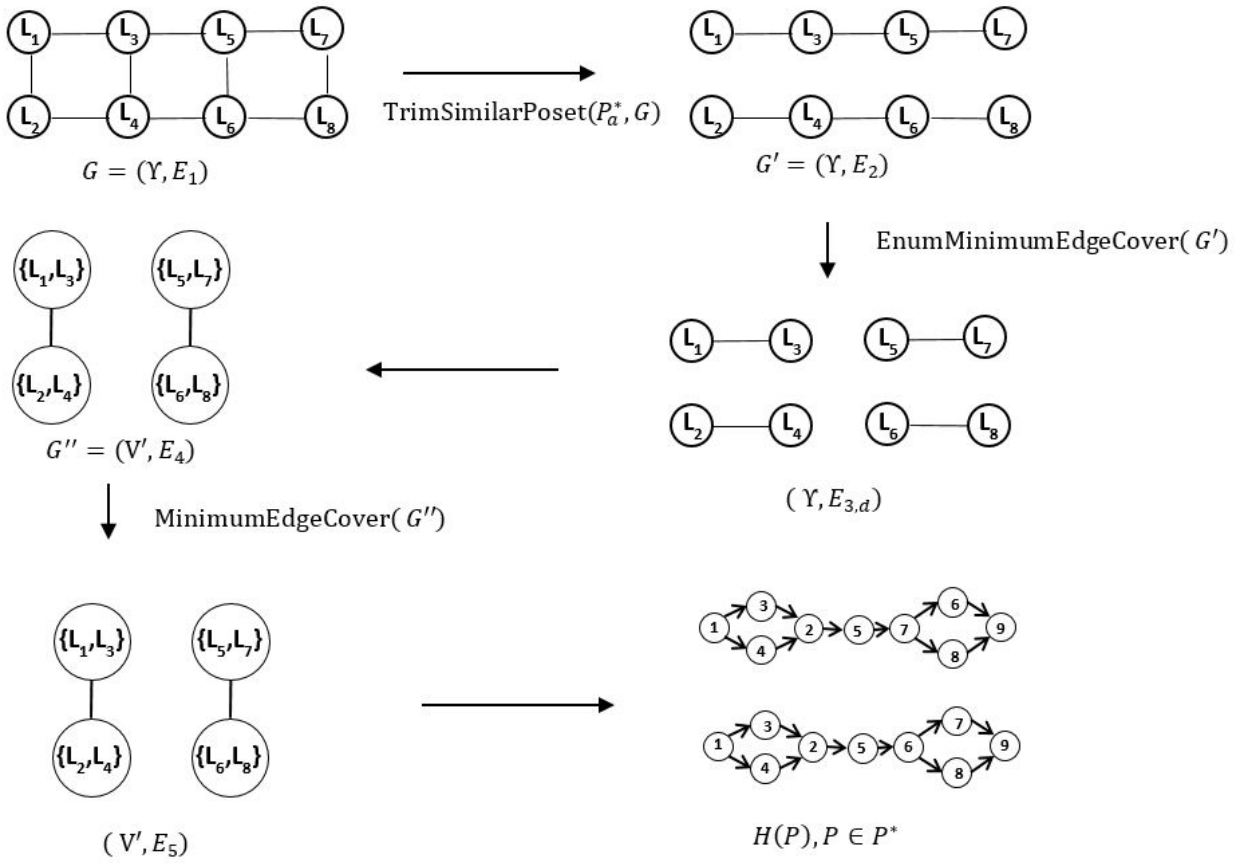

Figure 10. Algorithm 2 example.

Algorithm 2 Deterministic Algorithm for Minimum Hammock(2,2)-Poset Cover Problem

INPUT: A set $\Upsilon = \{L_1, L_2, \ldots, L_m\}$ of linear orders over set $\{1,2, \ldots, n\}$
OUTPUT: A set $P^* = \{P_1, P_2, \ldots, P_k\}$ of Hammock(2,2) posets where $\bigcup_{P_i \in P^*} \mathcal{L}(P_i) = \Upsilon$ and k is minimum

1: $P_a^* = \text{GetAllHammock2kPosets}(\Upsilon, 2)$
2: **if** $\{P_a^* = null\}$
3: **return** null
4: **else**
5: $G \leftarrow (\Upsilon, E_1)$ where $E_1 = \{\{L_1, L_2\} | L_1, L_2 \in \Upsilon, L_1$ and L_2 differ by single transposition$\}$
6: $\left(P_b^*, G' = (\Upsilon, E_2)\right) \leftarrow \text{TrimSimilarPoset}(P_a^*, G)$
7: $\mathcal{M} \leftarrow \text{EnumMinimumEdgeCover}(G')$
8: **for** each $E_{3,i} = \mathcal{M}[i]$, i =1 to $|\mathcal{M}|$
9: $E_{4,i} \leftarrow \emptyset$
10: $V_i' \leftarrow E_{3,i}$
11: $Paired\left[\left|E_{3,i}\right|\right] \leftarrow$ initialize all elements to $false$
12: **for** each pair $\{e_i, e_j\}$ such that $e_i, e_j \in E_{3,i}, e_i \neq e_j$
13: **if** $\exists P \in P_b^*$ where $\mathcal{L}(P_{e_x}) = e_i$ and $\mathcal{L}(P_{e_y}) = e_j$
14: $E_{4,i} \leftarrow E_{4,i} \cup \{e_i, e_j\}$
15: $P_{c,i}^* \leftarrow P_{c,i}^* \cup \{P\}$
16: $Paired[i] \leftarrow true$
17: $Paired[j] \leftarrow true$
18: **end if**
19: **end for**
20: **for** each unpaired $e_p \in E_3$, i.e., $Paired[p] = false$
21: Select one $P \in P_b^*$ where $\mathcal{L}(P_{e_x}) = e_p$
22: $E_{4,i} \leftarrow E_{4,i} \cup \{e_p, \mathcal{L}(P_{e_y})\}$
23: $V_i' \leftarrow V_i' \cup \{\mathcal{L}(P_{e_y})\}$
24: $P_{c,i}^* \leftarrow P_{c,i}^* \cup \{P\}$
25: **end for**
26: **end for**
27: $P_{c,d}^* \leftarrow \arg min\left(\left|P_{c,i}\right|, P_{c,i}\right)$ for i =1 to $|\mathcal{M}|$
28: $G'' \leftarrow (V', E_4)$ where $V' = V_d'$ and $E_4 = E_{4,d}$
29: $E_5 \leftarrow \text{MinimumEdgeCover}(G'')$
30: $P^* = \left\{P \in P_{c,d}^* \middle| \left\{\mathcal{L}(P_{e_x}), \mathcal{L}\left(P_{e_y}\right)\right\} \in E_5\right\}$
31: **return** P^*
32: **end if**

Figure 11. Algorithm 2 pseudocode.

node, or intersect in an edge. Moreover, for each C_i, one of its pair of alternating edges corresponds to P (as defined in Item 1b) and the other pair of alternating edges corresponds to P' (defined in Item 1b).

This follows from Lemma 2.4 and 2.5. $|\mathbb{C}| = \frac{|P_a^*|}{2}$ since there are two pairs of alternating edges in a C_4 and that every pair of alternating edges corresponds to a poset in P_a^*.

3) In Step 6 after calling TrimSimilarPoset subroutine

a. $P_b^* = \{P \in P_a^* | P$ is unique with respect to the relation, *i.e.*, there exists no other poset $P' \in P_b^*$ where $<_P = <_P'\}$.

From Item 1b, every poset in P_a^* is similar (with respect to the binary relation) to exactly one other poset in P_a^*; hence, we have PairPoset structure in TrimSimilarPoset subroutine containing every pair of similar poset. From the execution of Lines 15–50

of TrimSimilarPoset, for each pair of similar poset, only one poset is selected (marked as true).

b. $\bigcup_{P\in P_b^*} \mathcal{L}(P) = \Upsilon$.

$<_P = <'_P$ if and only if $\mathcal{L}(P) = \mathcal{L}(P)'$. Since only the redundant poset (with respect to the binary relation) is removed from TrimSimilarPoset subroutine, then $\bigcup_{P\in P_b^*} \mathcal{L}(P) = \bigcup_{P\in P_a^*} \mathcal{L}(P) = \Upsilon$.

c. Let $C = (L_1, L_2, L_3, L_4) \in \mathbb{C}$ with pair of alternating edges $\{\{L_1, L_2\}, L_3, L_4\}\}$ and $\{\{L_2, L_3\}, \{L_1, L_4\}\}$. Then, $\{\{L_1, L_2\}, \{L_3, L_4\}\} \subseteq E_2$ xor $\{\{L_2, L_3\}, \{L_1, L_4\}\} \subseteq E_2$.

Initially in Line 2, E_2 contains all edges of every $C \in \mathbb{C}$. As redundant poset P is removed in Lines 15–50, the corresponding alternating edges $\mathcal{L}(P_{e_x})$ and $\mathcal{L}(P_{e_y})$ are also removed from E_2. Hence, only one pair of alternating edges is selected for each $C \in \mathbb{C}$.

d. Let $C, C' \in \mathbb{C}$ and C, C' have a common edge, say $\{L_1, L_2\}$. Let $\{L_3, L_4\}$ and $\{L_5, L_6\}$ be the edges alternating to $\{L_1, L_2\}$ in C and C', respectively. Then $\{L_1, L_2\}, \{L_3, L_4\}, \{L_5, L_6\} \in E_2$ xor $\{L_1, L_2\}, \{L_3, L_4\}, \{L_5, L_6\} \notin E_2$.

This follows from Lines 39–48 of TrimSimilarPoset subroutine.

e. The return $G' = (V, E_2)$ of TrimSimilarPoset is a forest and $V = \Upsilon$. From Lemma 2.4 and 2.5, G can have cycles which have no intersection. This implies that G can be not connected. Hence, $G' \subset G$ can be also not connected. Now we show that G' has no cycles. Suppose it has a cycle say C, then C must have at least two edges that are adjacent in one of the cycles in $\mathbb{C}$. However, this cannot happen because in TrimSimilarPoset – for each cycle – only a pair of alternating edges is in E_2. Hence, G' is not connected and has no cycles. G' is a forest. Since only one pair of alternating edges in every cycle in $\mathbb{C}$ is removed, then the set of vertices of $V = \Upsilon$.

At this point, we can say that the minimum set of Hammock(2,2)-Poset that covers Υ is the minimum edge cover of G', which can be represented by minimum set of Hammock(2,2) in P_b^*. We can find such edge covering by checking all possible minimum edge covers of G'.

4) At Step 7, $\mathcal{M}$ contains all possible minimum edge covers of G'.

From Item 3e, G' is a forest; hence, it is bipartite. We can then use the algorithm Enum_ Maximum_ Matchings of Uno (1997) for bipartite graphs. For each maximum matching M, the Minimum Edge Cover of a graph can be solved by just adding to the maximum matching M an edge incident to every uncovered vertex (Norman & Rabin 1959).

5) In Step 27–28,

a. $E_{3,d} \in \mathcal{M}$ that can be represented by minimum set of posets $P_{c,d}^* \subseteq P_b^*$

b. $G'' = (V', E_4)$ where $E_{3,d} \subseteq V' \subseteq E_2$ and $E_4 = \{\{e_i, e_j\} | \exists\ P \in P_{c,d}^*$ where $\mathcal{L}(P_{e_x}) = e_i$ and $\mathcal{L}(P_{e_y}) = e_j\}$

Items 5a and 5b follow directly from the construction executed in Lines 8–28 of Algorithm 2.

$\bigcup_{P\in P_{c,d}^*} \mathcal{L}(P) = \Upsilon$

Since $P_{c,d}^* \subseteq P_b^*$ and from Item 3b $\bigcup_{P\in P_b^*} \mathcal{L}(P) = \Upsilon$, then $\bigcup_{P\in P_{c,d}^*} \mathcal{L}(P) \subseteq \Upsilon$. To show that $\Upsilon \subseteq \bigcup_{P\in P_{c,d}^*} \mathcal{L}(P)$, let $L_1 \in \Upsilon$. $E_{3,d}$ being an edge cover implies that there exists an edge $e \in E_{3,d}$ such that $L_1 \in e$. From the construction of $P_{c,d}^*$ from Lines 8–28, there exists a $P \in P_{c,d}^*$ that covers e. Hence, $L \in \bigcup_{P\in P_{c,d}^*} \mathcal{L}(P)$.

6) In Step 30, let E_5 be the minimum edge cover of G''. Then, $P^* = \{P \in P_{c,d}^* | \{\mathcal{L}(P_{e_x}), \mathcal{L}(P_{e_y})\} \in E_5\}$ is the minimum set of posets that cover exactly the input Υ.

a. $\bigcup_{P\in P^*} \mathcal{L}(P) = \Upsilon$
Since $P^* = P_{c,d}^*$ and from Item 5c, $\bigcup_{P\in P_{c,d}^*} \mathcal{L}(P) = \Upsilon$, then $\bigcup_{P\in P_{c,d}^*} \mathcal{L}(P) \subseteq \Upsilon$. To show that $\Upsilon \subseteq \bigcup_{P\in P_{c,d}^*} \mathcal{L}(P)$, let $L_1 \in \Upsilon$. $E_{3,d}$ being an edge cover implies that there exists an edge $e \in E_{3,d}$ such that $L_1 \in e$. $E_{3,d} \subseteq V'$, which is the set vertices of G''. The minimum edge cover of G'', which is E_5, then contains an edge in $E_{4,d}$ that covers e. The edge in $E_{4,d}$ corresponds to a poset that generates e. Hence, $L_1 \in \bigcup_{P\in P^*} \mathcal{L}(P)$. This means $\Upsilon \subseteq \bigcup_{P\in P^*} \mathcal{L}(P)$. Thus, $\Upsilon = \bigcup_{P\in P^*} \mathcal{L}(P)$.

b. P^* is minimum *i.e.*, there exists no other $P^{*\prime}$ where $|P^{*\prime}| < |P^*|$ that covers Υ.

Suppose there exists such $P^{*\prime}$ and let Hammock(2,2)-Poset $P \in P^* \backslash P^{*\prime}$. This means $\mathcal{L}(P)$ is already covered by some posets in $P^{*\prime}$. Let $P_1, P_2, \ldots, P_r$ be the posets that cover $\mathcal{L}(P)$ in $P^{*\prime}$. Given the possible intersection of the Hammock(2,2)-Posets from Lemma 2.5, then $2 \leq r \leq 4$. Note that $P^*, P^{*\prime} \subseteq P_a^*$; hence, we can also say that P, $P_1, P_2, \ldots, P_r \in P_a^*$. We then have the following three possibilities in getting P^* from P_a^* in lines 5–30 of the algorithm, which can be more clearly visualized in transposition graph in Figure 13.

i. r = 4
In this case, any edge of $G(P)$ should not have been included in Minimum Edge Cover $E_{3,d}$. Therefore, $P \notin P^*$.

ii. r = 3
In this case, if the alternating edges $\{L_1, L_3\}$ and $\{L_2, L_4\}$ are selected in E_2, then both are not included in Minimum Edge Cover $E_{3,d}$. On the other hand, if alternating edges $\{L_1, L_2\}$ and $\{L_3, L_4\}$ are selected in E_2, then only the edge $\{L_1, L_2\}$ is included in $E_{3,d}$. However, $\{L_3, L_4\}$ will not be included in V_d' since $\{L_1, L_2\}$ can already be paired to $\{L_5, L_6\}$ since $\{L_5, L_6\}$ must also be in E_2 by 2.d. Hence, in either condition, $P \notin P^*$.

iii. r = 2
In this case, if the alternating edges $\{L_1, L_3\}$ and $\{L_2, L_4\}$ are selected in E_2, then both are not included in Minimum Edge Cover $E_{3,d}$. On the other hand, if alternating edges $\{L_1, L_2\}$ and $\{L_3, L_4\}$ are selected in E_2, then both will also be included in $E_{3,d}$ and so in V_d'. However, it will not be included in P^* since E_5 – as minimum edge cover – will not include the edge $\{\{L_1, L_2\}, \{L_3, L_4\}\}$ as $\{L_1, L_2\}$ can be already covered by corresponding edge of P_1 while $\{L_3, L_4\}$ can be already covered by corresponding edge

```
1: function TrimSimilarPoset(P_a^* ,G=(V,E))
2:     G' = (V, E_2) ← initialize G' to be equal to G
3:     q ← 0
4:     for i ← 1 to |P_a^*|
5:         for j ← i + 1 to |P_a^*|
6:             if P_i = P_j
7:                 q ← q + 1
8:                 PairPoset[q][1] = P_i
9:                 PairPoset[q][2] = P_j
10:                break
11:            end if
12:        end for
13:    end for
14:    PairMark[q][2] ← initialize each element to null
15:    for r ← 1 to q
16:        if PairMark [r][1] = null and PairMark [r][2] = null
17:            PairMark[r][1] ← true
18:            Enqueue(Q, (r, 1))
19:            PairMark[r][2] ← false
20:            E_2 ← E_2 − {{L(PairPoset[r][2]_{e_x}), L(PairPoset[r][2]_{e_y})}}
21:            Enqueue(Q, (r, 2))
22:        else if PairMark[r][1] ≠ null or PairMark[r][2] = null
23:            PairMark[r][2] ← !PairMark[r][1]
24:            if PairMark[r][2] = false
25:                E_2 ← E_2 − {{L(PairPoset[r][2]_{e_x}), L(PairPoset[r][2]_{e_y})}}
26:            end if
27:            Enqueue(Q, (r, 2))
28:        else if PairMark[r][1] = null or PairMark[r][2] ≠ null
29:            PairMark[r][1] ← !PairMark[r][2]
30:            if PairMark[r][1] = false
31:                E_2 ← E_2 − {{L(PairPoset[r][2]_{e_x}), L(PairPoset[r][2]_{e_y})}}
32:            end if
33:            Enqueue(Q, (r, 1))
34:        else
35:            continue
36:        end if
37:        while Q is not empty
38:            (i, j) = Dequeue(Q)
39:            for each PairPoset[u][v] ∈ PairPoset where PairMark[u][v] = null
40:                if PairPoset[u][v]_{e_x} ∈ {PairPoset[i][j]_{e_x}, PairPoset[i][j]_{e_y}} or
41:                  PairPoset[u][v]_{e_y} ∈ {PairPoset[i][j]_{e_x}, PairPoset[i][j]_{e_y}}
42:                    PairMark[u][v] = PairMark[i][j]
43:                    if PairMark[u][v] = false
44:                        E_2 ← E_2 − {{L(PairPoset[u][v]_{e_x}), L(PairPoset[u][v]_{e_y})}}
45:                    end if
46:                    Enqueue(Q, (u, v))
47:                end if
48:            end for
49:        end while
50:    end for
51:    P_b^* = {PairPoset[i][j] | PairMark[i][j] = true}
52:    return P_b^* , G'
53:end function
```

Figure 12. TrimSimilarPoset subroutine.

of P_2. Hence, in either condition, $P \notin P^*$. Every case arrives at a contradiction. Hence, we can say that there exists no $P \in P^* \setminus P^{*\prime}$. In other words, $P^* \subseteq P^{*\prime}$. This means $|P^{*\prime}| \not< |P^*|$. There is no smaller set of poset $P^{*\prime}$ that covers Υ.

Now, we determine the time complexity of the algorithm. It is easy to see that the running time of the algorithm is dominated by the subroutines GetAllHammock2kPosets, TrimSimilarPoset, EnumMinimumEdgeCover, and MinimumEdgeCover. From Theorem 2.3, the running time in the call to GetAllHammock2kPosets is $O(n^2m^2)$. The transposition graph $G = (\Upsilon, E_1)$ in Step 5 – considering its properties as described in Item 2a, – has the greatest number of C_4 when every cycle $C \in \mathbb{C}$ share same edge, say $e \in E_1$. In this case, $|\mathbb{C}|=\frac{m-2}{2}$. Since every pair of alternating edge in G corresponds to a poset in P_a^*, then $|P_a^*| \leq \frac{m-2}{2} \cdot 2 = m-2$. Hence, each for or while loop in TrimSimilarPoset runs in $O(m)$. As in Theorem 2.3, comparing two posets take $O(n^2)$. Hence, the running time of TrimSimilarPoset is $O(n^2m^2)$.

From 4, EnumMinimumEdgeCover can use the algorithm Enum_ Maximum_Matchings of Uno (1997), which runs in $O(|E||V|^{\frac{1}{2}} + |V|\, M)$ where M is the number of possible maximum matchings in the graph. The largest possible size of $|E_1|$ is $\frac{m}{4}$ – this is when every cycle has no intersection with any other cycle in $|\mathbb{C}|$ $E_2 \subseteq E_1$. Hence, for graph $G' = (\Upsilon, E_2)$, the call to EnumMinimumEdgeCover runs in $O(m^{\frac{3}{2}} + mM)$.

Given M possible maximum matchings, Steps 8 – 26 then runs in $O(M \cdot (\frac{m}{2}) \cdot m)$ or $O(m^3M)$.

Lastly, for the MininumEdgeCover in Step 29, we can use the general maximum matching algorithm of Micala and Vazirani (Peterson & Loui 1988) that runs in $O(|V|^{\frac{1}{2}}\, |E|)$. Hence, getting Minimum Edge Cover for G'' runs in $O(m^{\frac{3}{2}})$.

Thus, the total running time of the algorithm is $O(n^2m^3 + m^3M)$. ■

Theorem 2.7. Edge Cover Problem $\leq_P$Hammock(2,2)-Poset Cover Problem

Proof. Suppose we have $G = (V, E)$ and $K \leq |E|$ be an instance of the Minimum Edge Cover Problem. Let $n_v = |V|$ and $n_E = |E|$. Let $V = \{v_1, v_2, \dots, v_{n_V}\}$ and $E = \{e_1, e_2, \dots, e_{n_E}\}$.

The idea of the reduction is to make the edge in G corresponds to a Hammock(2,2)-Poset so that getting a minimum edge cover corresponds to getting minimum Hammock(2,2) cover.

To create the set of linear orders, we need the following:

- $n' = 3(n_V + 2) + 1$
- $V' = \{1, 2, \dots, n'\}$
- $L_b = \{1, 2, \dots, n'\}$ (base linear order)
- $L[i] \leftarrow$ linear order obtained by swapping the elements in $3i - 1$ and $3i$ position in L_b, $1 \leq i \leq n_E + 1$
- $L[i,_j] = (L[i])[j]$ where $i \neq j$
- $C[L] = \{L, L[x], L[y], L[x, y]\}$ where
- $x = n_V + 1$, $y = n_V + 2$. Note that there is a unique Hammock(2,2) that covers $C[L]$. We can also call this unique Hammock(2,2) as cleanup poset.

For each vertex $v_1 \in V$, we denote $L_{v_i} = L[i]$. For each edge $e_t = \{v_r, v_s\} \in E$, we denote $L_{e_t} = L[r, s]$.

Now, we construct the corresponding instance of Hammock(2,2)-Poset Cover Problem.

$n' = 3(n_V + 2) + 1$
$V' = \{1, 2, \dots, n'\}$
$\Upsilon = \{L_b\} \cup \{L_{v_i} | v_i \in V\} \cup C[L_{e_1}] \cup \dots \cup C[L_{e_{nE}}]$
$K' = K + n_E$

Next, we show that G has an edge cover of size $\leq K$ if Υ has a Hammock(2,2)-Poset cover of size $\leq K'$.

Suppose G has an edge cover, say $E \subset E'$, where $E'| \leq K$. Note that there is Hammock(2,2)-Poset whose set of linear extension is $\{L_b, L_{v_i}, L_{v_j}, L_{e_r}\}$ where $e_r = \{v_i, v_j\}$. We denote this Hammock(2,2)-Poset as P_{e_r}. Hence, to cover L_b and the L_{v_i}'s. it is sufficient to choose K Hammock(2,2)-Posets P_{e_i} where $e_1 \in E'$. The remaining linear orders to cover are the linear orders in $C[L_{e_i}]$, $i = 1, 2, \dots, n_E$. However, as noted above, there is also a unique Hammock(2,2)-Poset that covers $C[L]$. Hence, Υ has a Hammock(2,2)-Poset cover of size $\leq K + n_E = K'$.

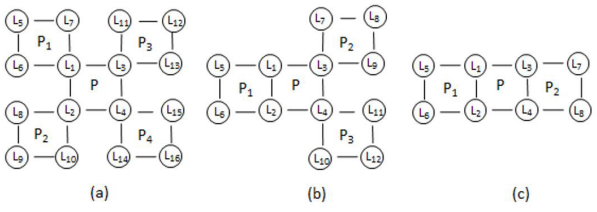

Figure 13. Possible cases in getting P^* from P_b^*.

On the other hand, suppose P^* be the Hammock(2,2)-Poset cover where $P^*| \leq K'$. P^* must contain the n_E cleanup posets plus some number of posets P_e's. It is clear that these posets P_e's correspond to an edge cover of size at most $K' - n_E = K$. ■

CONCLUSION

In this study, we have shown an optimal deterministic algorithm for Hammock(2,2)-Poset Cover Problem. The algorithm runs in $O(n^2m^3 + m^3M)$ where n is the size of set, m is the number of linear orders, and M is the number of all maximum matchings of a subgraph of the transposition graph of linear orders. Counting M is also a problem that is in #P-complete (Valiant 1979). In our knowledge, there is also no known upper-bound of M but it could go exponential with respect to the size of the graph. While the algorithm provides a solution to the problem, it does not have an implication on the computational complexity of the Hammock(2,2)-Poset Cover Problem. However, future study could try to improve the part that incurs O(M) time and determine if it can be efficiently done without getting all possible maximum matchings.

The results of this paper also include properties that show that the Hammock(2,2)-Poset Cover Problem is closely related to 4-Set Cover Problem, C_4 Cover Problem (or Square Packing Problem), and Edge Cover Problem – all of which are problems with complexity already known. These properties could give ideas on future studies on what problems can be reduced to or from Hammock(2,2)-Poset Cover Problem to determine its computational complexity.

ACKNOWLEDGMENTS

I. Ordanel acknowledges the support from DOST-ERDT.

H. Adorna acknowledges support from DOST-ERDT and Semirara Mining Corporation professorial chair.

REFERENCES

AGRAWAL R, GUNOPULOS D, LEYMANN F. 1998. Mining process models from workflow logs. Springer Berlin Heidelberg.

ARKIN A, SHENG P, ROSS J. 1997. A test case of correlation metric construction of a reaction pathway from measurements. Science 277: 1275–79.

BLÄSER M, MANTHEY B. 2002. Two approximation algorithms for 3-cycle covers. Berlin, Heidelberg: Springer.

CHASE PJ. 1973. Transposition Graphs. SIAM Journal on Computing 2(2): 128–133.

DER AALST WV, WEIJTERS T, MARUSTER L. 2004. Workflow mining: Discovering process models from event logs. Knowledge and Data Engineering, IEEE Transactions 16(9): 1128–42.

FERNANDEZ PL, HEATH LS, RAMAKRISHNAN N, TAN M, VERGARA JP. 2013. Mining Posets from Linear Orders. Discrete Mathematics, Algorithms and Applications 5(4).

GAREY M, JOHNSON D. 1979. A Guide to the Theory of NP-Completeness. New York: WH Freemann.

HEATH LS, NEMA AK. 2013. The Poset Cover Problem. Open Journal of Discrete Mathematics 3(03): 101–111.

LEE A, WILSON M. 2005. A Combinatorial Method for Analyzing Sequential Firing Patterns Involving an Arbitrary Number of Neurons Based on Relative Time Order. Journal of Neurophysiology 92(4): 2555–73.

MANNILA H. 2008. Finding Total and Partial Orders from Data for Seriation. In: International Conference on Discovery Science. Berlin, Heidelberg: Springer. p. 16–25.

NORMAN R, RABIN M. 1959. An algorithm for a minimum cover of a graph. Proceedings of the American Mathematical Society 10(2): 315–319.

ORDANEL I, ADORNA H. 2017. Two Approximation Algorithms to the Poset Cover Problem. Proceedings of the 17th Philippine Computing Science Congress. p. 179–184.

ORDANEL I, FERNANDEZ P. 2011. Reconstructing a Tree Poset from Linear Extensions. Philippine Information Technology Journal 4: 18–23.

ORDANEL I, ADORNA H, CLEMENTE J. 2017. Approximation of two simple variations of the Poset Cover Problem. Workshop on Computation: Theory and Practice (WCTP 2017), 12–13 Sep 2017, Osaka University Nakanoshima Center, Osaka, Japan.

PETERSON PA, LOUI M. 1988. The general maximum matching algorithm of Micali and Vazirani. Algorithmica 3(1–4): 511–533.

PUOLAMÄKI K, FORTELIUS M, MANNILA H. 2006. Seriation in Paleontological Data: Using Markov Chain Monte Carlo Methods. PLoS Computational Biology 2(2).

SANCHEZ GA, FERNANDEZ P, VERGARA JP. 2014. Some Heuristics for the 2-Poset Cover Problem. Philippine Computing Journal 9: 26–32.

UNNIKRISHNAN KP, RAMAKRISHNAN N, SASTRY PS, UTHURUSAMY R. 2006. Network reconstruction from dynamic data. ACM SIGKDD Exploration Newsletter 8(2): 90–91.

UNO T. 1997. Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs. International Symposium on Algorithms and Computation. Berlin, Heidelberg: Springer. p. 92–101.

VALIANT LG. 1979. The complexity of enumeration and reliability problems. SIAM Journal on Computing 8(3): 410–421.

WIGGINS C, NEMENMAN I. 2003. Process Pathway via Time Series Analysis. Experimental Mechanics 43(3): 361–370.