Mining Posets from Linear Orders

Proceso L. Fernandez[†], Lenwood S. Heath[*],
Naren Ramakrishnan[*], and John Paul C. Vergara[†]
[†] Department of Information Systems and Computer Science,
Ateneo de Manila University, Quezon City 1108, Philippines
[*] Department of Computer Science,
Virginia Tech, Blacksburg VA 24061, USA

Abstract

There has been much research on the combinatorial problem of generating the linear extensions of a given poset. This paper focuses on the reverse of that problem, where the input is a set of linear orders, and the goal is to construct a poset or set of posets that generates the input. Such a problem finds applications in computational neuroscience, systems biology, paleontology, and physical plant engineering. In this paper, several algorithms are presented for efficiently finding a single poset that generates the input set of linear orders. The variation of the problem where a minimum set of posets that cover the input is also explored. It is found that the problem is polynomially solvable for one class of simple posets (kite(2) posets) but NP-complete for a related class (hammock(2,2,2) posets).

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications - Data Mining; I.2.6 [Artificial Intelligence]: Learning

General Terms: Algorithms.

Keywords: partial orders, posets, linear extensions.

1 Introduction

With the growing popularity of knowledge discovery in databases (KDD) and its applications in numerous scientific domains [10, 14], algorithms for data mining have become a fertile ground for theoretical developments. Modern data mining algorithms process massive amounts of data, typically more than what can fit into main memory, and yield patterns that can be viewed either as compressed representations [21] or as generative models of data.

Many data mining algorithms can be viewed as addressing enumeration or counting problems. The classical example is the search for frequent subsets in a collection of sets [1], where frequency is determined by a user-determined *support* threshold. The support criterion is harnessed effectively by algorithms like *Apriori* [2] that enumerate all possible candidates and search levelwise, beginning with singleton subsets, estimating their support among the collection, and building upon those subsets that pass the threshold to explore bigger subsets. Researchers have since generalized the scope of such algorithms to finding sequential patterns from a collection of lists [3], frequent trees from a forest [31], and even frequent subgraphs from a collection of graphs [17].

In this paper, we focus on the task of *poset mining*, i.e., finding order constraints (expressed as partial orders) from a collection of total orders. Pei et al. [23] study this problem in the traditional framework of frequent pattern mining and present an Apriori-like algorithm for mining frequent posets. Mannila and Meek [20] cast a variation of this problem in a probabilistic setting and present algorithms that mine a specific category of posets. Ukkonen [29] introduces a scoring function to accurately define a "best-fit" poset or set of posets against the input set of total orders. Gionis et al. [12] seek *bucket orders* (total orders with ties), instead of posets.

Although these works offer much practical significance, few theoretical results on mining posets from linear orders have been published. In particular, since a poset can be viewed as a generator of linear extensions (orders), the underlying data mining problem is the converse

of the well-studied combinatorial problem of generating the set of linear extensions of a given poset, but has not been studied from a classical algorithmic perspective. Therefore, we focus on the most basic version of poset mining, where we are given a set of linear orders and we must find one poset (or a small number of posets) that generates the linear orders. We study the theoretical complexity of this problem, present a general framework to pose and study various inference tasks, and develop algorithms for mining restricted classes of posets. Our work finds applications in many domains where we seek to reconstruct system dynamics from sequential data traces, such as computational neuroscience [19], systems biology [4, 30], paleontology [27], and physical plant engineering [18], although we do not discuss these applications in detail in this paper.

The rest of the paper is structured as follows. Section 2 presents definitions and notations used in the paper. In Section 3, we formulate the problem of determining a single poset that generates the input set of linear orders, and present two algorithms for solving the problem. Section 4 investigates the same problem for restricted classes of posets. In Section 5, the problem of finding a minimum set of posets to cover the input set is formally defined, and complexity results for two poset classes are presented. We summarize our results and present future directions in Section 6.

2 Preliminaries

A (finite) *partially ordered set* or *poset* $P = (V, <_P)$ is a pair consisting of a finite set V and a binary relation $<_P \subseteq V \times V$ that is irreflexive, antisymmetric, and transitive. For any $u, v \in V$, we write $u <_P v$ if $(u, v) \in <_P$.

For a given poset $P = (V, <_P)$, we say that a pair of distinct elements $u, v \in V$ are *comparable in* P, written $u \perp_P v$, if either $u <_P v$ or $v <_P u$. Otherwise, u and v are *incomparable in* P, written $u \parallel_P v$. Moreover, if $u <_p v$ and there is no $w \in V$ such that $u <_P w <_P v$, then we say v *covers* u, written $u \prec_P v$, and also say that (u, v) is a *cover relation* in P.

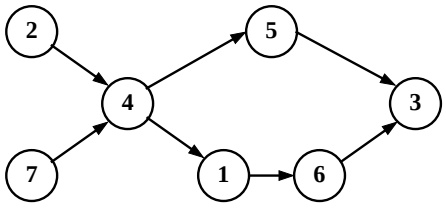

Figure 1: Hasse diagram of the example poset

A poset $P = (V, <_P)$ corresponds to a directed acyclic graph (DAG) $G = (V, E)$ with vertex set V and edge set $E = \{(u, v) \mid u <_P v\}$. The *Hasse diagram* $H(P)$ for the poset P is a drawing of the transitive reduction of the DAG G. Equivalently, the edge set of the Hasse diagram $H(P)$ consists of all cover relations (u, v) in P.

As an example, let $V = \{1, 2, 3, 4, 5, 6, 7\}$, and let

$$\begin{aligned} <_P \;\; = \;\; & \{(1,3),(1,6),(2,1),(2,3),(2,4),(2,5),(2,6),(4,1),(4,3), \\ & (4,5),(4,6),(5,3),(6,3),(7,1),(7,3),(7,4),(7,5),(7,6)\} \end{aligned}$$

be a binary relation on V. The reader may verify that $P = (V, <_P)$ is a poset. Its Hasse diagram $H(P)$ is in Figure 1.

When the Hasse diagram of a poset $P = (V, <_P)$ is a single path consisting of all the n elements of V, then the poset P is also called a *linear order*. Formally, a linear order $L = (V, <_L)$ is a poset such that $u \perp_L v$ for every pair of distinct elements $u, v \in V$. A linear order L determines a unique permutation $(v_1, v_2, \ldots, v_n)$ of the elements of V with $v_1 <_L v_2 <_L \cdots <_L v_n$. In this case, we use the notation $L[i] = v_i$ for the i^{th} element in the permutation and $L^{-1}[v_i] = i$ for the position of v_i.

Given two posets $P_1 = (V, <_{P_1})$ and $P_2 = (V, <_{P_2})$ over the same set, we say that P_2 is an *extension* of P_1, written $P_1 \sqsubseteq P_2$, if $<_{P_1} \subseteq <_{P_2}$. Moreover, if P_2 is a *linear order*, then we say that P_2 is a *linear extension* of P_1. For a given poset P, we denote its set of linear extensions by $\mathcal{L}(P)$, and say that P *generates* $\mathcal{L}(P)$. Generating the set of linear extensions of a given poset P is equivalent to generating all topological sorts of its Hasse diagram [8]. For the poset P whose Hasse diagram is shown in Figure 1, the set of linear

extensions is readily computed to be

$$\begin{aligned}\mathcal{L}(P) &= \{(2,7,4,1,5,6,3),(7,2,4,1,5,6,3),(2,7,4,1,6,5,3),\\ &\quad (7,2,4,1,6,5,3),(2,7,4,5,1,6,3),(7,2,4,5,1,6,3)\},\end{aligned}$$

where the six linear extensions are given in permutation notation. Note that, for any two posets P_1 and P_2 over V, if $P_1 \sqsubseteq P_2$, then $\mathcal{L}(P_1) \supseteq \mathcal{L}(P_2)$. Interestingly, the set of all posets on a given base set is also a partial order (with binary relation $\sqsubseteq$) and forms a semilattice [9]. The bottom of the lattice is the empty poset $(V, \emptyset)$, while the top consists of all the linear orders on the given base set.

Much attention has been given to the combinatorial problems of counting [5, 6] and generating the linear extensions of a given poset [7, 16, 22, 25, 28]. Brightwell and Winkler [6] prove that the problem of determining the number of linear extensions of a given poset is #P-complete. Pruesse and Ruskey [26] provide an algorithm that generates all linear extensions of a given poset, which may be exponential in number. Here, we investigate problems whose input is a set Υ of linear orders on a fixed base set V. The problem space that we have in mind results in a poset or set of posets that generates (or approximately generates) Υ, in the senses we develop in the next sections. In some of these problems, we restrict the poset or set of posets to specific classes. We now define those classes of posets.

A poset $P = (V, <_P)$ is called a *leveled poset* if the vertex set V can be partitioned into *levels* $V_1, V_2, \ldots, V_k$ such that, for $u \in V_i$ and $v \in V_j$, we have $u <_P v$ if and only if $i < j$. The sequence $V_1, V_2, \ldots, V_k$ is called a *leveling* of P. Figure 2 illustrates a leveled poset. To facilitate discussion of leveled posets, we write the leveled poset P as a function of the ordered set of sets:

$$P = \text{leveled}(V_1, V_2, \ldots, V_k).$$

The poset of Figure 2 is thus written

$$\text{leveled}\big(\{1,4,9\},\{5,10\},\{3,6,7,12\},\{2,8,11\}\big).$$

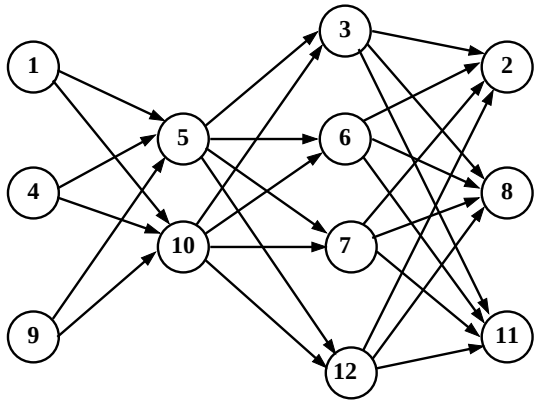

Figure 2: The Hasse diagram of a leveled poset on $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$

The definition for leveled posets implies a couple of important consequences. First, for distinct elements $u, v \in V_i$, we have $u \parallel_P v$. Second, the set of cover relations in $P = \text{leveled}(V_1, V_2, \ldots, V_k)$ is

$$\prec_P \;\; = \;\; \{(u, v) \mid u \in V_i \text{ and } v \in V_{i+1}, \text{ for } 1 \leq i < k\}.$$

Simple counting reveals that the number of linear extensions of a leveled poset P is

$$\left|\mathcal{L}\left(P\right)\right| \;\; = \;\; \prod_{1 \leq i \leq k} |V_i|!$$

Leveled posets belong to a larger class of posets called *graded posets*, of which the semilattice of posets is an example. By definition, a graded poset is a poset in which every maximal *chain* has the same length, where a *chain* is defined as a totally ordered subset of the poset [13].

Next, define a *hammock poset* to be a leveled poset where $|V_1| = |V_k| = 1$ and, for $2 \leq i \leq k-1$, either $|V_i| = 1$ or $|V_{i+1}| = 1$. Figure 3 shows the Hasse diagram of a hammock poset, with partition

$$\{3\}, \{4, 14\}, \{7\}, \{1\}, \{6\}, \{12, 9\}, \{11\}, \{2, 8, 13\}, \{10\}, \{5\}.$$

A non-singleton V_i in a hammock poset is a *hammock set* (or simply *hammock*), and its elements are *hammock vertices*. A vertex in a singleton partition, on the other hand, is

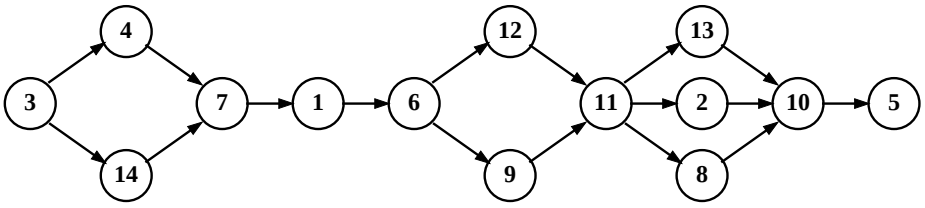

Figure 3: Hasse diagram of a hammock poset

called a *link vertex*. The hammock poset described in Figure 3 is more specifically called a hammock(2,2,3) poset to indicate the ordered sizes of the hammocks.

When a hammock poset has only one hammock, we call it a *kite poset*, or specifically a kite(k) poset if the size of the hammock is k. Kite posets are the simplest class of posets that we consider.

We also consider a *tree poset*, which is a poset whose Hasse diagram is a rooted directed tree, with each node (except for the root) covering exactly one other node. The class of tree posets belongs to a well-known class of posets called *series-parallel* posets, whose Hasse diagrams are series-parallel digraphs, and which naturally model certain electrical networks.

3 Generating Posets

The simplest nontrivial problem from the problem space we wish to explore asks whether there is a single poset that generates a set of linear orders.

Generating Poset

INSTANCE: A set $\Upsilon = \{L_1, L_2, \ldots, L_m\}$ of linear orders on $V = \{1, 2, \ldots, n\}$.

SOLUTION: A poset P such that $\mathcal{L}(P) = \Upsilon$.

We show that the Generating Poset problem can be solved in polynomial time.

Theorem 1. *The problem* Generating Poset *can be solved in* $O\left(mn^2\right)$ *time.*

Proof. The correctness and time-complexity of the GeneratingPosetOne algorithm in Figure 4 proves the theorem. In this algorithm, we build the binary relation $<_P$ by comput-

ALGORITHM: GeneratingPosetOne
INPUT: A set $\Upsilon = \{L_1, L_2, \ldots, L_m\}$ of linear orders on $V = \{1, 2, \ldots, n\}$.
OUTPUT: A poset P on V such that $\mathcal{L}(P) = \Upsilon$, if one exists.

```
1  <_P ← ⋂_{L∈Υ} <_L
2  if Υ = L(P)
3     then return P
4     else  return failure
```

Figure 4: First polynomial-time algorithm to solve Generating Poset

ing $\cap_{L\in\Upsilon} L$, the smallest partial order that contains all relations common to all the given linear orders. This can be done in $O\left(mn^2\right)$ time, since each of the m linear orders has $O\left(n^2\right)$ ordered pairs. By setting $<_P = \cap_{L\in\Upsilon} L$, we are sure that P generates all linear extensions, and no larger one can. It is sufficient, therefore, to show that P does not generate a linear extension outside Υ. This can be done by generating $\mathcal{L}(P)$ and verifying the linear extensions against Υ. At most $m+1$ linear extensions will be generated before we are sure whether or not $\mathcal{L}(P) = \Upsilon$. This can be done in $O(mn)$ time by using the algorithm of Pruesse and Ruskey [26] that generates the linear extensions of P in $O(n)$ amortized time per linear order. Therefore the GeneratingPosetOne algorithm in Figure 4 is correct, and runs in $O\left(mn^2\right)$ time. □

The first algorithm computes the elements of the binary relation of the desired poset while reading the input linear extensions. Another approach to solving the Generating Poset problem is to derive the cover relations first, then compute the transitive closure of the set of these cover relations to determine the desired poset. We first present two lemmas related to this approach.

Lemma 2. *Let $P = (V, <_P)$ be a poset, and let $u, v \in V$ be distinct. Then $u \parallel_P v$ if and only if there exists a linear order $L \in \mathcal{L}(P)$ such that $u \prec_L v$ and a linear order $L' \in \mathcal{L}(P)$ such that $v \prec_{L'} u$.*

Proof. First, assume that $u \parallel_P v$. Let $A = \{w \in V \mid w <_P u \text{ or } w <_P v\}$, $B = \{w \in V \mid u <_P w \text{ or } v <_P w\}$, and $C = V \setminus (A \cup B \cup \{u, v\})$. Since $u \parallel_P v$, we have $A \cap B = \emptyset$.

Moreover, for all $w \in C$, we have $u \parallel_P w$ and $v \parallel_P w$. Construct a linear order $L = (V, <_L)$ as follows. Start with a linear extension of $(A, <_P)$, follow it with a linear extension of $(C, <_P)$, follow that with u and then v, and conclude with a linear extension of $(B, <_P)$. (We can suggestively say that L matches the pattern $ACuvB$.) It is straightforward to check that L is a linear extension of P that has $u \prec_L v$. Similarly, to obtain L' such that $v \prec_{L'} u$, let L' be the same as L, except swap the positions of u and v. (L' matches the pattern $ACvuB$.)

Now, assume that there exist linear orders $L, L' \in \mathcal{L}(P)$ such that $u \prec_L v$ and $v \prec_{L'} u$. The existence of L implies that in the poset P, either $u <_p v$ or $u \parallel_P v$. The existence of L', on the other hand, implies that either $v <_P u$ or $v \parallel_P u$. It follows, therefore, that $u \parallel_P v$. □

Lemma 3. *Let $P = (V, <_P)$ be a poset, and let $u, v \in V$ be distinct. Then $u \prec_P v$ if and only if there exists a linear order $L \in \mathcal{L}(P)$ such that $u \prec_L v$ and there is no linear order $L' \in \mathcal{L}(P)$ such that $v \prec_{L'} u$.*

Proof. First, assume that $u \prec_P v$. Let $L \in \mathcal{L}(P)$ be any linear extension of P. As in the proof of Lemma 2, let $A = \{w \in V \mid w <_P u \text{ or } w <_P v\}$, $B = \{w \in V \mid u <_P w \text{ or } v <_P w\}$, and $C = V \setminus (A \cup B \cup \{u, v\})$. Construct a linear order $L = (V, <_L)$ as follows. Start with a linear extension of $(A, <_P)$, follow it with a linear extension of $(C, <_P)$, follow that with u and then v, and conclude with a linear extension of $(B, <_P)$. As in the proof of Lemma 2, L is a linear extension of P that has $u \prec_L v$. Since $u <_P v$, every linear extension L' of P has $u <_{L'} v$ and therefore not $v \prec_{L'} u$.

Now, assume that there exists a linear order $L \in \mathcal{L}(P)$ such that $u \prec_L v$ and there is no linear order $L' \in \mathcal{L}(P)$ such that $v \prec_{L'} u$. Since $u \prec_L v$, either $u \prec_P v$ or $u \parallel_P v$. Since there is no $L' \in \mathcal{L}(P)$ such that $v \prec_{L'} u$, by Lemma 2, u and v cannot be incomparable in P. Hence, $u <_P v$ and $u \prec_P v$. □

With Lemma 3, the cover relations of a poset can be computed from the cover relations of all of its linear extensions.

ALGORITHM: GeneratingPosetTwo
INPUT: A set $\Upsilon = \{L_1, L_2, \ldots, L_m\}$ of linear orders on $V = \{1, 2, \ldots, n\}$.
OUTPUT: A poset P on V such that $\mathcal{L}(P) = \Upsilon$, if one exists.

```
1   ▷ First, set up a matrix to record cover relations in all linear orders
2   for u ← 1 to n do for v ← 1 to n do C_{u,v} ← 0
3   ▷ Next, record each of the n − 1 cover relations for each of the m linear orders
4   for i ← 1 to m do for (u, v) ∈ ≺_{L_i} do C_{u,v} ← 1
5   ▷ Then extract the cover relations from the matrix
6   ≺_P ← ∅
7   for u ← 1 to n − 1
8       do for v ← u + 1 to n
9               do if C_{u,v} = 1 and C_{v,u} = 0
10                    then ≺_P ← ≺_P ∪ {(u, v)}
11                    else  if C_{u,v} = 0 and C_{v,u} = 1
12                                then ≺_P ← ≺_P ∪ {(v, u)}
13  <_P ← transitive closure (≺_P)
14  if Υ = L(P)
15    then return P
16    else  return failure
```

Figure 5: An $O\left(mn + n^3\right)$-time algorithm to solve Generating Poset

Theorem 4. *The* Generating Poset *problem can be solved in* $O\left(mn + n^3\right)$ *time.*

Proof. Algorithm GeneratingPosetTwo (Figure 5) is used to solve Generating Poset in the requisite time. Lemma 3, together with the uniqueness of the transitive closure, assures us that the poset P created in the two algorithms in Figures 4 and 5 are the same. This proves the correctness of the second algorithm. The time complexity of the algorithm is as follows. Setting up the $n \times n$ matrix C is done in $O\left(n^2\right)$. Steps 4 requires $O(mn)$ time since there are $n-1$ cover relations in each of the $O(m)$ linear orders. Extracting the cover relations in Steps 6–12 requires $O\left(n^2\right)$ time. Computing the transitive closure from the cover relations can be done in $O\left(n^3\right)$ time using the Floyd-Warshall algorithm. Finally, verifying the generated linear extensions of the computed poset requires $O(mn)$ time, as discussed in the proof for Theorem 1. Therefore, the algorithm GeneratingPosetTwo has running time $O\left(mn + n^3\right)$. The theorem follows. □

It should be noted that the number m of linear extensions of a given poset on n elements is, in the worst case, $n!$. Algorithm GeneratingPosetTwo is more efficient than algorithm GeneratingPosetOne when $mn+n^3$ is asymptotically smaller than mn^2, that is, when $n = o(m)$.

We conclude this section with a corollary that states a similar result for a more relevant version of the problem. The candidate poset computed by both algorithms presented in this section in fact produces the best poset possible, if we relax the problem constraints and allow the poset to generate extensions outside the input set:

Poset Super-Cover

INSTANCE: A nonempty set $\Upsilon = \{L_1, L_2, \ldots, L_m\}$ of linear orders on $V = \{1, 2, \ldots, n\}$.

SOLUTION: A poset P such that $\Upsilon \subseteq \mathcal{L}(P)$ and $\left|\mathcal{L}(P)\right|$ is minimum.

Note that what is desired is a poset that is able to generate all of the input linear orders with as few as possible extra linear extensions.

Corollary 5. *The* Poset Super-Cover *problem is solvable in polynomial time.*

Use either GeneratingPosetOne or GeneratingPosetTwo for the Generating Poset problem, but skip the verification step, and simply output the resulting poset.

4 Restricted Generating Poset Problem

If we have any knowledge about the structure of the poset that generates the set of linear extensions, then specialized algorithms may have time complexity better than that of either GeneratingPosetOne or GeneratingPosetTwo. In this section, we consider a restricted version of the Generating Poset problem and apply it to a few classes of posets.

Let C be a predicate applicable to posets (perhaps C characterizes the Hasse diagram of a poset). A poset on V that satisfies C is called a C-poset. Each such predicate defines a class of posets, namely, $\{P \mid P \text{ is a } C\text{-poset}\}$. We define a restricted version of the Generating Poset problem using a predicate C as follows.

Generating C-Poset ($\text{GENPOSET}_{\text{C}}$)

INSTANCE: A nonempty set $\Upsilon = \{L_1, L_2, \ldots, L_m\}$ of linear orders on $V = \{1, 2, \ldots, n\}$.

QUESTION: Is there a C-poset P on V that generates Υ, that is, such that $C(P)$ is true and $\mathcal{L}(P) = \Upsilon$?

As a simple example, let PATH be the predicate that describes a poset whose Hasse diagram is a simple path. There is exactly one linear extension associated with such a poset, namely, itself. Hence, $\text{GENPOSET}_{\text{PATH}}$ is trivially solvable in polynomial time (in fact, in $O(n)$ time).

Consider now the predicate TREE defining the class of tree posets. We show that $\text{GENPOSET}_{\text{TREE}}$ is solvable in $O\left(mn + n^2\right)$-time.

Theorem 6. *There is an $O\left(mn + n^2\right)$-time algorithm to solve* $\text{GENPOSET}_{\text{TREE}}$.

Proof. The algorithm that we present for this problem is the same as GeneratingPosetTwo, except for the construction of the transitive closure. Since each vertex $v \in V$

(except for the root) covers exactly one other vertex, we can construct all $(u, v) \in <_P$ by simply traversing the path from v to the root. This can be done in $O(n)$-time for each vertex, and therefore $O\left(n^2\right)$ overall. Thus the entire algorithm runs in $O\left(mn + n^2\right)$-time. The theorem follows. □

To develop an efficient algorithm for the GENERATING POSET problem applied to leveled posets, we first make the following definition and present some useful results related to it.

Define the *depth* $\text{depth}(v; P)$ of element v in poset P as 1 plus the number of elements less than v:

$$\text{depth}(v; P) \quad = \quad 1 + \left|\{u \in V \mid u <_P v\}\right|.$$

Thus, for a linear order $L = (v_1, v_2, ..., v_n)$, the depth of a vertex v_i is given by $L^{-1}[v_i] = i$. The depth of any vertex in a poset can be determined from the linear extensions of the poset, as follows.

Lemma 7. *Let $P = (V, <_P)$ be any poset. For any element $v \in V$,*

$$\text{depth}(v; P) \quad = \quad \min\left\{\text{depth}(v; L) \mid L \in \mathcal{L}(P)\right\}.$$

Proof. Let $A = \{u \in V \mid u <_P v\}$. Let $L \in \mathcal{L}(P)$ be any linear extension of P. Since $u <_P v$ implies $u <_L v$, we have $A \subseteq \{u \in V \mid u <_L v\}$ and hence $\text{depth}(v; P) \leq \text{depth}(v; L)$. Let $B = V \setminus (A \cup \{v\})$. Construct a linear order L' on V as a linear extension of $(A, <_P)$, followed by v, and followed by $(B, <_P)$. (L' matches the pattern AvB.) It is straightforward to check that L' is a linear extension of P. Moreover, $\text{depth}(v; L') = 1 + |A| = \text{depth}(v; P)$. We conclude that

$$\text{depth}(v; P) \quad = \quad \min\left\{\text{depth}(v; L) \mid L \in \mathcal{L}(P)\right\}.$$

□

Another result, specifically for leveled posets, is presented next.

Lemma 8. *Let $P = (V, <_P)$ be a leveled poset with leveling $V_1, V_2, \ldots, V_k$. Let $\ell_0 = 0$ and $\ell_i = \sum_{j=1}^{i} |V_j|$, for $1 \leq i \leq k$. Then, $|\mathcal{L}(P)| = \prod_{j=1}^{k} |V_j|! = \prod_{j=1}^{k} (\ell_j - \ell_{j-1})!$. Fix i such that $1 \leq i \leq k$, and fix $v \in V_i$. Then, for all $L \in \mathcal{L}(P)$, the depth of v in L satisfies $\ell_{i-1} + 1 \leq \text{depth}(v; L) \leq \ell_i$. Let d satisfy $\ell_{i-1} + 1 \leq d \leq \ell_i$. Then, the number of linear extensions L of P for which $\text{depth}(v; L) = d$ is $|\mathcal{L}(P)|/|V_i|$.*

Proof. A linear extension of P is constructed by starting with the $\ell_1 - \ell_0$ elements of V_1, in any order, followed by the $\ell_2 - \ell_1$ elements of V_2, in any order, and continuing until it is concluded by the $\ell_k - \ell_{k-1}$ elements of V_k, in any order. The number of ways of constructing a linear extension is therefore $|\mathcal{L}(P)| = \prod_{j=1}^{k} (\ell_j - \ell_{j-1})!$. If L is such a linear extension, then the depth of v in L satisfies $\ell_{i-1} + 1 \leq \text{depth}(v; L) \leq \ell_i$, by construction. Also by the construction, the number of linear extensions L of P for which $\text{depth}(v; L) = d$ is $|\mathcal{L}(P)|/|V_i|$. □

Theorem 9. *Let* LEVELED *be the predicate defining the class of leveled posets. There is an $O\left(mn + n^2\right)$-time algorithm to solve* $\text{GENPOSET}_{\text{LEVELED}}$.

Proof. Algorithm GeneratingLeveledPoset in Figure 6 solves the given problem. The correctness is established by Lemma 8. For the running time of the algorithm, setting up the Min and Max arrays is done in $O(n)$ time while updating them is done in $O(mn)$ time. Producing the sorted array A can be done in $O(n \log n)$ time using mergesort. Checking if Lemma 8 is satisfied in lines 7–11 is done in $O(n)$ time, while building the partial order is in $O\left(n^2\right)$ time. For the last part, checking whether the number of linear extensions of P matches the size of Υ can actually be done in constant time if the sizes of the vertex partitions are computed while performing the Lemma 8 check. Thus, the overall running time of the algorithm is $O\left(mn + n^2\right)$. The theorem follows. □

ALGORITHM: GeneratingLeveledPoset

INPUT: A set $\Upsilon = \{L_1, L_2, \ldots, L_m\}$ of linear orders on $V = \{1, 2, \ldots, n\}$.

OUTPUT: A leveled poset P on V such that $\mathcal{L}(P) = \Upsilon$, if one exists.

```
1   ▷ Derive the min and max depths of every vertex in every linear extension
2   for v ← 1 to n
3       do Min[v] ← min_{L∈Υ} depth(v; L)
4          Max[v] ← max_{L∈Υ} depth(v; L)
5   ▷ Check whether Lemma 8 is satisfied
6   A[1..n] ← V sorted by Min value
7   for j ← 1 to n − 1
8       do u ← A[j]
9          v ← A[j + 1]
10         if not ((Min[u] = Min[v] and Max[u] = Max[v]) or Max[u] < Max[v])
11            then return failure
12  ▷ Build the partial order, if passed the Lemma 8 check
13  <_P ← ∅
14  for u ← 1 to n − 1
15      do for v ← u + 1 to n
16             do if Min[u] < Min[v]
17                   then <_P ← <_P ∪ {(u, v)}
18                   else if Min[u] > Min[v]
19                           then <_P ← <_P ∪ {(v, u)}
20  if |Υ| = |L(P)|
21     then return P
22     else return failure
```

Figure 6: A polynomial-time algorithm to solve $\text{GENPOSET}_{\text{LEVELED}}$

5 Poset Cover Problem

A *poset cover* for a set Υ of linear orders on V is a set $\mathcal{P}$ of posets such that the union of all linear extensions of all posets in $\mathcal{P}$ is Υ, that is, such that $\Upsilon = \bigcup_{P \in \mathcal{P}} \mathcal{L}(P)$. There is always at least one poset cover of Υ, since Υ is a poset cover of itself. The computationally interesting problem is to minimize the number of posets in a poset cover. As a decision problem, this is the following.

Poset Cover

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

QUESTION: A poset cover $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ of Υ such that $k \leq K$.

Most sets of linear orders do not have a corresponding generating poset. Hence, the Poset Cover problem is usually the one that must be addressed. Heath and Nema [15], however, have recently proved that Poset Cover is NP-complete. Hence, to investigate polynomial-time solvable variants of Poset Cover, we restrict our attention to poset covers whose elements come from a particular class. Let C be any predicate defined on posets. The restricted decision problem is the following.

C-Poset Cover (COVER$_C$)

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

QUESTION: A poset cover $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ of Υ such that $k \leq K$ and such that $C(P_i)$ is true for every $P_i \in \mathcal{P}$.

We begin by developing notation for partial covers, which will be the candidate members for the poset cover.

A poset $P = (V, <_P)$ is a *partial cover* of Υ if $\mathcal{L}(P) \subseteq \Upsilon$, that is, if every linear extension of P is one of the linear orders in Υ. A partial cover $P = (V, <_P)$ is *maximal* in Υ if there is no poset $P' \neq P$ on V such that $P' \sqsubseteq P$ and $\mathcal{L}(P') \subseteq \Upsilon$.

The number of partial cover posets for a set Υ of m linear orders may be exponential in m. However, if we restrict our attention to some particular classes of posets, it may be possible to show that the number of maximal partial covers in that class is polynomial in m and indeed can be generated in polynomial time.

Recall that a kite poset is a hammock poset with a single hammock. For kite posets, there exists a polynomial time algorithm for determining partial covers of such types from a given set of linear orders, as shown in the next theorem.

Theorem 10. *Let $\Upsilon = \{L_1, L_2, \ldots, L_m\}$ be a nonempty set of linear orders on $V = \{1, 2, \ldots, n\}$. The set of all partial cover kites for Υ can be generated in $O\left(mn^3\right)$ time.*

Proof. Let P be a kite poset that is a partial cover of Υ. Let $V_h \subset V$ be its hammock, and let $u, v \in V$ be the unique elements such that, for all $w \in V_h$, we have $u \prec_P w \prec_P v$. Let $k = |V_h|$, let $i = \text{depth}(u; P)$, and let $j = \text{depth}(v; P)$. It follows that $1 \leq i < j \leq n$ and $j - i = k + 1 \geq 3$. We search for the elements u and v by considering the $O\left(n^2\right)$ possible i, j pairs. For a linear order $L = (v_1, v_2, \ldots, v_n)$, define its i, j-restriction to be

$$L(i, j) \quad = \quad (v_1, v_2, \ldots, v_{i-1}, v_i, v_j, v_{j+1}, \ldots, v_n).$$

For a given i, j pair, sort the elements of Υ by their i, j restrictions, ordered by the entries from the leftmost to the rightmost. This can be done in $O(mn)$ time using radix sort. Let $L_r \in \Upsilon$. There is a partial cover kite(k) poset that has linear extension L_r if and only if there are $k!$ elements of Υ having the same i, j restriction as L_r. This is easily determined by scanning the sorted linear orders in $O(mn)$ time. Thus, detecting partial cover kites requires $O\left(mn^3\right)$ time.

Each kite requires $O\left(n^2\right)$ time to construct. We now count the number of possible kite posets that can be returned by this algorithm. Fix k, where $2 \leq k \leq n - 2$. There are $n - k - 1$ possible i, j pairs for kite(k) posets. For a fixed i, j pair, there are at most $m/k!$

kite(k) posets. We obtain the following upper bound on the total number of kite posets:

$$\begin{aligned}\sum_{k=2}^{n-2} \frac{m(n-k-1)}{k!} &\leq mn \sum_{k=2}^{n-2} \frac{1}{k!} \\ &< mne \\ &= O(mn).\end{aligned}$$

We conclude that it takes $O\left(mn^3\right)$ time to construct $O(mn)$ kite posets. Consequently, the running time of this algorithm is $O\left(mn^3\right)$. The theorem follows. □

Theorem 11. $\text{COVER}_{\text{KITE(2)}}$ *can be solved in* $O\left(m^{1.5}n + mn^3\right)$ *time.*

Proof. For a set Υ of linear orders on V, the set of all partial cover posets that satisfy the predicate KITE(2) can be generated in $O\left(mn^3\right)$ time as shown in the proof of Theorem 10. Let p be the number of partial cover posets returned; clearly, $p = O(mn)$, since every linear order is associated with $n-3$ kite posets satisfying KITE(2). Construct an undirected graph with vertex set Υ and an edge between L_r and L_s if one of the generated posets has both L_r and L_s as linear extensions. This graph has m vertices and p edges. We need to choose a minimum set of edges such that every linear order is incident on one of the edges. This can be accomplished as follows. Find a maximum matching in the graph using the algorithm of Micali and Vazirani [24], which runs in $O\left(m^{1/2}p\right) = O\left(m^{1.5}n\right)$ time. Choosing the kite poset for each of the edges in a maximum matching plus one edge for every unmatched vertex yields an optimal solution to $\text{COVER}_{\text{KITE(2)}}$. □

We move our attention to hammock(2,2,2) posets; let HAMMOCK$(2,2,2)$ be the predicate that describes such posets. We now show the NP-completeness of $\text{COVER}_{\text{HAMMOCK(2,2,2)}}$, using a reduction similar to that in Heath and Nema [15]. In particular, we reduce from Cubic Vertex Cover, a known NP-complete problem (see [11]), which is defined here.

Cubic Vertex Cover

INSTANCE: A nonempty undirected graph $G = (V, E)$ that is cubic, that is, in which every vertex has degree 3; and an integer $K \leq |V|$.

QUESTION: Is there a subset $V' \subset V$ of cardinality K or less such that every edge in E is incident on at least one vertex in V'?

The main idea in the reduction is to represent edges with linear extensions, and vertices with hammock(2,2,2) posets, so that a linear extension representing an edge (u, v) can only be covered by the hammock posets representing the vertices u and v. We construct it in such a way that if a vertex cover contains a particular vertex, then the corresponding poset cover contains the corresponding hammock poset. This intuition is further elucidated in the proof of the following theorem.

Theorem 12. *The problem* $\text{COVER}_{\text{HAMMOCK(2,2,2)}}$ *is NP-complete.*

Proof. First, we show that $\text{COVER}_{\text{HAMMOCK(2,2,2)}}$ is in the class NP. Let $\Upsilon = \{L_1, L_2, \ldots, L_m\}$, $V = \{1, 2, \ldots, n\}$, and K, where $K \leq m$, constitute an instance of $\text{COVER}_{\text{HAMMOCK(2,2,2)}}$. Let $\mathcal{P}$ be a set of posets. First, check whether each poset in $\mathcal{P}$ satisfies HAMMOCK$(2, 2, 2)$. Second, check that $|\mathcal{P}| \leq K$. Third and finally, check that $\Upsilon = \bigcup_{P \in \mathcal{P}} \mathcal{L}(P)$. Generate the linear extensions of every poset $P \in \mathcal{P}$, and collect these in the set Υ'. If $\Upsilon' = \Upsilon$, then we have a poset cover. Each hammock(2,2,2) poset has exactly $2!2!2! = 8$ linear extensions that can be easily generated in polynomial time. Collecting these linear extensions into a single set Υ' and then comparing this set with Υ can also be done in polynomial time using known efficient algorithms for set union and comparison. Thus, $\text{COVER}_{\text{HAMMOCK(2,2,2)}}$ is in NP. To complete the proof of the theorem, we show a polynomial-time reduction from Cubic Vertex Cover to $\text{COVER}_{\text{HAMMOCK(2,2,2)}}$.

Let $G = (V_G, E_G)$ and $K' \leq |V|$ be an instance of the Cubic Vertex Cover problem. If $n_v = |V_G|$ and $n_e = |E_G|$, then $n_e = 3n_v/2$, since G is a cubic graph. Figure 7 shows an example of a cubic graph with $n_v = 6$ and $n_e = 9$. Construct the corresponding instance of $\text{COVER}_{\text{HAMMOCK(2,2,2)}}$ as follows. Let $n = 3(n_e+3)+1$, so the base set is $V = \{1, 2, \ldots, n\}$. Let the *base linear order* be $L_b = (1, 2, \ldots, n)$, here written in permutation notation. For

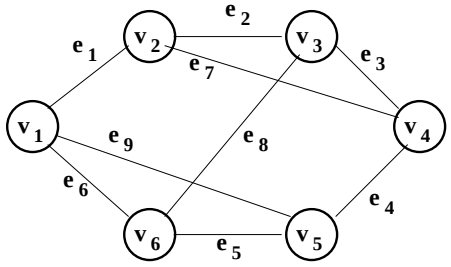

Figure 7: A cubic graph example.

our example, $n = 37$ and

$$\begin{aligned} L_b &= (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,\\ &\quad 21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37). \end{aligned}$$

For $1 \leq i \leq n_e + 3$, define the *i-swap pair* to be $(3i-1, 3i)$. If L is a linear order on V, then its *i-swap* $L[i]$ is obtained from L by swapping the two elements of V in its i-swap pair. We extend the notation to any number of swaps, so that $L[i,j,k] = ((L[i])[j])[k]$. (Note that the order of swapping does not matter, since different swap pairs are disjoint.) Without loss of generality, assume that the edges of G are $e_1, e_2, \ldots, e_{n_e}$. The *linear order for* e_i is $L_{e_i} = L_b[i]$. For example, for edge e_2, we have

$$\begin{aligned} L_{e_2} &= (1,2,3,4,6,5,7,8,9,10,11,12,13,14,15,16,17,18,19,20,\\ &\quad 21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37). \end{aligned}$$

Let $x = n_e + 1$, $y = n_e + 2$, and $z = n_e + 3$. Let L be a linear order on V. Define the set of *cleanup linear orders* for L to be

$$C[L] = \{L, L[x], L[y], L[z], L[x,y], L[x,z], L[y,z], L[x,y,z]\}.$$

Then there is a unique hammock(2,2,2) poset that covers $C[L]$; call that poset $P[L]$.

We choose the set of linear orders to be $\Upsilon = \{L_b\} \cup \{L_{e_i} \mid e_i \in E_G\} \cup C$, where the *cleanup set* C is defined later. Fix $v \in V_G$. Assume that v is incident on edges e_i, e_j, and e_k. Define

the poset P_v to be the unique hammock(2,2,2) poset such that $\{L_b, L_{e_i}, L_{e_j}, L_{e_k}\} \subset \mathcal{L}(P_v)$. Note that, in fact,

$$\mathcal{L}(P_v) = \{L_b, L[i], L[j], L[k], L[i,j], L[i,k], L[j,k], L[i,j,k]\}.$$

Since we want the P_v's to be the posets to cover the linear orders for the edges, we must put the additional four linear orders in C. Since we do not want to be forced to choose every P_v, we must put even more linear orders in C to provide alternative posets to cover the additional four linear orders. In particular, define the set

$$L_v = C[L[i,j]] \cup C[L[i,k]] \cup C[L[j,k]] \cup C[L[i,j,k]].$$

L_v contains all additional four linear orders in $\mathcal{L}(P_v)$. Moreover, L_v is exactly covered by the cleanup posets $P[L[i,j]]$, $P[L[i,k]]$, $P[L[j,k]]$, and $P[L[i,j,k]]$.

We can now define C, which is

$$C = \bigcup_{v \in V_G} L_v.$$

Careful counting shows that $|C| = 32n_v$ and that $|\Upsilon| = 1 + n_e + 32n_v$.

To complete the instance of $\text{COVER}_{\text{HAMMOCK(2,2,2)}}$, define $K = K' + 4n_v$.

We claim that G has a vertex cover of size $\leq K'$ if and only if Υ has a hammock(2,2,2) cover of size $\leq K$. First, suppose that G has a vertex cover of size $\leq K'$. Let that vertex cover be $V' \subseteq V_G$. Any hammock(2,2,2) cover of Υ must contain the $4n_v$ cleanup posets. To cover the base linear order and the edge linear orders, it suffices to choose the K' vertex posets P_v, where $v \in V'$. Hence, Υ has a hammock(2,2,2) cover of size $|V'| + 4n_v \leq K' + 4n_v = K$, as required. Now, suppose that $\mathcal{P}$ is a hammock(2,2,2) cover of size $|\mathcal{P}| \leq K$. As before, $\mathcal{P}$ must contain all $4n_v$ cleanup posets, plus some number of vertex posets P_v. It is clear that these vertex posets correspond to a vertex cover of size at most $K - 4n_v = K'$, as required.

It is easy to see that (Υ, V, K) can be constructed in polynomial time in the size of the Cubic Vertex Cover instance. Hence, we have demonstrated a polynomial-time reduction of Cubic Vertex Cover to $\text{COVER}_{\text{HAMMOCK(2,2,2)}}$. The theorem follows.

□

6 Conclusions

This paper has formalized problems related to identifying sets of posets that summarize or compress order-theoretic data sets. Through formalization, we hope to open the door for greater research into these problems. We have presented polynomial-time algorithms for Generating Poset and provided some complexity results for Poset Cover. While the problems bear much resemblance to classical set cover problems, they also have striking differences, as the objects to be used in a solution are only available implicitly, rather than explicitly given as in set cover problems. Future work may be directed with complexity results on relaxed variations of Generating Poset that allow the poset to generate a majority of the input set. There are also variations of Poset Cover that ask for approximate solutions. For example, one might allow a solution that is a set of posets that has linear extensions outside of the input set of linear orders; in this case, one must decide what it means to have a good approximation.

Acknowledgements

This research was supported in part by NSF Grant ITR-0428344.

References

[1] R. Agrawal, T. Imielinski, and A.N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'93)*, pages 207–216, May 1993.

[2] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB'94)*, pages 487–499, Sep 1994.

[3] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proceedings of the Eleventh International Conference on Data Engineering (ICDE'95)*, pages 3–14, March 1995.

[4] A. Arkin, P. Shen, and J. Ross. A test case of correlation metric construction of a reaction pathway from measurements. *Science*, Vol. 277(5330):pages 1275–1279, Aug 1997.

[5] G. Brightwell, H. J. Promel, and A. Steger. The average number of linear extensions of a partial order. *Journal of Combinatorial Theory Series A*, 73(2):pages 193–206, 1996.

[6] G. Brightwell and P. Winkler. Counting linear extensions. *Order*, Vol. 8(3):pages 225–242, 1991.

[7] E. R. Canfield and S. G. Williamson. A loop-free algorithm for generating the linear extensions of a poset. *Order*, 12(1):pages 57–75, 1995.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[9] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, UK, second edition, 2002.

[10] U.M. Fayyad and R. Uthurusamy. Evolving Data into Mining Solutions for Insights. *Communications of the ACM*, Vol. 45(8):pages 28–31, Aug 2002.

[11] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:pages 237–267, 1976.

[12] A. Gionis, H Mannila, Kai Puolamaki, and Anttii Ukkonen. Algorithms for discovering bucket orders from data. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)*, pages 561–566, 2006.

[13] J. R. Griggs. Matchings, cutsets, and chain partitions in graded posets. *Discrete Mathematics*, 144(1-3):pages 33–46, 1995.

[14] J. Han, R.B. Altman, V. Kumar, H. Mannila, and D. Pregiborn. Emerging Scientific Applications in Data Mining. *Communications of the ACM*, Vol. 45(8):pages 54–58, Aug 2002.

[15] L.S. Heath and A.K. Nema. The poset cover problem. Submitted, 2007.

[16] J. F. Korsh and P. LaFollette. Loopless generation of linear extensions of a poset. *Order*, 19(2):pages 115–126, 2002.

[17] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. In *Proceedings of the First IEEE International Conference on Data Mining (ICDM'01)*, pages 313–320, 2001.

[18] S. Laxman, P.S. Sastry, and K.P. Unnikrishnan. Discovering frequent episodes and learning hidden Markov models: A formal connection. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17(11):pages 1505–1517, 2005.

[19] A. K. Lee and M. A. Wilson. A combinatorial method for analyzing sequential firing patterns involving an arbitrary number of neurons based on relative time order. *Journal of Neurophysiology*, Vol. 92(4):pages 2555–2573, 2004.

[20] H. Mannila and C. Meek. Global partial orders from sequential data. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*, pages 161–168, 2000.

[21] H. Mannila and H. Toivonen. Multiple Uses of Frequent Sets and Condensed Representations (Extended Abstract). In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 189–194, Aug 1996.

[22] A. Ono and S. Nakano. Constant time generation of linear extensions. In *Proceedings of the 15th International Symposium on Fundamentals of Computation Theory (FCT 2005)*, volume 3623, pages 445–453, 2005.

[23] J. Pei, J. Lui, K. Wang, J. Wang, and P.S. Yu. Discovering frequent closed partial orders from strings. *IEEE Transactions on Knowledge and Data Engineering*, 18(11):pages 1467–1481, 2006.

[24] P.A. Peterson and M.C. Loui. The general maximum matching algorithm of Micali and Vazirani. *Algorithmica*, Vol. 3:pages 511–533, 1988.

[25] G. Pruesse and F. Ruskey. Generating the linear extensions of certain posets by transpositions. *SIAM Journal on Discrete Mathematics*, 4(3):pages 413–422, 1991.

[26] G. Pruesse and F. Ruskey. Generating linear extensions fast. *SIAM Journal on Computing*, Vol. 23(2):pages 373–386, 1994.

[27] K. Puolamaki, M. Fortelius, and H. Mannila. Seriation in paleontological data: Using Markov Chain Monte Carlo methods. *PLoS Computational Biology*, Vol. 2(2), Feb 2006.

[28] F. Ruskey. Generating linear extensions of posets by transpositions. *Journal of Combinatorial Theory Series B*, 54(1):pages 77–101, 1992.

[29] Antti Ukkonen. Data mining techniques for discovering partial orders. Master's thesis, Helsinki University of Technology, Helsinki, 2004.

[30] C.H. Wiggins and I. Nemenman. Process pathway inference via time series analysis. *Experimental Mechanics*, Vol. 43(3):pages 361–370, Sep 2003.

[31] M.J. Zaki. Efficiently Mining Frequent Trees in a Forest: Algorithms and Applications. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17(8):1021–1035, 2005.