Covering Linear Orders with Posets

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

Much research has been done 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. Several variations of the problem are addressed. Algorithms are presented for constructing posets whose set of linear extensions is a subset of the input. Finally, it is shown that the problem of finding the minimum set of posets that cover the input 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

The problem of reconstructing system dynamics from sequential data traces is an important one, with applications in computational neuroscience [10], systems biology [1, 18], paleontology [16], and physical plant engineering [9]. In these applications, we are given time-indexed discrete symbolic sequences or continuous-valued measurements, and the aim is to recover an underlying system-wide network model (reflecting connectivity, hierarchy, or strength of influences) of the observed temporal data. A key step in such network reconstruction is to elucidate order-theoretic constraints (e.g., lag, lead, or lack thereof) among the system variables underlying a given dataset.

In neuroscience, for instance, the goal is to ascertain the connectivity of the neuronal network from sequential information (time stamps, delays) about individual neuron firings. In systems biology, researchers seek to reconstruct an underlying reaction pathway by studying correlations between enzyme concentrations and protein levels. In paleontology, the approach is to infer evolutionary relationships among various taxa, in particular whether evidence supports that one species definitely originated before another species. Finally, in physical plant engineering, the data comprises symbol sequences indicative of process stages and diagnostics, and the goal is to identify causative relationships that might precede an event of interest.

Network reconstruction algorithms for unraveling system dynamics fall into two main categories. In the first category, we assume a generative model for data and seek to infer parameters of this model, conditioned on observed data. This is typically approached probabilistically, e.g., using ML or MAP estimation. In the second category, we identify constraints inferable from the given data and attempt to piece together these constraints into a system-wide model that summarizes, reconciles, or compresses the individual constraints.

Here, we formalize problems that require the inference of order constraints from sequential data to reconstruct partial order information, in particular, to infer partially ordered sets (posets) from linear extensions. Mannila and Meek [11] studied a version of these

problems; they cast the problem in a probabilistic setting and also presented algorithms to mine a specific category of posets. We carefully study the theoretical complexity of this problem, present a general framework to pose and study various inference tasks and algorithms for mining restricted classes of posets.

The contexts of the remainder of the paper are as follows. In the next section, some definitions and notations used in the paper are presented. Section 3 then covers the problem of determining a single poset that generates the input linear order, and presents two algorithms for solving the problem. The same problem is investigated for restricted class of posets in the next section. In Section 5, two variations of the problem are discussed, and these two may be aptly described as over-generation and under-generation. Section 6 introduces two data structures for storing the input more compactly, and shows how they can be used by algorithms for deriving the output. The next section covers how to construct a class of posets inductively, and also shows how this can be done using a data structure in the previous section. Finally, in Section 8, the problem of finding a mininmum set of posets to cover the input is formally defined, and the complexity results for two classes are presented.

2 Preliminaries

Let V be a finite set of cardinality $n \geq 0$. A (finite) *partially ordered set* or *poset* $P = (V, <_P)$ is a pair consisting of a vertex 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.

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

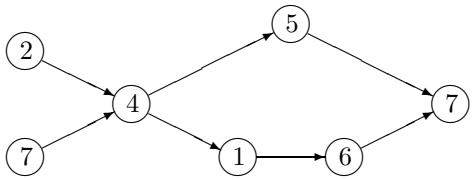


Figure 1: Hasse diagram of the example poset

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

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

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 $v_1 <_L v_2 <_L \cdots <_L v_n$.

Given two posets $P_1 = (V, <_{P_1})$ and $P_2 = (V, <_{P_2})$ over the same vertex 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 [5]. For the poset P whose Hasse diagram is shown in Figure 1, the set of linear

extensions is readily computed to be

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

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 semi-lattice. The bottom of the lattice is the empty poset, 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 [2, 3] and generating the linear extensions of a given poset [4, 8, 12, 14, 17]. Brightwell and Winkler [3] prove that the problem of determining the number of linear extensions of a given poset is #P-complete. Pruesse and Ruskey [15] provide an algorithm that generates all linear extensions of a given poset, which may be exponential in n in number. In this paper, 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 any $u \in V_i$ and $v \in V_j$, $u <_P v$ if and only if $i < j$. Figure 2 illustrates a leveled poset. To facilitate discussion of leveled posets, we denote the leveled poset P by the ordered set of sets

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

The poset of Figure 2 is thus written $\text{leveled}\big(\{1,4,9\},\{5,10\},\{3,6,7,12\},\{2,8,11\}\big)$.

The definition for leveled posets implies a couple of important consequences. First, for

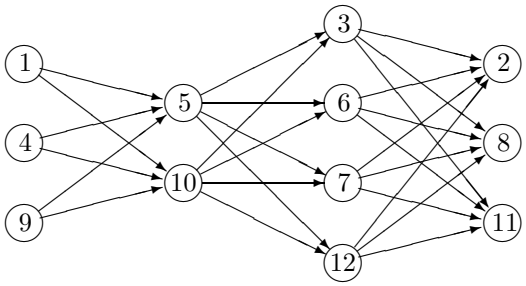


Figure 2: The Hasse diagram of a leveled poset

distinct elements $u, v \in V_i$, we have $u \parallel_P v$. Second, the cover relations in P consist of

$$\prec_P = \{(u, v) \mid u \in V_i \text{ and } v \in V_{i+1}\}$$

Simple counting will also reveal that the number of linear extensions of a leveled poset P is given by

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

Leveled posets belong to a larger class of posets called *graded posets*, of which the semi-lattice 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 [6].

Next, define a *hammock poset* as a leveled poset where the first and last partitions (V_1 and V_k) consist of a single element each, and where for $2 \le i \le k-1$ either V_i and V_{i+1} is a singleton. Figure 2 shows the Hasse diagram of a hammock-poset, with partition

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

In this paper, we call a non-singleton V_i in a hammock poset as a *hammock set* (or simply *hammock*), and call its elements *hammock vertices*. A vertex in a singleton partition, on the other hand, is called a *link vertex*. The hammock poset described in Figure 2 is more specifically called a hammock(2,2,3)-poset to indicate the ordered sizes

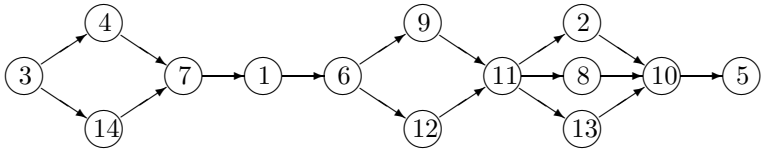


Figure 3: Hasse diagram of a hammock-poset

of the hammocks. By a slight abuse of notation, we shall refer to this poset as $P = \big(3, \{4, 14\}, 7, 1, 6, \{9, 12\}, 11, \{2, 8, 13\}, 10, 5\big)$ instead of the usual ordered set of sets.

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 in this paper.

We also consider in this paper 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 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 time polynomial in n, the cardinality of V, and in m, the cardinality of Υ.

Theorem 1. *The decision problem* Generating Poset *can be solved with an $O(mn^2)$-time algorithm that takes a set Υ of m linear orders on n elements as input and finds the*

ALGORITHM: Generating Poset

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 $\mathcal{L}(V) = \Upsilon$, if one exists.

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

Figure 4: Polynomial-time algorithm to solve Generating Poset

poset P that generates Υ, if it exists.

Proof. The correctness and time-complexity of the Generating Poset algorithm in Figure 4 proves the theorem. In this algorithm, we build the binary relation $<_P$ by computing $\cap_{L\in\Upsilon} L$, the smallest partial order that contains all relations common to all the given linear orders. This can be done in $O(mn^2)$ since each of the m linear orders has $O(n^2)$ 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 [15] that generates the linear extensions of P in $O(n)$ amortized time per linear order. Therefore the Generating Poset algorithm in Figure 4 is correct, and runs in $O(mn^2)$ 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 there exists a linear order $L' \in \mathcal{L}(P)$ such that $v \prec_{L'} u$.*

Proof. If $u \parallel_P v$, then there exists at least one topological sort L of P in which u appears immediately to the left of v, due to the alternate choices available to the depth first search used in constructing a topological sort [5]. By the same argument, there must exist a topological sort L' of P in which v appears immediately to the left of u. This proves one direction of the lemma.

For the other direction, suppose 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. If $u \prec_P v$, take any linear extension $L \in \mathcal{L}(P)$. If there exists an element w such that $u <_L w <_L v$, then either $w \parallel_P u$ or $w \parallel_P v$, for otherwise, $u \not\prec_P v$. We can move u so that all elements w satisfying the relation $w \parallel_P u$ will be on its left, and we can move v so that all w incomparable with v will be on its right, to produce a topological sort L_T. Verify that L_T still satisfies all the elements of $<_P$ (since we have just rearranged incomparable elements) and therefore is a linear extension of P. Therefore, there is a linear extension $L \in \mathcal{L}(P)$ for which $u \prec_L v$. Moreover, since $u <_p v$, then for all $L \in \mathcal{L}(P)$, it follows that $u <_L p$ and therefore there is no linear extension $L' \in \mathcal{L}(P)$ for which $v \prec_{L'} u$. This proves one direction of the lemma.

We now prove the other direction. If there exists $L \in \mathcal{L}(P)$ such that $u \prec_L v$, then either $u <_P v$ or $u \parallel_P v$. But since there is no $L' \in \mathcal{L}(P)$ such that $v \prec_{L'} u$, then, by

Lemma 2, u and v cannot be comparable in P. Hence, $u <_P v$. Furthermore, $u \prec_P v$ because if there is a w such that $u <_P w <_P v$, then for all $L \in \mathcal{L}(P)$, it follows that $u <_L w <_L v$, and we therefore cannot have $u \prec_L v$. This proves the other direction of the lemma. □

With Lemma 3, the cover relations of a poset can be computed from the cover relations of all of its linear extensions. Such cover relations in the linear extensions are the adjacent entries when encoded in n-tuple form.

Theorem 4. *The* Generating Poset *problem can be solved with an $O(mn + n^3)$-time algorithm that takes a set of m linear orders Υ over a vertex set with n elements as input and finds the poset P that generates Υ, if it exists.*

Proof. The algorithm in Figure 5 can be used to prove the theorem. Lemma 3, together with the uniqueness of the transitivity 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 $O(n^2)$. Steps 6-8 require $O(mn)$-time since there are $O(n)$ orders in each of the $O(m)$ linear orders. Extracting the cover relations in Steps 11-16 requires $O(n^2)$ time. Computing the transitive closure from the cover relations can be done in $O(n^3)$-time using Floyd-Warshall algorithm. Finally, verifying the generated linear extensions of the computed poset requires $O(mn)$ time, as discussed in the proof for the previous algorithm. Therefore, the entire algorithm has a running time of $O(mn+n^3)$. The theorem follows. □

It should be noted that the number m of linear extensions of a given poset on n elements is almost always bigger than $O(n)$, and in the worst case, $m = O(n!)$. The second algorithm is better than the first one in two significant areas. First, on most instances, it has a lower time complexity. Second, when only the set of cover relations is required, the second algorithm immediately gives the desired result, whereas, the first

ALGORITHM: Generating Poset Using Cover Relations

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 $\mathcal{L}(V) = \Upsilon$, if one exists.

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

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

requires an additional step to find the transitive reduction of the binary relation in the generating 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 can be used with time complexity better than that of any of the two previously discussed algorithms. 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 solved in polynomial time.

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

Theorem 5. *Given a set of m linear extensions on an n-element vertex set, there is an $O(mn + n^2)$-time algorithm to solve* $\text{GENPOSET}_{\text{TREE}}$.

Proof. The algorithm that we present for this problem is the same as the last algorithm, except for the part that constructs 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 $O(n^2)$ overall. Thus the entire algorithm runs in $O(mn + n^2)$-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:

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

Thus, for a linear order, the depth of a vertex is simply its position in the permutation representation of the linear order. The depth of any vertex in a poset can be determined from the linear extensions of the poset as follows.

Lemma 6. *Let $P = (V, <_P)$ be any poset. For any element $v \in V$,*

$$\text{depth}(v; P) = \min\big\{\text{depth}(v; L) \mid L \in \mathcal{L}(P)\big\}$$

Proof. Let $h = \text{depth}(v; P)$ and let $L' \in \mathcal{L}(P)$ be a linear extension of P such that

$$\text{depth}(v; L') = \min\big\{\text{depth}(v; L) \mid L \in \mathcal{L}(P)\big\}.$$

The $h-1$ elements $u \in V$ for which $u <_P v$ must also precede v in L'. It follows, therefore, that $h \leq \text{depth}(v; L')$. Now suppose $h < \text{depth}(v; L')$, and let $j = \text{depth}(v; L') - h$. Then there are exactly j elements $w \in V$ such that $w <_{L'} v$ and $w \not<_P v$ (and hence $w \not<_P u$ for each of the $h-1$ elements mentioned earlier). This also implies that we can move these j elements immediately after v, while keeping their relative orders, to produce a linear order L'' that is also a linear extension of P. This will, however, imply that $\text{depth}(v; L'') < \text{depth}(v; L')$, a contradiction. Therefore, $h = \text{depth}(v; L')$, and the lemma follows. □

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

Lemma 7. *Let $P = (V, <_P)$ be a leveled poset with vertex partition $V_1, V_2, \ldots, V_k$ such that $u <_P v$ if and only if $u \in V_i$, $v \in V_j$ and $i < j$. The depths of vertices belonging to the same partition, in all linear extensions of P is fixed within an interval. That is, there exist integers k_1 and k_2 such that for all $v \in V_i$, and for all $L \in \mathcal{L}(P)$, the depth of v is bounded by $k_1 \leq \text{depth}(v; L) \leq k_2$. In particular, k_1 and k_2 are given by*

$$\begin{aligned} k_1 &= 1 + \sum_{h<i} \big|V_h\big| \\ k_2 &= \sum_{h \leq i} \big|V_h\big| \quad = \quad k_1 + |V_i| - 1 \end{aligned}$$

Moreover, the number of linear extensions L of P for which $\text{depth}(v; L) = k$ for any particular $k \in \{k_1, \ldots, k_2\}$ is given by $\big|L(P)\big| \Big/ \big|V_i\big|$.

Proof. Let V_h^* be the union of all partitions before V_i, and V_j^* be the union of all partitions after V_i so that $u <_P v <_P w$ for all $u \in V_h^*$, $v \in V_i$ and $w \in V_j^*$. Clearly, $V_h^* \cup V_i \cup V_j^* = V$. Any topological sort, therefore, of the Hasse diagram $H(P)$ will have all the vertices in V_h^* before all the vertices in V_i and all the vertices in V_j^* after all the vertices in V_i. Thus for all $u \in V_h^*$, $\text{depth}(u; L) < k_1$ for every $L \in \mathcal{L}(P)$, and for all $w \in V_j^*$, $\text{depth}(w; L) > k_2$ for every $L \in \mathcal{L}(P)$. This proves the first part of the lemma.

For the other part, observe that each of the $|V_i|$ vertices of the partition are equal except in the labelling. Thus, we are sure that the depth of these vertices are equally distributed among the indicated positions. There are $|V_i|$ such positions, and $|\mathcal{L}(P)|$ total linear extensions. Therefore, a simple formula for counting number of linear extensions L of P for which $\text{depth}(v; L) = k$ for any particular $k \in \{k_1, \ldots, k_2\}$ is given by $\big|L(P)\big| \Big/ \big|V_i\big|$. □

Theorem 8. *Let* LEVELED *be the predicate defining the class of leveled posets. Given a set of m linear extensions on an n-element vertex set, there is an $O(mn + n^2)$-time algorithm to solve* $\text{GENPOSET}_{\text{LEVELED}}$.

Proof. The algorithm in Figure 6 solves the given problem. The correctness is established by Lemma 7. For the running time of the algorithm, setting up the Min and Max arrays is done in $O(n)$ while updating them is done in $O(mn)$. Producing the sorted array A

can be done in $O(n \log n)$-time using mergesort. Checking if Lemma 7 is satisfied in lines 13-17 is done in $O(n)$, while building the partial order is in $O(n^2)$. For the last part, checking if 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 7 check. Thus, the overall running time of the algorithm is $O(mn + n^2)$. The theorem follows. □

5 Relaxed Generation

In some instances, it is desirable to relax the generating requirement for the poset. For example, consider as input a recorded set of neuron firing sequences, encoded as permutations. Even though there is a single neural network that generates each of the sequences, it is unlikely that all occuring sequences are recorded because of experimental constraints. In this case, finding a single poset that generates the observed set of sequences will probably fail. Thus, one way to relax the generating requirement is to allow a poset to generate linear extensions outside those in the input set. Of course, the empty poset will always be able to satisfy this, but 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. This is formalized below.

Poset Extended-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.

Theorem 9. *The* Poset Extended-Cover *problem is solvable in polynomial time.*

Proof. We claim that the solution to the Poset Extended-Cover problem consists of getting the intersection of the order relations in the set of linear orders. To see why this is so, let $<_P = \bigcap_{L \in \Upsilon} <_L$. It immediately follows that if $P = (V, <_P)$, then $P \sqsubseteq L$ for all $L \in \Upsilon$. What remains to be shown is that there is no poset P' that covers all of the linear extensions in Υ but covers fewer linear extensions than P. But for P' to generate

ALGORITHM: Generating Leveled Poset

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 $\mathcal{L}(V) = \Upsilon$, if one exists.

```
1   ▷ Initialize the arrays to record minimum and maximum depth for each vertex
2   for v ← 1 to n
3       Min_v ← n
4       Max_v ← 1
5   ▷ Derive the min and max depths of every vertex in every linear extension
6   for i ← 1 to m
7       for h ← 1 to n
8           Let v be the element in linear extension L_i having depth h
9           Min_v ← min(h, Min_v)
10          Max_v ← max(h, Max_v)
11  ▷ Check if Lemma 7 is satisfied
12  Set A to be the array of vertices v ∈ V sorted by Min_v value
13  for j ← 1 to n − 1
14      u ← A_j
15      v ← A_{j+1}
16      if not ((Min_u = Min_v and Max_u = Min_v) or Max_u < Max_v)
17          return failure
18  ▷ Build the partial order, if passed the Lemma 7 check
19  <_P ← ∅
20  for u ← 1 to n − 1
21      for v ← u + 1 to n
22          if Min_u < Min_v
23              <_P ← <_P ∪ (u, v)
24          else if Min_u > Min_v
25              <_P ← <_P ∪ (v, u)
26  if |Υ| = |L(P)|
27      return P
28  else
29      return failure
```


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

a super set of Υ, $<'_P \subseteq \bigcap_{L\in\Upsilon} <_L$. Clearly, removing an orderd pair from $<'_P$ will only increase the number of linear extensions. Therefore, P is the desired poset. To construct P in polynomial time, we use the first or second algorithm for the GENERATING POSET problem, but skip the verification step. □

Another possible relaxation that can be applied to the generating requirement is to allow a poset to not completely generate the input set of linear orders. Let Υ be a set of linear orders on V. 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. Moreover, $P = (V, <_P)$ is said to be *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 partial cover posets 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 discussed 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 kite posets for Υ can be generated in $O(mn^4)$ time.*

Proof. Suppose that P is a kite poset that is a partial cover of Υ and let $V_h \subset V$ be its hammock. Then there exists unique elements $u, v \in V$ such that for all $win V_h$, $u \prec_P w \prec_P v$. Let $i = \text{depth}(u; P)$ and $j = \text{depth}(v; P)$. It follows that $1 \leq i < j \leq n$ and $j - i \geq 3$. We search for the elements u and v by considering the $O(n^2)$ possible i, j pairs. For each 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 poset that has linear extension L_r if and only if there are $(j-i-1)!$ 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 $O(mn^3)$ time.

To complete the time-complexity analysis, we count the number of possible kite posets that can be returned by this algorithm. For kite(2)-posets, a linear order $L \in \Upsilon$ can be a linear extension of at most $O(n)$ kite(2)-posets since there are only $O(n)$ adjacent positions in a linear order. Thus, at most $O(mn)$ kite(2)-posets can be found for Υ. For each of these, constructing the binary relation requires $O(n^2)$. Thus all partial cover kite(2)-posets can be constructed in $O(mn^3)$ (after being detected). The same is true for all kite(k)-posets for $2 < k < n-2$. Thus for all possible kites, at most $O(mn^4)$-time for construction is required. The running time of this algorithm is therefore $O(mn^3)$ for detection and $O(mn^4)$ for construction. The theorem follows. □

We now proceed to an extension of this result to a class of hammock posets.

Theorem 11. *Given a set Υ of m linear extensions on an n-element vertex set V, determining all partial cover hammock posets having exactly 2 hammocks can be done in $O(mn^5)$.*

Proof. The algorithm we use is the same as the one for finding partial cover kite posets, except, rather than considering only (i,j) pairs, we now consider two pairs (i_1,j_1) and (i_2,j_2), with $1 \le i_1 < j_1 \le i_2 < j_2 \le n$, and restrict the linear extensions based on these two pairs. There will be $O(n^4)$ pairs (of pairs) to consider instead of $O(n^2)$. For each restriction, we sort in $O(mn)$-time and check if there are $(j_1-i_1-1)! \cdot (j_2-i_2-1)!$ linear orders in Υ having the same restriction. Detecting all partial cover hammock posets (having exactly 2 hammocks) therefore requires $O(mn^5)$ time. It is easy to see that there will be less such hammocks than kite posets for the same Υ. Therefore the construction time will not require more than the $O(mn^4)$-time needed for constructing all partial cover kites. The theorem thus follows. □

The last theorem can be extended to hammock posets with exactly k hammocks. Using a similar reasoning, we can prove the following theorem.

Theorem 12. *Given a set Υ of m linear extensions on an n-element vertex set V, determining all partial cover hammock posets having exactly k hammocks can be done in $O(mn^{2k+1})$.*

6 Using Trees and DAGs

The previous section touched on some algorithms for mining some classes of partial cover posets. The general strategy involves sorting the set of linear extensions over different possible restrictions. Such algorithms however are expensive when the input is stored in a database since they require possibly many access to secondary storage. In this section we consider (potentially) more compact data structures for representing the linear orders, which can be used not only to reduce storage space but may improve some algorithms for finding partial cover posets.

Given a set $\Upsilon = \{L_1, L_2, \ldots, L_m\}$ over the base set $V = \{1, 2, \ldots, n\}$, we define a *linear order tree* of Υ as a rooted directed tree $\mathcal{T}(\Upsilon)$ satisfying the following conditions:

1. There are m different paths from the root to the leaves, each corresponding to a linear order (in n-tuple form) from Υ. Each of the vertices is labeled by the corresponding element in the linear order.

2. No two distinct subpaths from the root to any node in the tree have the same vertex sequence. Equivalently, each distinct prefix of the linear orders (in n-tuple form) corresponds to a single subpath in the tree.

3. Each edge (u, v) is assigned a weight $weight(u, v)$ equal to the number of distinct paths (from root to leaves) that contain it. Equivalently, this weight is the number of leaves in the maximal subtree whose root is the node v.

Figure 7 shows an example of a linear order tree. The space complexity of any linear order tree is discussed in the next theorem.

Theorem 13. *Let $\mathcal{T}(\Upsilon)$ be a linear order tree for a set Υ of m linear orders on an n-element base set. Then the linear order tree requires $O(mn)$-space in the worst case, and $\Omega(m+n)$-space in the best case.*

Proof. In the worst case, none of the linear orders have common prefix. In such a case, the tree consists of m paths of length n that intersect only at the root node. The tree, therefore, requires $O(mn)$-space in such an instance.

Consider now the case when $m = k!$ for some k. In the best possible scenario, all of these linear orders share a common prefix, and the common prefix is as long as it is possible for a given set of linear orders. We count the number of nodes after the subpath corresponding to the common prefix. When $m = 2!$, there are 4 nodes in the best case (see Figure 8). When $m = 3!$, the best case has 15 nodes, and is constructed by adding a (parent) node to the 4 nodes earlier, and then making 3 copies of it (which are then connected to the subpath representing the common prefix). The number a_k of nodes occuring after the subpath of the common prefix in the best possible scenario for $k!$ linear orders is given recursively as follows:

$$a_k = k(a_{k-1} + 1) = ka_{k-1} + k$$

with the base case $a_k = 1$. We can expand this equation to derive the desired result.

$$\begin{aligned}
a_k &= (ka_{k-1}) + (k) \\
&= \big(k(k-1)(a_{k-2})\big) + \big(k(k-1)\big) + (k) \\
&= \big(k(k-1)(k-2)(a_{k-3})\big) + \big(k(k-1)(k-2)\big) + \big(k(k-1)\big) + (k) \\
&\vdots \\
&= \big(k(k-1)(k-2)\cdots 1\big) + \big(k(k-1)(k-2)\cdots 1\big) \\
&\quad + \big(k(k-1)(k-2)\cdots 2\big) + \cdots + \big(k(k-1)\big) + (k) \\
&= k! + k! + \big(k(k-1)(k-2)\cdots 2\big) + \cdots + \big(k(k-1)\big) + (k) \\
&< k! + k! + k!
\end{aligned}$$

Thus, for $m = k!$, it is possible to have a (best) case where less than $3m$ nodes are added after the subpath corresponding to the longest common prefix. Because the longest common prefix can be $O(n)$ long, then the best case has $\Omega(m+n)$-space complexity. The theorem follows. □

The linear order tree for an input set Υ can be constructed in $O(mn^2)$-time. This is because processing each of the mn elements requires checking for a worst-case n children of a particular node in the tree. Actually, there is no need to check n children of a particular node, if for each node an array of (fixed) n pointers is used. However, if such is the data structure used, then the size of the tree will be increased by a factor of n in the worst case.

Let $L = (v_1, v_2, \ldots, v_n)$ be a linear order on V. We define the *path* $p = (r, v_1, v_2, \ldots, v_n)$ to be the path associated with L in the linear order tree $\mathcal{D}(\Upsilon)$, from the root r to the leaf v_n. For each vertex v, let $branch_p(v)$ be the number of children the node v from the path p has in the tree $\mathcal{D}(\Upsilon)$.

A brute force way of extracting partial cover kites from the tree $\mathcal{T}$ is given in Figure 9. This algorithm processes each linear order and checks whether it is possible to have a kite at any set of contiguous positions. The linear order tree in this case is simply used to

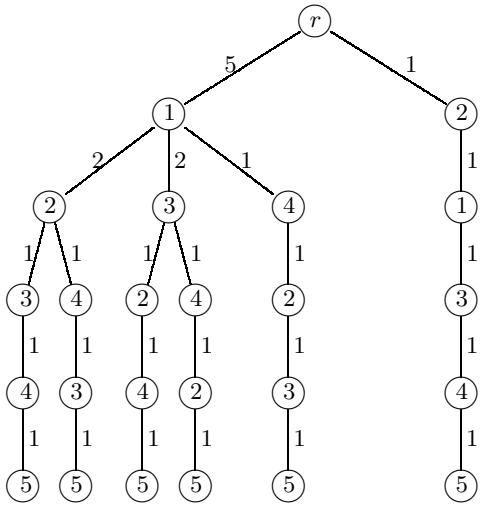


Figure 7: A linear order tree for $\Upsilon = \{12345, 12435, 13245, 13425, 14235, 21345\}$

search for specific linear orders in Υ quickly. The overall running time of this algorithm, however, is $O(m^2n^4)$, mainly due to the loops in Steps 3-8. Step 3 requires $O(mn)$ since there are m paths each of length $O(n)$. Steps 4-5 clearly run in $O(n^2)$, then, finally, Step 7 may require $O(mn)$, in the worst case when all the paths in $\mathcal{T}$ have to be traversed.

We can improve the brute force algorithm in Figure 9 by doing the checking of the linear extensions of $P = \big(v_1, \ldots v_{i-1}, \{v_i, v_{i+1}, \ldots, v_j\}, v_{j+1}, \ldots, v_n\big)$ only once instead of the current $(j - i + 1)!$ times. This can be done in several ways. One such way is to check this only on a path where the hammock $\{v_i, v_{i+1}, \ldots, v_j\}$ is arranged in increasing order. Another is to put flags on the nodes in the tree $\mathcal{T}$ in order to indicate the size of the hammocks (in partial cover kites) that begin with that node. In any case, the brute force algorithm leaves a lot of room for improvement. One such improvement is to take advantage of the edges (and their weights) in the linear order tree in order to prune the search space further. The lemma that follows is the basis for this strategy.

Lemma 14. *Let* $P = \big(v_1, v_2, \ldots, v_i, \{v_{i+1}, v_{i+2}, \ldots, v_{i+k}\}, v_{i+k+1}, \ldots, v_n\big)$ *be a kite(k)-poset.* P *is a partial cover of* Υ *only if the following conditions are satisfied:*

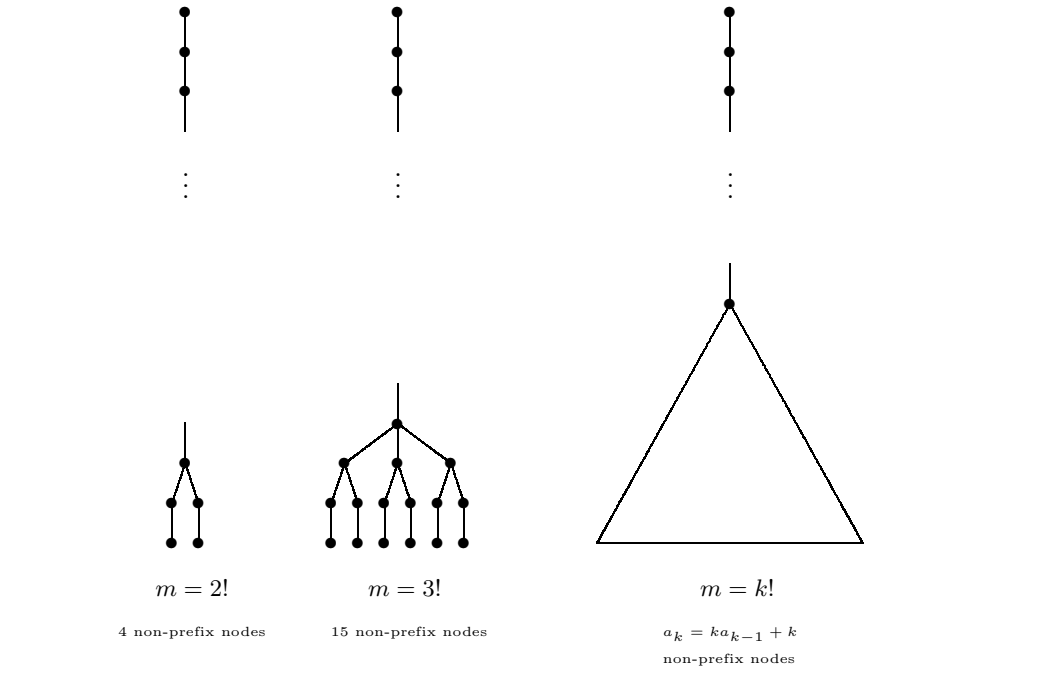


Figure 8: Compact linear order trees requiring $O(m + n)$ space

ALGORITHM: Brute Force Mining of Partial Cover Kite Posets

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

OUTPUT: A set $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ of kite posets on V such that for each $P \in \mathcal{P}$, $\mathcal{L}(P) \subseteq \Upsilon$.

```
1  Let T be the keyword tree for the set Υ
2  P ← ∅
3  for each path p = (r, v_1, v_2, ..., v_n) in the tree T
4      for i ← 2 to n − 2
5          for j ← i + 1 to n − 1
6              Poset P ← (v_1, ... v_{i−1}, {v_i, v_{i+1}, ..., v_j}, v_{j+1}, ..., v_n)
7              if every linear extension L ∈ L(P) has a corresponding path in T
8                  P ← P ∪ P
9  return P
```

Figure 9: Mining kite posets from a linear order tree using brute force algorithm.

1. *the path* $p = (r, v_1, v_2, \ldots, v_i, v_{i+1}, \ldots, v_{i+k}, v_{i+k+1}, \ldots, v_n)$ *exists in the tree* $\mathcal{T}(\Upsilon)$

2. $weight_p(v_{i-1}, v_i) \geq k!$

3. $branch_p(v_i) \geq k$

Proof. Clearly, the first item has to be satisfied, otherwise the linear extension

$$L = (v_1, v_2, \ldots, v_i, v_{i+1}, \ldots, v_{i+k}, v_{i+k+1}, \ldots, v_n)$$

of the poset P is not in Υ, and therefore P is not a partial cover of Υ. The second item follows from the number of linear extension of kite(k)-poset, i.e., $k!$, each of which has the common prefix $(v_1, v_2, \ldots, v_i)$. The final item is due to the fact that any of the k elements in the hammock can follow the vertex v_i. The lemma follows. □

Other pruning strategies can be implemented. For instance, we can also check if the vertex v_i has an edge of weight at least $(k-1)!$ to at least k of its children, and do the checking recursively (i.e., to the descendants of v_1). Another way to prune the search space is to improve the data structure itself.

Define a *linear order DAG* $\mathcal{D}$ of a set Υ of linear orders as a directed acyclic graph derived from combining common suffixes in the linear order tree of Υ into single paths. See Figure 10 for a sample linear order DAG that corresponds to the tree in Figure 7. A bottom-up strategy for combining the nodes defining common suffixes will generate the desired DAG from a given tree. Verify that such construction requires fewer steps than constructing the tree from Υ. Thus, the linear order DAG can still be constructed in $O(mn^2)$ time from a given set Υ of linear orders (by first producing the tree $\mathcal{T}(\Upsilon)$).

Clearly, for a given set Υ of linear orders, the DAG $\mathcal{D}(\Upsilon)$ is more compact than the tree $\mathcal{T}(\Upsilon)$, unless there exists no common suffix among the set of linear orders. In fact, we can think of the DAG $\mathcal{D}(\Upsilon)$ as a loss-less compression of both Υ and $\mathcal{T}(\Upsilon)$ since both of these can be completely reconstructed from the DAG. However, on the worst case, it still requires $O(mn)$ space, while in the best case, only $O(m+n)$ is required, and therefore, the same as those of the linear order tree. What the DAG $\mathcal{D}(\Upsilon)$ offers, though, is a more

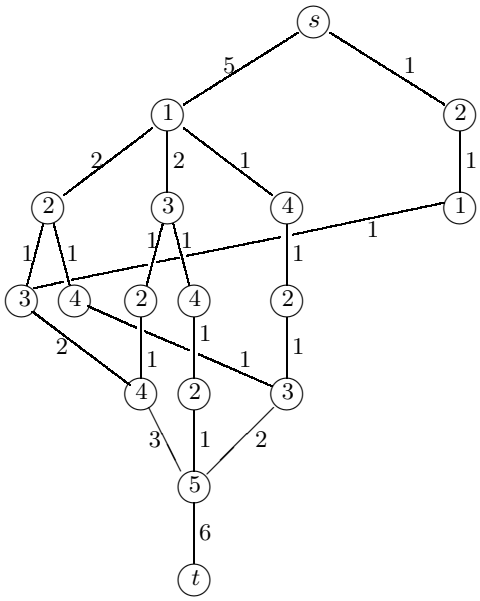


Figure 10: A linear order DAG for $\Upsilon = \{12345, 12435, 13245, 13425, 14235, 21345\}$

efficient way to mine certain classes of posets than can be done using the linear order tree. The next lemma can be used to find partial cover kite posets from a linear order DAG.

Lemma 15. *Let* $P = \big(v_1, v_2, \ldots, v_i, \{v_{i+1}, v_{i+2}, \ldots, v_{i+k}\}, v_{i+k+1}, \ldots v_n\big)$ *be a kite(k)-poset.* P *is a partial cover of the set* Υ *of linear orders if and only if the following conditions hold.*

1. *The path* $p = (s, v_1, v_2, \ldots, v_i, v_{i+1}, v_{i+2}, \ldots, v_{i+k}, v_{i+k+1}, t)$ *exists in the DAG* $\mathcal{D}(\Upsilon)$.

2. *There are* $k!$ *subpaths from the nodes* v_i *and* v_{i+k+1} *(of the path* p*).*

Proof. If P is a partial cover of Υ, then since $L = (v_1, v_2, \ldots, v_i, v_{i+1}, v_{i+2}, \ldots, v_{i+k}, v_{i+k+1})$ is one of its linear extensions and is therefore an element of Υ, the path p defined in the lemma must be a path in the DAG $\mathcal{D}(\Upsilon)$. Moreover, its $k!$ linear extensions must also be in Υ, and therefore there must be $k!$ subpath associated with the hammock $(v_i, \{v_{i+1}, v_{i+2}, \ldots, v_{i+k}\}, v_{i+k+1})$. This proves one direction of the lemma. For the other direction, since each path consists of a permutation of the n vertices in the base set of the

poset P, then if there are exactly $k!$ subpaths from the nodes v_i and v_{i+k+1} (of the path p), then we are certain that (since the prefix and suffixes are common), it must follow that the elements in these $k!$ subpaths are the same, but in different order. Consequently all of the $k!$ linear extensions of P have their corresponding paths in the DAG $\mathcal{D}(\Upsilon)$. Thus, $\mathcal{L}(P) \subseteq \Upsilon$ and therefore P is a partial cover of Υ, proving the other direction. The lemma follows.

□

Figure 11 shows an algorithm that makes use of Lemma 15. In particular, the second condition in Step 8 is sufficient to prove that the kite(2)-poset in Step 9 is a partial cover of the input set. Steps 6-7 are also added to prune the search space. The basis for this pruning technique is Lemma 14. By performing the tests on all paths in the DAG $\mathcal{D}(\Upsilon)$, we are certain that the set of all partial cover kite(2)-posets is returned by the algorithm. Thus, only the time complexity of the algorithm in Figure 11 is needed to prove the next theorem.

Theorem 16. *Given an input set Υ of m linear orders on an n-element base set, the set of all partial cover kite(2)-posets can be determined in $O(mn^3)$-time.*

Proof. Figure 11 shows an algorithm for mining the partial cover kite(2) posets. The correctness is established in the preceding paragraph. For the time complexity, observe the following crucial steps in the algorithm.

(a) Construction of the DAG in Step 1 can be done in $O(mn^2)$-time as discussed previously.

(b) The loop in Step 3 iterates $O(m)$ times, exactly the number of linear orders in Υ.

(c) Step 5 clearly iterates $O(n)$ times.

(d) For the second condition in Step 8, we can start the search with v_i and need only to check in $O(n)$-time if v_{i+2} is a child of v_i, then proceed to check (again in $O(n)$-time) v_{i+1} and finally v_{1+3}. Checking the second condition, therefore, requires $O(n)$-time.

ALGORITHM: Mining Partial Cover Kite(2)-Posets from Linear Order DAGS

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

OUTPUT: The set $\mathcal{P} = \{P \mid P \text{ is a kite(2)-poset and } \mathcal{L}(P) \subseteq \Upsilon\}$.

```
1   Let 𝒟 be the linear order DAG for the set Υ
2   𝒫 ← ∅
3   for each path p = (s, v_1, v_2, ..., v_n, t) in the DAG 𝒟
4       v_0 ← s
5       for i ← 1 to n − 3   ▷ v_i is the candidate fork node (i.e., vertex before hammock)
6           if weight(v_{i−1}, v_i) < 2!   ▷ impossible to form kite(2)-poset
7               i ← n   ▷ break the inner for-loop
8           else if v_{i+1} < v_{i+2}   ▷ ensure no duplicate checking is done
                    and subpath (s, v_1, ..., v_i, v_{i+2}, v_{i+1}, v_{i+3}) ends in same node v_{i+3} of p
9               𝒫 ← 𝒫 ∪ (v_1, ... v_i, {v_{i+1}, v_{i+2}}, v_{i+3}, ..., v_n)
10  return 𝒫
```

Figure 11: Mining kite(2)-posets from a linear order DAG

(e) Performing the set union in Step 9 requires no additional scanning of $\mathcal{P}$ since we are sure that, because of the first condition in Step 8, the poset to be added is not yet in $\mathcal{P}$. Thus, this can Step can be done in $O(n)$-time, proportional to the length of the tuple.

The algorithm in Figure 11, therefore, runs in $O(mn^3)$-time. Howeve, this is just done to collect the kite(2)-posets in tuple form. Constructing the actual partial order from the permutations requires additional processing. Similar to the discussion of the proof of Theorem 10, there will be at most $O(mn)$ permutations representing kite(2)-posets that can be detected. For each of these permutations, there are $O(n^2)$ elements in the binary relation of the corresponding poset. Therefore, constructing the partial cover kite(2)-posets detected by the algorithm in Figure 11 runs in $O(mn^3)$-time. The theorem follows.

□

7 Inductive Construction of Partial Cover Posets

This section discusses techniques for mining partial cover posets based on previously determined partial covers. For instance, we can build partial cover kite(k)-posets if we know all the partial cover kite($k-1$)-posets. This is presented formally as a lemma.

Lemma 17. *Let* $P = \big(v_1, \ldots, v_i, \{v_{i+1}, \ldots, v_{i+k}\}, v_{i+k+1}, \ldots, v_n\big)$ *be a kite(k)-poset. Then* P *is a partial cover of an input set* Υ *if and only if for each* $j \in \{1, 2, \ldots, k\}$*, the poset* $P_x(j) = \big(v_1, \ldots, v_i, v_{i+j}, \{v_{i+1}, \ldots, v_{i+j-1}, v_{i+j+1}, \ldots, v_{i+k}\}, v_{i+k+1}, \ldots, v_n\big)$ *is a partial cover poset of* Υ*.*

Similarly, P *is a partial cover of* Υ *if and only if for each* $j \in \{1, 2, \ldots, k\}$*, the poset* $P_y(j) = \big(v_1, \ldots, v_i, \{v_{i+1}, \ldots, v_{i+j-1}, v_{i+j+1}, \ldots, v_{i+k}\}, v_{i+j}, v_{i+k+1}, \ldots, v_n\big)$ *is a partial cover poset of* Υ*.*

Proof. The proof is clearly established by the fact that

$$\begin{aligned}\mathcal{L}(P) &= \bigcup_{1\leq j\leq k} \mathcal{L}(P_x(j)) \\ &= \bigcup_{1\leq j\leq k} \mathcal{L}(P_y(j)).\end{aligned}$$

Each of these three expressions consists of the set of linear orders having all possible permutations of $\{v_{i+1}, \ldots, v_{i+k}\}$ in between the same prefix $(v_1, v_2, \ldots, v_i)$ and suffix $(v_{i+k+1}, v_{i+k+2}, \ldots, v_n)$. □

The lemma offers two options for building partial cover kites having larger hammocks inductively. Either start with the left end of the hammock or start with the other end. Both of these ideas can be taken advantage of in mining for partial cover kites using the linear order DAG. Observe also that for the base case (i.e., mining kite posets for $k = 2$) has been discussed in two separate parts (using two different approaches) of this paper.

The same inductive approach can be extended to leveled-posets. The proof is also similar to that of the preceding lemma, that is by verifying that the set of linear extensions generated is exactly the same. Similar also to the previous lemma, the inductive approach can start with the left end of a partition or on the right end, but in this case only one side will be presented, for brevity.

Lemma 18. *Let $P = (V_1, V_2, \ldots, V_k)$ be a leveled poset, and consider any partition V_h, for some $h \in \{1, \ldots, k\}$. Without loss of generality, suppose V_h has $j > 2$ vertices $v_{i+1}, v_{i+2}, \ldots, v_{i+j}$ (for some integer i). Then P is a partial cover leveled poset of a set Υ of linear orders if and only if for all $l \in \{1, 2, \ldots, j\}$,*

$$P_h(l) = (V_1, V_2, \ldots, V_{h-1}, \{v_{i+l}\}\{v_{i+1}, \ldots, v_{i+l-1}, v_{i+l+1}, \ldots, v_{i+j}\}, V_{h+1}, \ldots, V_k)$$

is also a partial cover leveled poset in Υ.

8 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.

Poset 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 cover $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ of Υ such that k is minimum.

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 [7], 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 particular classes of posets and poset covers whose elements come from a particular class.

C-Poset Cover (COVER_C)

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

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

In fact, if we restrict our attention to KITE(2)-posets, the Poset Cover problem is solvable in polynomial time.

Theorem 19. *There is an $O(m^{1.5}n + mn^3)$-time algorithm to solve* $\text{COVER}_{\text{KITE(2)}}$.

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 $O(mn^3)$ time as discussed 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 $n-1$ 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 can find a maximum matching in the graph using the algorithm of Micali and Vazirani [13], which runs in $O(m^{1/2}p) = O(m^{1.5}n)$ 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 $\text{COVER}_{\text{KITE(2)}}$. □

In this section, we also show the NP-completeness of $\text{COVER}_{HAMMOCK(2,2,2)}$ using a reduction similar to what Heath and Nema [7] used. In particular, we reduce from the Cubic Vertex Cover, a known NP-complete problem, which is described below.

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 $K \leq |V|$.

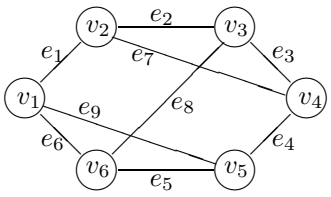
QUESTION: Is there a subset $V' \subset V$ of cardinality K or less such that every edge in E is incident on at least one vertex in V'?

The main idea in the reduction is to represent edges with linear extensions, and vertices with 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 such that if a vertex cover contains a particular vertex, t hen the corresponding poset cover contains the corresponding hammock poset. This is further elucidated in the proof of the following theorem.

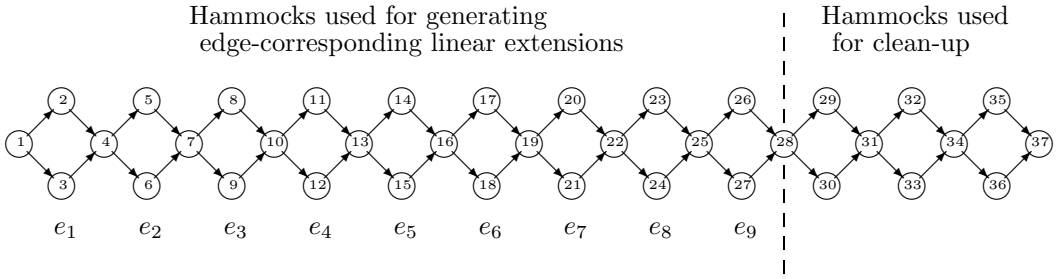
Theorem 20. *The problem* $\text{COVER}_{HAMMOCK(2,2,2)}$ *is NP-complete.*

Proof. It is easy to easy to see that $\text{COVER}_{HAMMOCK(2,2,2)}$ is in the class NP. Given a set of posets $\mathcal{P}$, we check if it satisfies the cardinality requirement and then the covering requirement. For the second requirement, we 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, the $\text{COVER}_{HAMMOCK(2,2,2)}$ problem is in NP. To complete

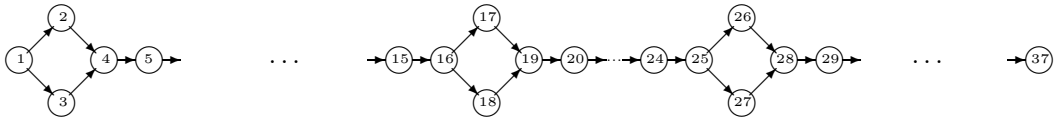
(a) A cubic graph example



(b) Template for constructing the hammock posets



(c) Hasse diagram of the hammock poset P_{v_1} corresponding to vertex v_1 (incident on e_1, e_6, e_9)



(d) Linear extensions that correspond to edges e_1, e_6 and e_9

L_{e_1}=(1,**3,2**,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37)
L_{e_6}=(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,**18,17**,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37)
L_{e_9}=(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,**27,26**,28,29,30,31,32,33,34,35,36,37)

Figure 12: A cubic graph in an instance of Cubic Vertex Cover, and the hammock posets and linear extensions that correspond to the vertices and edges respectively.

the proof of the theorem, we show a polynomial-time redcution from the CUBIC VERTEX COVER problem, a known NP-complete problem, to the $\text{COVER}_{HAMMOCK(2,2,2)}$.

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. From the CUBIC VERTEX COVER instance, define a $\text{COVER}_{HAMMOCK(2,2,2)}$ instance as follows. The base set is $V = \{1, 2, \ldots, 3(n_e + 3) + 1\}$, and the set of linear orders Υ contains the *base linear order* $(1, 2, \ldots, 3(n_e + 3) + 1)$, written in tuple notation. The other elements of Υ will depend on the labeling of the edges in G. In any case, we fix the elements of the tuple at positions $1, 4, 7, \ldots, 3(n_e + 3) + 1$ and let the other elements be possibly interchanged with their adjacent non-fixed elements (see Figure 12). Using this idea, an edge e_i is represented by the linear order derived from the base linear order but whose elements at $3i - 1$ and $3i$ have been interchanged.

Although we do not include the posets in the instance of the $\text{COVER}_{HAMMOCK(2,2,2)}$ problem, we imagine that for every vertex v in the cubic graph G, there corresponds a unique hammock(2,2,2) poset P_v. This hammock must cover the three linear orders corresponding to the edges incident on v. This can be accomplished by making the vertices $3i-1$ and $3i$ be elements of a hammock in P_v if the edge e_i is incident on v (see Figure 12(c)). We include all of the linear extensions of P_v in the set Υ. With such construction, the linear order L_{e_i} corresponding to the edge e_i can only be covered by hammocks that correspond to the vertices that are incident on e_i. The only problem is that the hammock poset P_v that covers a linear order L_e also generates linear orders that do not represent any edge in the cubic graph. These are the base linear order and four other linear orders (which we shall *extra* linear orders) that result from interchanging the adjacent entries in at least two of the three hammocks. The base linear order is covered by all hammocks that correspond to vertices. For the remaining four linear orders, a clean-up operation is performed.

For clean-up, we introduce enough linear orders and make some partial cover hammock posets (not representing a vertex in the cubic graph) essential elements of any poset cover.

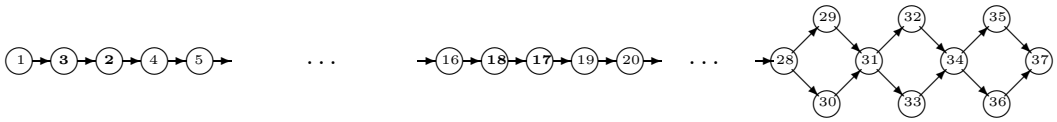
This way, we can ensure that the extra linear orders will always be covered. Let L be one such (extra) linear order. By our construction, the pair of interchanged elements can not occur beyond the pair of elements $(3n_e - 1, 3n_e)$. Let P be a hammock(2,2,2) poset whose linear extensions include L, and whose hammocks occur at the last 3 possible hammock positions i.e., on the pairs $(3n_e+2, 3n_e+3)$, $(3n_e+5, 3n_e+6)$ and $(3n_e+8, 3n_e+9)$, as shown in the template in Figure 12(b). Thus the first $3n_v + 1$ elements will form a chain, and this chain is a chain copy of the first $3n_v + 1$ elements of L. P generates exactly $2!2!2! = 8$ linear orders, one of which is L. See Figure 13 for an illustration of clean-up for one extra linear extension of the hammock poset of Figure 12(c). We include the seven other linear extensions of P in the input intance Υ in our reduction. Furthermore, we include P in the poset cover since the seven additinonal linear extensions that we introduce using it cannot be covered by any other hammock(2,2,2) poset in the reduction. After processing each extra linear order (i.e., constructing additional linear orders and pre-selecting the hammock(2,2,2) poset that cover it), exactly $4n_v$ hammock(2,2,2) posets (used for clean up) will have been pre-selected for the poset cover, and $32n_v$ linear orders in the set Υ will have been covered. Furthermore, the only linear orders not covered will be those that correspond to edges in the cubic graph, together with the base linear order.

Verify that every edge-corresponding linear order can only be covered by the two hammock posets corresponding to the vertices that the edge is incident on. The base linear order, on the other hand, is covered by every vertex-corresponding poset. With the following construction, if there exists K vertices in the cubic graph that forms a vertex cover, then $K' = K + 4n_v$ posets can cover the set Υ of linear orders in the corresponding instance of the $\text{COVER}_{HAMMOCK(2,2,2)}$ problem. This is done by selecting the K hammock posets that correspond to the elements of the vertex cover, together with the $4n_v$ hammock posets that are used for the clean-up. Similarly, if there exists K' posets that can cover Υ, then the $K = K' - 4n_v$ vertices that correspond to the vertex-corresponding hammock posets in the poset cover of Υ also forms a vertex cover in the cubic graph instance of the Cubic Vertex Cover problem. Thus our reduction is correct.

(a) A sample extra linear order L

L=(1,**3,2**,4,5,6,7,8,9,10,11,12,13,14,15,16,**18,17**,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37)

(b) A hammock(2,2,2) poset P to cover L



(c) Additional linear orders

L_1=(1,**3,2**,4,5,6,7,8,9,10,11,12,13,14,15,16,**18,17**,19,20,21,22,23,24,25,26,27,28,**30,29**,31,32,33,34,35,36,37)
L_2=(1,**3,2**,4,5,6,7,8,9,10,11,12,13,14,15,16,**18,17**,19,20,21,22,23,24,25,26,27,28,**30,29**,31,**33,32**,34,35,36,37)
L_3=(1,**3,2**,4,5,6,7,8,9,10,11,12,13,14,15,16,**18,17**,19,20,21,22,23,24,25,26,27,28,**30,29**,31,32,33,34,**36,35**,37)
L_4=(1,**3,2**,4,5,6,7,8,9,10,11,12,13,14,15,16,**18,17**,19,20,21,22,23,24,25,26,27,28,**30,29**,31,**33,32**,34,**36,35**,37)
L_5=(1,**3,2**,4,5,6,7,8,9,10,11,12,13,14,15,16,**18,17**,19,20,21,22,23,24,25,26,27,28,29,30,31,**33,32**,34,35,36,37)
L_6=(1,**3,2**,4,5,6,7,8,9,10,11,12,13,14,15,16,**18,17**,19,20,21,22,23,24,25,26,27,28,29,30,31,**33,32**,34,**36,35**,37)
L_7=(1,**3,2**,4,5,6,7,8,9,10,11,12,13,14,15,16,**18,17**,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,**36,35**,37)

Figure 13: Performing a clean-up by handling the extra linear orders.

What needs to be done now is to check that the reduction can be done in polynomial time. Let n_v and $n_e = \frac{3n_v}{2}$ be the number of vertices and edges respectively in the cubic graph, part of the instance of the CUBIC VERTEX COVER problem. The number of elements in the base set, and hence the length of each linear order, will be given by $3n_e+10$. The number of linear extensions that will be created is exactly $7n_v+1+4\cdot 7n_v = 35n_v+1$. Thus, the input set Υ of linear orders can be constructed in time polynomial to the number of vertices and edges in the cubic graph instance. Also, since computing $K' = K + 4n_v$ is done in constant time, then the entire reduction is done in polynomial time. Because the CUBIC VERTEX COVER problem is NP-complete, we can now conclude that the $\text{COVER}_{HAMMOCK(2,2,2)}$ problem is also NP-complete. □

9 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. 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. 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] 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.

[2] 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):193–206, 1996.

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

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

[5] 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.

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

[7] L.S. Heath and A.K. Nema. The Poset Cover Problem. Submitted, 2007.

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

[9] 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.

[10] 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.

[11] 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.

[12] A. Ono and S. Nakano. Constant time generation of linear extensions. *Fundamentals of Computational Theory, Proceedings*, 3623:445–453, 2005.

[13] P.A. Peterson and M.C. Loui. The General Maximum Matching Algorithm of Micali and Vazirani. *Algorithmica*, Vol. 3:pages 511–533, 1988.

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

[15] G. Pruesse and F. Ruskey. Generating Linear Extensions Fast. *SIAM Journal on Computing*, Vol. 23(2):pages 373–386, 1994.

[16] 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.

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

[18] C.H. Wiggins and I. Nemenman. Process Pathway Inference via Time Series Analysis. *Experimental Mechanics*, Vol. 43(3):pages 361–370, Sep 2003.