Discovering Partial Orders in Binary Data

Deepak Rajan
IBM T. J. Watson Research Center
Hawthorne, NY 10532
drajan@us.ibm.com

Philip S. Yu
IBM T. J. Watson Research Center
Hawthorne, NY 10532
psyu@us.ibm.com

Abstract

We approach the problem of discovering interesting orders in data. In many applications, it is more important to find interesting partial orders since there is often no clear ordering between certain sets of elements. Furthermore, a partial order is more robust against partially erroneous data. We present the notion of fundamental partial orders (FPO), and argue that any partial order that satisfies this property is an interesting partial order. To mine such partial orders, we present a two-stage methodology that first finds an interesting total order, and then discovers a partial order satisfying FPO using this total order. To illustrate, we focus on \{0, 1\} data. This is an important problem with many applications, e.g., in paleontology, where we chronologically order fossil sites by minimizing Lazarus counts. We present the experimental results of our method on paleontological data, and show that it outperforms existing approaches. The techniques developed here are general and can be abstracted for mining partial orders in any setting.

1. Introduction

The vast field of data mining, also known as knowledge-discovery in databases, primarily deals with searching for patterns in large volumes of data for a variety of purposes. We consider a particular aspect, one that relates to discovering interesting orders in data. The problem of finding ordering relationships in data is an extremely important problem, and with a large variety of applications. At the same time, the problem often requires a different solution approach in each domain depending on the criterion for selecting the ordering. Furthermore, in certain applications, there is no clear ordering between certain elements. Thus, what is often required is a partial ordering.

We introduce the notion of “fundamental partial orders”, and emphasize that it can be used for any ordering problem where there is no interesting total ordering among data. Assume that we wish to find the data order that minimizes certain objective \(\chi\). Our general methodology has two stages. First, we find a total order that minimizes objective \(\chi\). Then, using this total order, we discover a minimal partial order that also attains the same minimum value of objective \(\chi\). We argue that our methodology is a fundamentally sound approach for discovering interesting partial orders.

In this paper, we focus on an application from paleontology and describe techniques for discovering interesting partial orders. In this application, we wish to chronologically order fossil sites based on the presence of various genera at these sites; see [2, 8, 9, 17, 6] for some approaches. Since there are often errors in the data, and there is often no clear chronological ordering between certain pairs of sites, we need to find partial orders; see Section 1.2 for a detailed discussion. For this problem, the criterion used for selecting the ordering is minimizing the Lazarus count.

Discovering interesting partial orders in binary data has many applications, even within the vast field of biology. The most widely used strategy in the Human Genome Project for mapping of chromosomes is the Sequence-Tagged-Site mapping strategy [16]. In this technique, a chromosome is broken down into overlapping fragments (clones) for detailed study. Relatively short substrings (probes) are extracted from the chromosome, and used to find interesting partial orders among the clones [1, 5]. These partial orders are then used to map the chromosome in its entirety. In this problem, the criterion used for selecting the ordering reduces to finding a good dependency graph.

Applications also abound in other fields, and even in other data mining frameworks such as pattern discovery. In this application, a collection of patterns is used to estimate the frequencies of other patterns. In [14], the authors address the problem of discovering an ordering among this collection of patterns. The criteria for selecting the ordering here is decreasing informativeness for the estimation method. Thus, while the criteria, and hence the techniques used, vary depending on the application, the goal is the same: to find an interesting partial order in data.

Before we describe our methodology in detail, we introduce and define all the concepts used in this paper. Some
of these concepts are also formally defined in [17], which is most closely related to our work.

1.1. Definitions and notation

The data can be represented without loss of generality as a \{0, 1\} matrix. Here, rows represent the sites and columns represent the genera. For any site/genus combination, the matrix has a 1 if a record of the genus is present at the site, and 0 otherwise. We represent this matrix by \(\alpha\) and refer to the element in row \(i\) and column \(k\) by \(\alpha_{i,k}\). Using this information, we wish to find the chronological ordering of the sites. Observe that any permuted matrix \(\beta\) is equivalent to a total ordering of the sites; illustrating the equivalence between total orders, permutations and matrices.

To illustrate many of the concepts introduced in this paper, we use Example \(E1\), presented in Table 1. We see for instance that genus \(C\) is present at sites \(a, d,\) and \(e\).

<table>
<thead>
<tr>
<th></th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(b)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(c)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(d)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(e)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

*Table 1. Example \(E1\): data matrix \(\alpha_1\)*

Throughout this paper, we use index \(k\) for genera, and \(i, j, l\) for sites. As is convenient, we interchangeably use rows/sites/nodes and genera/columns. Let \(N\) and \(M\) be the set of sites and genera, respectively. Let \(n := |N|\) and \(m := |M|\). The size of the problem is therefore \(O(nm)\). For ease of exposition, we also treat \(N\) and \(M\) as lists.

An order (partial or total) is a collection of precedence relations among the sites. Thus, it is natural to represent an order as a set of ordered pairs of elements. In such a set, we include as elements all pairs of sites that have a precedence relation amongst them. Let \(T\) be a partial order. Using \(i <_T j\) to denote that site \(i\) precedes site \(j\) in the partial order \(T\),

\[
T = \{(i, j) : i \in N, j \in N, i <_T j\}.
\]

Naturally, not all sets of ordered pairs represent a partial order. Next, we provide conditions on when a set of pairs of elements represents a partial order. We revisit Example \(E1\), and introduce the set of ordered pairs \(T_1\). From Definition 1, this set of ordered pairs represents a partial order.

\[
T_1 = \{(a, c), (a, e), (a, d), (a, b), (c, b), (e, d), (e, b), (d, b)\}
\]

Definition 1 Partial Orders:

Any collection \(T\) of ordered pairs of sites corresponds to a partial order if and only if

- The order \(T\) is transitive; i.e., if \((i, j) \in T\) and \((j, l) \in T\), then \((i, l) \in T\) for all \(i \neq j \neq l\).
- If \((i, j) \in T\), then \((j, i) \notin T\).
- \((i, i) \notin T\).

A total order is a special case of partial orders: there exists a precedence relation between all pairs of elements. In other words, for any two sites \(i\) and \(j\), either \(i <_T j\) or \(j <_T i\). Thus, a partial order \(T\) is a total order if and only if \(|T| = n(n - 1)/2\). Since this can be a large set, for ease of exposition, we present total orders as permutations of sites. Revisiting Example \(E1\), we introduce total order \(T_2\).

\[
T_2 = \{a, e, c, d, b\}
\]

An order can also be represented using a graph where the sites correspond to nodes, and there exists an arc from node \(i\) to node \(j\) if and only if \((i, j) \in T\). This demonstrates the equivalence between partial orders and graphs. Since any order is transitive (Definition 1), this directed graph must be acyclic. Redundant arcs correspond to ordered pairs which can be derived from other ordered pairs using transitivity.

Definition 2 Redundant Arcs/Ordered Pairs:

Given order \(T\), an ordered pair \((i, j) \in T\) is redundant if there exists \(l \in N\) such that \(i <_T l <_T j\).

For example, in partial order \(T_1\), arc \((a, d)\) is redundant since \(a <_T e <_T d\). In Figure 1, we present the graphical representation of orders \(T_1\) and \(T_2\).

![Figure 1. Graph defined by orders \(T_1\) and \(T_2\).](image-url)

Given matrix \(\alpha\) and partial order \(T\), a non-existing genus \(k\) at site \(i\) is called a Lazarus event if genus \(k\) exists at sites before and after (as defined by \(<_T\)) site \(i\). In both \(T_1\) and \(T_2\), genus \(D\) of site \(e\) is a Lazarus event. The Lazarus count of a partial order \(T\) is the number of Lazarus events.
**Definition 3** *Lazarus Count:*

The Lazarus count of a partial order $T$, denoted by $L(T)$, is the number of Lazarus events. Formally,

$$L(T) = |\{(i, k) : (\alpha_{i,k} = 0) \\
\wedge (\exists j < T i : \alpha_{j,k} = 1) \wedge (\exists j > T i : \alpha_{i,j,k} = 1)\}|.$$ 

We leave it to the reader to verify that the Lazarus counts of orders $T_1$ and $T_2$ are:

$$L(T_1) = 4; \quad L(T_2) = 6.$$ 

Consider two partial orders $T_a$ and $T_b$ such that $T_a \subseteq T_b$. In other words, every precedence relation between sites $i$ and $j$ in $T_a$ also exists in $T_b$. Therefore, if genus $k$ at site $i$ is a Lazarus event for partial order $T_a$, then it is also a Lazarus event for partial order $T_b$. We have proved the following.

**Theorem 1** *If $T_a \subseteq T_b$, then $L(T_a) \leq L(T_b)$.*

Observe that the trivial partial order (i.e., when there is no order among the data; $T = \emptyset$) has Lazarus count 0. This is because, for any site $i$, there exist no sites that precede/follow it in the partial order $\emptyset$. Therefore, we restrict ourselves to total orders while minimizing Lazarus counts. We define the minimum Lazarus count $l^*(\alpha)$ as the minimum over all total orders for given matrix $\alpha$. For $\alpha_1$, defined in Example 1, the minimum Lazarus count is 1; i.e.,

$$l^*(\alpha_1) = 1 \quad (1)$$

$T_3$ attains this Lazarus count, where $T_3 = \{e, d, a, b, c\}$. There may be other minima; i.e., other total orders with minimum Lazarus counts. For example, the total order defined by $T_4$ also has $L(T_4) = 1$, where $T_4 = \{e, d, b, a, c\}$.

### 1.2. Problem discussion and objectives

Recall that we wish to discover the chronological order of fossil sites. To see why minimizing Lazarus counts is a good objective, consider a particular genus $k$. If the matrix is sorted correctly, then the values of column $k$ will never change from 1 to 0 and back to 1, since this would indicate that the genus became extinct and reappeared later. Thus, we want to find whether matrix $\alpha$ has the “consecutive-ones” property; i.e., whether there exists some matrix $\beta$ obtained by permuting the rows of $\alpha$ such that all 1s in any column of $\beta$ are adjacent to each other. In fact, this is an easy problem. (One can obtain a consecutive-ones permutation of a matrix in linear time, if such a permutation exists [4, 10].) Therefore, if $\alpha$ is perfect (no errors), then chronologically sorting sites is an easy problem. Revisiting Example 1, we see from equality 1 that there exist errors in $\alpha_1$. However, if genus $B$ of site $a$ in data set $\alpha_1$ is equal to 1, then the optimal Lazarus count (attained by $T_3$) is 0; i.e., the resulting matrix is error-free.

However, there are often errors in the data, and thus we would like to minimize the number of Lazarus counts. This is a hard problem, and has been approached by a variety of heuristic techniques [3, 2]. Furthermore, these sites have a large number of “false zeros”, often as much as half of the data. This is not surprising since the existence of a genus during the time of a particular site does not mean that it will be present in the fossils at the site. Many 1s get recorded as 0s. Therefore, the optimal Lazarus count for paleontological data tends to be quite large.

Since there are false 0s in the data, it is difficult to determine an ordering between some pairs of sites based on the data. Techniques that find a total order enforce an order among all sites, even those that should not be ordered. This usually manifests itself in the fact that many total orders are optimal (have the same minimum Lazarus count).

We believe that the best method should discover a partial order. This argument is also the basis of [13, 17], where the authors consider a different approach to solving the problem. Furthermore, any technique that discovers partial orders must not attempt to find a partial order that minimizes the Lazarus count, since $T = \emptyset$ has a Lazarus count of 0.

Minimizing Lazarus counts is not the only criteria that can be used for minimizing the number of Lazarus events. For instance, one could minimize the number of blocks of 0s with 1s above and below (number of blocks of consecutive Lazarus events as opposed to the total number of Lazarus events). This metric has been argued to be more appropriate for certain ordering problems; in particular, the problem of mapping chromosomes [1, 5]. This problem also reduces to the consecutive-ones problem in the absence of errors.

We reiterate that all the methodologies and metrics proposed in this paper extend to other applications for discovering interesting partial orders in data. However, the techniques used may differ since the sorting criteria varies.

### 1.3. Contribution

From the previous discussion, it is clear that we wish to discover partial orders with low Lazarus counts (to find the right ordering), but at the same time not achieve this objective by discovering partial orders with small size. For two partial orders with the same Lazarus count, the one with lower size is unarguably better since it only captures the important ordered pairs. We wish to discover a partial order that satisfies the following property.

**Definition 4** *Fundamental Partial Order (FPO):*

A partial order $P$ satisfies the FPO property if

- Among all partial orders with the same Lazarus count, $P$ has the smallest size.
- The optimal total order has Lazarus count $\leq L(P)$.
This is a key contribution: we believe that the ideal partial order satisfies FPO, and claim that this is a better measure than other techniques used in the literature. The second condition is required to eliminate partial orders that have a small Lazarus count because their size is small.

This notion (property FPO) can be analogously defined for any criteria for selecting the ordering. For chronologically ordering fossil sites, the criteria for ordering is to minimize Lazarus counts. For mapping chromosomes, the criteria for selecting the ordering can be reduced to finding appropriate dependency graphs. Nevertheless, the notion of FPO can be defined for this problem. Furthermore, the methodology proposed in Section 1.4 can be applied. However, the analysis and solution techniques will change.

1.4. Methodology

Now, we propose our technique for finding partial orders that satisfy property FPO. We believe that a minimal partial order is a good candidate for satisfying property FPO; see Theorem 2. In a minimal partial order, removing any precedence relation in the order reduces its Lazarus count.

**Definition 5 Minimal Partial Order:**
Given a number, we say that a partial order is minimal with respect to if

1. \( L(P) = s \), and
2. For all partial orders \( P' \subset P \), \( L(P') < s \).

In Theorem 2, we prove that every partial order that satisfies property FPO is minimal. This is the primary justification behind restricting our search to minimal partial orders.

**Theorem 2** For partial order \( P \), let \( L(P) = s \). If \( P \) satisfies property FPO, then it is minimal with respect to.

**Proof:** Consider any \( P_1 \subset P \). To prove that \( P \) is minimal with respect to, we need to show that the Lazarus count of any such partial order \( P_1 \) is less than \( s \), i.e., \( L(P_1) < s \). From Theorem 1, \( L(P_1) \leq s \). We show that \( L(P_1) \neq s \) by contradiction. Assume that \( L(P_1) = s \). Since \( P \) satisfies FPO, it is the smallest among all partial orders with the same Lazarus count. In particular \( |P| \leq |P_1| \). We have a contradiction since \( P_1 \subset P \).

The converse is not necessarily true. We wish to discover the minimal partial order with minimum size, since such a partial order satisfies property FPO. Let \( M(T) \) be the set of all minimal partial orders such that \( P \subset T \) and \( L(P) = L(T) \). We want to find \( P_m \), defined as

\[
P_m = \arg\min \{|P| : P \in M(T)\}
\]

This is a hard optimization problem, and is computationally intractable. Observe that the definition of \( T_m \) depends on the total order \( T \). Since we would like to discover partial orders that satisfy property FPO and have low Lazarus counts, we discover minimal partial orders with the same Lazarus count as the minimum total order (= \( l^* \)). We believe that this is a good technique for finding interesting partial orders among all those with property FPO.

We solve the problem in two stages. We first find interesting total orders, and then discover minimal partial orders.

- **Problem 1** \( (P1) \): Given a matrix \( \alpha \) of sites as rows and genera as columns, we find a total order \( T \) that minimizes the total Lazarus count.

- **Problem 2** \( (P2) \): Given total order \( T \), we find a minimal partial order \( P \subset T \) with the same Lazarus count.

We mention that our proposed algorithms for Problems \( P1 \) and \( P2 \) are not parametric in nature. They are randomized algorithms; however, we will see in Section 4.3 that they are quite indifferent to the randomization.

We mainly compare against the pioneering work of [17], which also attempts to discover partial orders among fossil sites using \( \{0, 1\} \) data. The authors mention that they wish to discover partial orders that minimize \( L(P) + \delta(P) \), where \( \delta \) is a monotonically decreasing function of the size of \( P \). However, such an approach (if solved to optimality) will always yield total orders. Since this is a difficult problem, the authors develop an algorithm that first samples many fragments (small number of sites) and then finds partial orders from these fragments. The authors consider fragments of 5 sites each, collect a predetermined number of good fragments, and merge them to discover partial orders.

In [17], the authors evaluate the quality of their solutions using \( L(P) \). We believe that trying to only minimize the Lazarus count can result in small partial orders (no incentive against trivial orders). On the other hand, the objective \( L(P) + \delta(P) \), while avoiding trivial orders, prefers total orders to partial. Our two-phased approach for discovering minimal partial orders that satisfy property FPO avoids both pitfalls (trivial partial orders and total orders).

While the approach of [17] finds interesting partial orders, these partial orders generally have much higher Lazarus counts than the optimal total order. Moreover, these partial orders are usually not minimal. Thus, these partial orders do not satisfy property FPO. Furthermore, the algorithm in [17] is highly parametric, and needs a large amount of tuning to find interesting partial orders. We also show that our algorithm discovers partial orders that are much closer to the accepted ordering/classification (by human experts) of the fossil sites in a real data set. We present a detailed computational comparison in Section 4.

In Section 2, we first attempt to solve Problem 1 exactly using integer programming (IP) techniques. We present two formulations, and discuss why both formulations are ineffective in solving even small problems. As a result, we re-
sort to heuristic techniques. We present a heuristic in Section 2.4. Given a total order \( T \), we find a minimal partial order in Section 3. In Section 4, we test our algorithm using a real data instance, and illustrate the effectiveness of our method. Finally, we conclude in Section 5 with some observations and directions for future research.

2. Discovering total orders

We first present two integer programming formulations for Problem 1. For a detailed study of integer programming, its applications and solution techniques, we refer the reader to [15]. Integer programming problems are usually solved using enumerative bounding schemes such as branch-and-bound. The time needed to solve problems optimally in this framework largely depends on the strength of the relaxation used. Therefore, the strength of the linear programming (LP) relaxation largely determines the computational effectiveness of any integer programming formulation.

The two formulations presented here are derived from the two paradigms for representing orders: permutations and graphs. In the first, we use variables indicating whether a row is in a certain position in the permuted matrix. In the second, we use variables indicating whether an arc exists between two sites in the graph representation of the order.

In both formulations, variables \( z \), \( w \), and \( y \) are introduced to calculate the Lazarus count of a given partial order. Variable \( z^k_j \) is forced to 1 if there exists at least one site \( i \) preceding site \( j \) such that genus \( k \) is present at site \( i \). Variable \( w^k_j \) is forced to 1 if site \( j \) precedes at least one site \( i \) such that genus \( k \) is present at site \( i \). Therefore, a Lazarus event occurs at genus \( k \) of site \( j \) if both \( w^k_j \) and \( z^k_j \) are 1 and genus \( k \) is not present at site \( j \); in this case, variable \( y^k_j \) is forced to 1.

We tried both formulations on random problem instances, but were unable to solve instances larger than \( 15 \times 10 \) in less than an hour (using CPLEX[11] 9.0 on a 1.7GHz IBM ThinkPad). This can be attributed to their weak LP relaxations. In Section 2.3, we study the LP relaxations of both formulations and discuss why they are ineffective in solving problems of even reasonable size.

Finally, we present an improvement heuristic that efficiently discovers an interesting total order. We remark that real data instances are usually of size greater than \( 100 \times 100 \). No brute-force search techniques can be expected to complete in a reasonable amount of time; since the size of the feasible solution space exceeds \( 100! \).

2.1. Formulation: Permutations

Given data matrix \( \alpha \), we find the total order \( T \) with minimum Lazarus count by using variables for row \( i \) in position \( j \). In this paradigm (permutations of sites), site \( i \) precedes site \( j \) \((i <_T j)\) if the position of row \( i \) is less than the position of row \( j \). We describe the required decision variables in Table 2. Variables \( v \) indicate the actual ordering of sites.

| \( v_{i,j} \) | 1 if row \( i \) is in position \( j \); 0 otherwise. |
| \( x^k_j \) | Value of genus \( k \) in position \( j \). |
| \( z^k_j \) | 1 if \( \exists i \in N \) such that \( i <_T j \) and \( \alpha[i,k] = 1 \). |
| \( w^k_j \) | 1 if \( \exists i \in N \) such that \( i >_T j \) and \( \alpha[i,k] = 1 \). |
| \( y^k_j \) | 1 if genus \( k \) at position \( j \) is a Lazarus event. |

Table 2. Variables of Formulation \( \mathcal{F}1 \)

Observe that variables \( x \) can be derived in terms of variables \( v \); these are introduced only for the sake of exposition. Problem 1 can now be formulated as:

\[
\begin{align*}
\min & \sum_{j \in N} \sum_{k \in M} y^k_j \\
\text{s.t.} & \quad \sum_{j \in N} v_{i,j} = 1 \quad \forall i \in N \quad (2) \\
& \quad \sum_{i \in N} v_{i,j} = 1 \quad \forall j \in N \quad (3) \\
& \quad \sum_{i \in N} v_{i,j} \alpha[i,k] = x^k_j \quad \forall j \in N, \forall k \in M \quad (4) \\
& \quad z^k_j \geq x^k_{j-1} \quad \forall j \in N, \forall k \in M \quad (5) \\
& \quad z^k_j \geq x^k_{j+1} \quad \forall j \in N, \forall k \in M \quad (6) \\
& \quad w^k_j \geq w^k_{j+1} \quad \forall j \in N, \forall k \in M \quad (7) \\
& \quad w^k_{j+1} \geq x^k_{j+1} \quad \forall j \in N, \forall k \in M \quad (8) \\
& \quad z^k_j + w^k_j - x^k_j - 1 \leq y^k_j \quad \forall j \in N, \forall k \in M \quad (9) \\
& \quad v \in \{0, 1\} \\
& \quad 0 \leq w, y, z \leq 1
\end{align*}
\]

Constraints 2 ensure that each row is assigned to some position. Constraints 3 ensure that each position is assigned to some row. Constraints 4 define \( x \). Constraints 5 and 6 define \( z^k_j \) by ensuring that it is 1 if there exists a 1 before position \( j \) for genus \( k \). Constraints 7 and 8 define \( w^k_j \) by ensuring that it is 1 if there exists a 1 after position \( j \) for genus \( k \). Constraints 9 ensure that variable \( y^k_j \) is 1 whenever genus \( k \) at position \( j \) is a Lazarus event. Observe that we only enforce \( v \) to be binary; other variables are continuous.

**Theorem 3** When \( v \) is \( \{0, 1\} \), \( x, w, z, \) and \( y \) are also \( \{0, 1\} \).

**Proof:** From constraints 4, variables \( x \) are binary whenever variables \( v \) are binary. From constraints 5-8, all variables \( z \) and \( w \) are not smaller than some variable \( x \). In other words, variables \( z \) and \( w \) are bounded from below by some variable \( x \). Since we are minimizing an increasing function of \( y \), and \( y \) is a monotonically increasing function of \( z \) and \( w \), all variables \( z \) and \( w \) will be set to their smallest possible values in any optimal solution to Formulation \( \mathcal{F}1 \).
As variables $z$ and $w$ are bounded from below by variables $x$, which are binary when $v$ is binary, we are done.

In the preceding proof, we also show that we do not need upper bounds on variables $y$, $w$, and $z$. This formulation has $O(n \max\{m, n\})$ variables and $O(nm)$ constraints.

### 2.2. Formulation: Directed Graphs

Here, we discover a total order $T$ with minimum Lazarus count $l^*(\alpha)$ by using the variables for arcs between pairs of sites (nodes). In this paradigm, site $i$ precedes site $j$ ($i <_T j$) if there exists an arc from node $i$ to node $j$. We describe the required decision variables in Table 3. Variables $u$ indicate the presence of arcs between pairs of nodes. Recall that we consider both redundant and non-redundant arcs in this paradigm. As a result $u_{i,j} = 1$ corresponds to all cases where site $i$ precedes site $j$, not just the adjacent sites.

<table>
<thead>
<tr>
<th>$u_{i,j}$</th>
<th>$z^k_j$</th>
<th>$w^k_j$</th>
<th>$y^k_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 if there is an arc between node $i$ and node $j$.</td>
<td>1 if $\exists i \in N$ such that $i &lt;<em>T j$ and $\alpha</em>{[i,k]} = 1$.</td>
<td>1 if $\exists i \in N$ such that $i &gt;<em>T j$ and $\alpha</em>{[i,k]} = 1$.</td>
<td>1 if genus $k$ at site $j$ is a Lazarus event.</td>
</tr>
</tbody>
</table>

| $T = \{ (i,j) : (j,i) \in T \}$ |

Furthermore, since we are minimizing an increasing function of variables $y$, we do not need upper bound constraints for all variables. This formulation has $O(n \max\{m, n\})$ variables and $O(n^2 \max\{m, n\})$ constraints.

### 2.3. Weak LP relaxations

Both formulations $F_1$ and $F_2$ perform poorly primarily because the problem is very symmetric; in the sense that many total orders usually have the same Lazarus count. Often, such sets of total orders (with same Lazarus count) can be found by minor perturbations among themselves. Furthermore, given a total order $T$, the total order given by its exact reversal also has the same Lazarus count. More formally, given total order $T$, we have $L(T) = L(T')$, where $T' = \{(i,j) : (j,i) \in T\}$.

This symmetry manifests itself in two ways. The former is more critical, and causes weak LP relaxations, resulting in poor performance of integer programming techniques. The strength of a relaxation is often measured using its “integrality gap”, and is calculated as 100%($1 - \gamma$) between the LP and IP optimal solutions). A smaller integrality gap usually results in better performance. Unfortunately, the optimal LP relaxation solution for both formulations $F_1$ and $F_2$ is always 0; proved in Theorems 5 and 6. Thus, the integrality gap for both formulations is always 100%.

### Theorem 5
For any $\alpha$, the optimal LP solution for $F_1$ is 0.

**Proof:** We prove this by constructing a feasible solution with objective 0. Consider the solution given by $u_{i,j} = 1/n$, $v_{i,j} = 1/n$, $\forall i \in N$, $\forall j \in N$. This allocation to variables $u$ satisfies constraints 2 and 3. Let $\gamma_k = \sum_{i \in N} \alpha_{[i,k]}$. Then, from constraint 4, we get $x^k_j = \gamma_k/n$, $\forall j \in N$, $\forall k \in M$. Satisfying constraints 5-8, for all $j \in N$, $k \in M$, we set $w^k_j = z^k_j = \gamma_k/n$. Since $\gamma_k \leq n$, setting $y^k_j = 0$, $\forall j \in N$, $\forall k \in M$ satisfies constraints 9.

### Theorem 6
For any $\alpha$, the optimal LP solution for $F_2$ is 0.

**Proof:** We prove this by constructing a feasible solution with objective 0. Consider the solution given by $u_{i,j} = 1/2$, $\forall j \in N$, $\forall i \in N$, $\forall j \in N$, $\forall i \in N$. This allocation to variables $u$ satisfies constraints 10, 11, and 12. Satisfying constraints 13 and 14, for all $j \in N$, $k \in M$, we set $w^k_j = z^k_j = 1/2$. Now, setting $y^k_j = 0$, $\forall j \in N$, $\forall k \in M$ satisfies constraints 15.
The latter manifestation of symmetric structure is that it significantly worsens the convergence of the branch and bound tree. As a result, the tree has to explore and prune out many more nodes before it can terminate. Since all exact algorithms to solve $\mathcal{P}1$ are too time-consuming even for small instances, we turn our attention to heuristic techniques.

2.4. Heuristic approaches

Recall that the LP relaxations of formulations $\mathcal{F}1$ and $\mathcal{F}2$ are weak (optimal LP solution is always 0). Furthermore, these optimal solutions are symmetric and contain no useful information. For Formulation $\mathcal{F}1$, the optimal LP relaxation solution has $v_{i,j} = 1/n$, $\forall i \in N$, $\forall j \in N$. For Formulation $\mathcal{F}2$, the optimal LP relaxation solution has $u_{i,j} = 1/2$, $\forall i \in N$, $\forall j \in N, j \neq i$. Thus, any heuristic based on the LP relaxation solutions is potentially no better than a random permutation of the sites.

Problem $\mathcal{P}1$ lends itself to improvement heuristics since it is easy to evaluate the Lazarus count for a given permutation of sites. For ease of exposition, we refer to permutations of sites, instead of matrices/graphs. Given a permutation, our heuristic improves the solution (i.e., finds a permutation with smaller Lazarus count) using the following operations: exchange, reverse, or insert; see Figure 2.

- $EXC(\pi, i, j)$: By applying $EXC(\pi, i, j)$, we exchange sites in positions $i$ and $j$ to obtain $\pi'$ from $\pi$.
- $REV(\pi, i, j)$: By applying $REV(\pi, i, j)$, we reverse the sequence of sites between (and including) positions $i$ and $j$. Without loss of generality, we assume $i < j$.
- $INS(\pi, i, j)$: By applying $INS(\pi, i, j)$, we insert the site in position $i$ at position $j$ (and adjust other positions appropriately) to obtain permutation $\pi'$.

![Figure 2. Operations in Algorithm A1.](image)

Thus, given a total order defined by permutation $\pi_0$, our algorithm (A1) uses the preceding operations for all combinations of sites so long as it keeps finding better solutions.

1. $\pi := \pi_0$
2. $l = L(\pi)$
3. While $\exists i, j$ such that $L(EXC(\pi, i, j)) < l$,
   a. $\pi := EXC(\pi, i, j)$, $l := L(\pi)$.
4. While $\exists i, j$ such that $L(REV(\pi, i, j)) < l$,
   a. $\pi := REV(\pi, i, j)$, $l := L(\pi)$.
5. While $\exists i, j$ such that $L(INS(\pi, i, j)) < l$,
   a. $\pi := INS(\pi, i, j)$, $l := L(\pi)$.
6. If $L(\pi) < l$, goto Step 2.

In each of these steps, the sequence in which $i, j$ are chosen depends on the implementation of the algorithm. One can develop various “ranking” criteria and sort the sites based on such criteria. For the $EXC$ and $REV$ operations, we consider pairs $i, j$ that are most dissimilar first. The motivation behind this is that changing the relative positions of highly dissimilar sites changes the Lazarus count most dramatically. For $INS$, we consider sites that are most similar first. By doing so, we try to put similar sites together first. Thus, we use a notion of similarity between sites based on the ranking criteria to choose pairs of sites. A simple criterion that works well is sorting the sites based on the number of active genes on the site (i.e., number of 1s in row). Of course, one could use any ranking criteria based on the pairwise distance (for instance, the $L_1$ distance) between sites.

Another strategy that speeds up the algorithm is switching between the three operations often (see Section 4 for a computational example). However, the final total order does not seem to depend on the criteria. In our computations, we choose pairs of sites randomly, and initialize the algorithm with a random $\pi_0$. In Section 4.3, we show that Algorithm A1 is quite insensitive to the randomization.

3. Discovering Partial Orders

Now we present our technique for discovering a minimal partial order, given a total order $T$. This partial order must be minimal with respect to $L(T)$. In other words, given a total order $T$, we wish to find $P \subseteq T$ such that $L(P) = L(T)$, and $L(P') < L(P), \forall P' \subset P$.

Any partial order $P$ can be represented as an acyclic directed graph $G_P = (V, A_P)$, where $V \equiv N$ is the set of nodes, and $A_P$ is the set of arcs. In this representation, we include both redundant and non-redundant arcs in $A_P$ (see Definition 2). For each ordered pair of sites $(i, j)$ in the partial order $P$, there exists an arc from node $i$ to node $j$ in $A_P$.

We define $f_P(i)$ as the set of sites in $P$ that are immediate
successors of node \(i\); i.e., \(f_P(i)\) is the set of all nodes to whom there exists a non-redundant arc from node \(i\). Thus,
\[
f_P(i) = \{ j : (i <_P j) \land (\neg (\exists l : i <_P l <_P j)) \}.
\]
The set \(f_P(\cdot)\) is crucial in the development of an incremental algorithm to discover a minimal partial order. Let \(j \in f_P(i)\). Given a partial order \(P\), we can obtain partial order \(P' \subseteq P\) such that \(|P'| = |P| - 1\) by removing \((i, j)\) from \(P\), for some \(i \in N\). In fact, this is the only operation that reduces \(|P|\) by exactly 1. In other words, if we eliminate \((i, l)\) where \(l \notin f_P(i)\), then the resulting set of ordered pairs is not a partial order since it is not transitive.

The simple operation \(REDUCE\) removes exactly one element from a partial order such that the resulting collection of elements is also a partial order.

- \(REDUCE(P, i, j)\). We define this operation only for \(j \in f_P(i)\). By applying \(REDUCE(P, i, j)\), we remove the ordered pair \((i, j)\) from \(P\) to obtain \(P'\); i.e., \(P' = P \setminus \{(i, j)\}\).

Next, we prove that this simple operation is non-exclusive; i.e., all minimal partial orders can be found using it.

**Theorem 7** Any partial order \(P'\) such that \(P' \subseteq P\) can be discovered by a sequence of \(REDUCE\) operations from \(P\).

**Proof:** (By construction.) Consider any partial order \(\hat{P}\) such that \(\hat{P}' \subseteq \hat{P} \subseteq P\). Let \(D\) be the set of arcs that define non-redundant orderings in \(\hat{P}\), but not in \(\hat{P}'\). Since \(D \neq \emptyset\), by applying \(REDUCE(\hat{P}, i, j)\) for some \((i, j) \in D\), we get a new partial order \(\hat{P}'\) such that \(\hat{P}' \supseteq \hat{P}\). Setting \(\hat{P} := \hat{P}'\) and repeating this until \(D = \emptyset\), we get \(P'\) from \(P\).

From Theorem 1, \(L(P') \leq L(P)\) for all \(P' \subseteq P\). In other words, for all partial orders that are subsets of partial order \(P\), the Lazarus count must be no greater than the Lazarus count of \(P\). Furthermore, if the Lazarus count of all such partial orders is strictly less the Lazarus count of \(P\) (i.e., \(L(P') < L(P)\), \(\forall P' \subseteq P\)), then partial order \(P\) is minimal with respect to \(L(P)\) (from Definition 5).

This motivates an intuitive algorithm for discovering minimal partial orders. We keep removing ordered pairs from our partial order as long as the Lazarus count does not decrease. In other words, we successively apply operation \(REDUCE\) while ensuring that the Lazarus count remains the same. When we can not find such an ordered pair to remove, we terminate with a minimal partial order. We formalizes this method for finding a minimal partial order with respect to \(L(T)\) as Algorithm \(A_2\).

1. \(P := T\)
2. While \(\exists j \in f_P(i)\) such that \(P' = REDUCE(P, i, j)\) and \(L(P') = L(P)\),
   (a) \(P' := P\)

In this algorithm, the sequence in which \(i\) and \(j\) are chosen is unspecified above. For a suitably defined distance metric, we could choose pairs of sites \(i, j\) that are most similar first. This is motivated by the argument that such arcs between similar sites are more likely to be redundant.

In our implementation, we chose these pairs of sites randomly. In Section 4.3, we present a computational test that indicates that our algorithm is insensitive to the randomization. The implementation can be done more efficiently using appropriate data structures that track the “next” and “previous” 1s for each Lazarus event. For any implementation of Algorithm \(A_2\), we state the following result.

**Theorem 8** \(A_2\) terminates with a minimal partial order.

4. Experiments

In this section, we analyze the performance of our algorithms. We study the partial order discovered in the best run of our algorithm, and compare our results with [17]. Recall that our algorithm is non-parametric. However, it is randomized. In Section 4.3, we show that our algorithm is quite insensitive to the seeding. On the other hand, the algorithm presented in [17] is highly parametric. Therefore, it requires a lot of tuning. This is why, in Table 4, we present only one run of our algorithm, while presenting a statistical summary of the runs in [17].

We test our algorithms on only one dataset, for two reasons. Firstly, we are not aware of other datasets of comparable size that have been classified by experts with domain information. Secondly, this dataset has been used in other work, in particular [17], allowing for a direct comparison.

This data is taken from the NOW database [12]. This database contains a record of Miocene to Pleistocene land mammal species at various Eurasian fossil sites. These sites have been classified into \(MN\) classes by experts with domain information. In fact, we analyze our algorithm by comparing its partial ordering with the partial ordering obtained from this classification. To compare our results with [17], we consider the same set of sites and genera. (124 sites and 139 genera) We remark that this instance has a very large solution space (> 100! feasible orders).

Recall that Algorithm \(A_1\) uses three operations: \(EXC, \text{REV},\) and \(\text{INS}\). We observed that the algorithm is much faster when we switch between the operations often. In this experiment, we vary how often we switch between operations. We run the algorithm with four settings: change operation after every \(n/6, n/3, n,\) and \(3n/2\) improvements, respectively. On the \(y\)-axis, we plot the Lazarus count of the total order obtained by the algorithm as a function of time (on the \(x\)-axis). From Figure 3, we see that the Lazarus count reduces more appreciably in the initial stages when we switch operations more often. This indicates that the
marginal improvement with each operation reduces with time. Nevertheless, the final Lazarus count generally does not seem to depend on the strategy.

![Lazarus count vs Time](image)

**Figure 3. Marginal output of Algorithm A1**

### 4.1. Attributes of partial orders

Let $MN(i)$ be the $MN$ classification for site $i$. For any total order $T$ of 124 sites, $|T| = 7626$. To quantitatively compare the performance of our algorithm with [17], we use the following attributes of a partial order $P$. The first two attributes quantify the size and Lazarus count of a partial order. The latter three measure its “closeness” to the partial order $P_{MN}$ defined by the $MN$ classification scheme.

- **Completeness:** $c_P$.
  The ratio between the size of the partial order $P$ and the size of a total order; $c_P = |P|/|T|$. This measures how similar $P$ is to a total order. While it is difficult to compare the completeness of two arbitrary partial orders, given two partial orders with the same Lazarus count, lower the completeness the better.

- **Lazarus count:** $L(P)$
  It is difficult to compare Lazarus counts of two arbitrary partial orders, since small partial orders have low Lazarus count. However, given two partial orders with same completeness, lower the Lazarus count the better.

- **Success count:** $c_P$
  The number of ordered pairs in $P$ oriented in the same direction as in the $MN$ classification. Formally,
  \[
  c_P = |\{(i, j) : ((i, j) \in P) \wedge (MN(i) < MN(j))\}|
  \]
  This is a measure of how often the partial order got the ordering right; the higher the better.

- **Similarity count:** $d_P$
  We define this as the number of pairs of sites which are not ordered in the partial order $P$, but have “similar” $MN$ classes. Here, we say that two $MN$ classes are similar if the absolute difference between them is no larger than 2. Formally,
  \[
  d_P = |\{(i, j) : ((i, j) \notin P) \wedge ((j, i) \notin P) \\
  \wedge (|MN(i) - MN(j)| \leq 2)\}|
  \]
  This is a measure of how often the partial order correctly did not order similar sites; higher the better.

- **Compatibility:** $r_P$
  We define this as the number of pairs of site that the partial order got right; either by ordering them in the right orientation, or not ordering them if they are too similar. Formally, $r_P = (c_P + d_P)$. Therefore, higher the compatibility the better.

We note that in [17], the authors also include, for all sites $i \in N$, the ordered pair $(i, i)$ in the description of partial orders. This offsets the size of any partial order $P$ by $n$. but does not affect the Lazarus count, success count, similarity count, or the compatibility. However, the value of completeness changes slightly; we adjust for this appropriately in our comparisons.

Recall that we wish to discover a partial order (say $P$) satisfying property FPO. We want $P$ to have the lowest completeness among all partial orders with Lazarus count $L(P)$. To illustrate the strength of our methodology, we also want to prove that partial orders that satisfy property FPO are closer to $P_{MN}$. To this effect, we introduce the notion of dominance for comparing two partial orders. One partial order dominates another if it scores better in terms of both Lazarus count and success count.

**Definition 6** **Dominance:**
We say that partial order $P_0$ dominates partial order $P$ if
1. $c_{P_0} > c_P$, and
2. $L_{P_0} < L_P$.

We see next that while the algorithm in [17] gives interesting partial orders, they are dominated by our partial order.

### 4.2. Interpretation of results

We present the results of a run our algorithms (A1 followed by A2), and analyze the output in detail. We present the final partial order in Figure 4 (created using Graphviz[7]). We refer to this partial order as $P_0$. For the sake of illustration, we only present the non-redundant arcs in the graph. Furthermore, we break the graph into three pieces; the graph should be read from top to bottom and from left to right. Each node represents a site, and the number in square braces is its $MN$ classification.
Figure 4. Partial order found by running Algorithm \( A_1 \) followed by Algorithm \( A_2 \)
In Table 4, we compare the performance of our algorithm with [17]. In columns 2, 3, and 4, we present the average, best, and standard deviation for 24 runs (with different settings) of the algorithm presented in [17]. We note that in [17], the best values of the various attributes are from different runs. In column 1, we present the values of the preceding attributes for the partial order $P_0$ given in Figure 4. In column 5, we present these values for $P_{MN}$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$ Partial orders from [17]</td>
<td>$P_{MN}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$L(P)$</td>
<td>3228</td>
<td>3687</td>
<td>3233</td>
<td>212.3</td>
</tr>
<tr>
<td>2</td>
<td>$c_P$</td>
<td>6723</td>
<td>6474</td>
<td>6587</td>
<td>62.6</td>
</tr>
<tr>
<td>3</td>
<td>$d_P$</td>
<td>274</td>
<td>25.6</td>
<td>53</td>
<td>14.7</td>
</tr>
<tr>
<td>4</td>
<td>$r_P$</td>
<td>6997</td>
<td>6511</td>
<td>6588</td>
<td>74.6</td>
</tr>
<tr>
<td>5</td>
<td>$\gamma_P$</td>
<td>0.964</td>
<td>0.972</td>
<td>0.944</td>
<td>0.014</td>
</tr>
</tbody>
</table>

Table 4. Comparison of algorithms

Comparing columns 1 and 2, we see that our partial order $P_0$ has a lower Lazarus count than [17], and at the same time higher success counts, similarity counts, and compatibility. In fact we obtain lower Lazarus counts (columns 1 and 3) and higher success counts, similarity counts, and compatibility (columns 1 and 3) than all runs in [17]. In other words, $P_0$ dominates all the partial orders in [17]. This illustrates the effectiveness of our methodology and algorithm.

The biggest difference in performance between the two approaches is in compatibility (row 4). The compatibility of our partial order ($r_{P_0} = 6997$) is significantly larger than that of any run in [17] (maximum $r = 6588$). This shows that our technique matches up with the $MN$ classification system much better. Even comparing the success count (row 2) and similarity count (row 3) separately, we see that $P_0$ outperforms all partial orders in [17].

Looking at the success counts and completeness (rows 2 and 5), we see that partial order $P_0$ (column 1) has much higher success count (6723) than partial orders from [17] (column 2) even though its completeness is smaller. Thus, even though the size of $P_0$ is smaller, it still gets more orders in sync with the $MN$ classification.

We next analyze why partial orders we find are more likely to satisfy FPO. We see that $P_0$ has lower completeness (row 5) than [17]. This is because we find minimal partial orders, removing all orderings that do not contribute to the Lazarus count.

From column 6, we see that $P_{MN}$ has much lower partial orders, removing all orderings that do not contribute to the Lazarus count. We also see that $P_{MN}$ has much lower Lazarus count (row 1), but that can be attributed to its lower completeness. Also, $P_{MN}$ attains the maximum values of the compatibility parameter; $r_P(P_{MN}) = |T|$. This is no surprise since this parameter measures how well a partial order matches with the $MN$ classes.

Finally, we qualitatively analyze how the partial order $P_0$ matches with the $MN$ classification system. From Table 4, we see that $P_0$ matches very well with the classification system ($r_P(|T|) = 0.92$). For example, all the sites in the $MN = 9$ class (early late Miocene localities) cluster together (Can Ponsic/Esselborn to Los Valles de Fuentidueña). At the same time, it bodes well to reevaluate the classification for certain sites which seem highly out of place in $P_0$ (for instance, Layna and Poksheshy). In fact, even in the partial order illustrated in Figure 2 in [17], Layna, which has an $MN$ classification of 13 – 14, is placed alongside sites with $MN$ classification 17.

### 4.3. Sensitivity analysis

We present a statistical analysis of many runs of our algorithms, each run with a different seed to the random number generator. Table 5 summarizes the results of this experiment. In row 1, we present the statistics for the Lazarus count of the total order found by Algorithm $A_1$. In row 2, we present the statistics for the size of the minimal partial order discovered by Algorithm $A_2$. To compare “apples against apples”, all runs of Algorithm $A_2$ are seeded with the same total order as in Section 4.2. We repeated this experiment for algorithm $A_2$ with different initial total orders, but the numbers were comparable; we do not present them here. In columns 1, 2, 3, and 4, we display the average, minimum, maximum and standard deviation for these runs.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ave Min Max Std</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>.A1 Lazarus count</td>
<td>3236</td>
<td>3228</td>
<td>3244</td>
</tr>
<tr>
<td>2</td>
<td>.A2 Size</td>
<td>7409</td>
<td>7331</td>
<td>7451</td>
</tr>
</tbody>
</table>

Table 5. Robustness of algorithms

From row 1, we see that Algorithm $A_1$ is extremely robust. The standard deviation of the Lazarus counts is 3.7, less than 2% of the average (3236). Algorithm $A_2$ is also quite robust, with the standard deviation of the size of the partial orders at 26.3, less than 4% of the average partial order of the sample (7409). This illustrates that our algorithms, while not guaranteeing optimality, yield results that are quite good all the time. There is minimal dependence on the inherent randomization in the algorithms.

To illustrate how small this dependence is, we present the output of Algorithm $A_2$ in detail. Observe that the size of a total order of 124 sites is 7626. In Figure 5, we plot the size of the partial orders found by Algorithm $A_2$. We see
that the variation in the size of the partial order discovered by Algorithm A2 is small.

![Figure 5. Robustness of Algorithm A2](image)

We have shown that the performance of our randomized algorithms is highly insensitive to its randomization.

5. Conclusions

In this paper, we introduced a new methodology for discovering interesting partial orders in data, in that we aim to discover partial orders that satisfy property FPO. We focused on an application from paleontology where we need to chronologically order fossil sites. These sites can be ordered by minimizing Lazarus counts. To achieve this goal, we first calculated interesting total orders, and then determined a minimal partial order.

In the first problem, we wish to find the total order with minimum Lazarus count. We first formulated this problem using exact integer programming techniques. These integer programming formulations performed poorly even for small problems; we proved that this is because they have weak LP relaxations. As a result, we resorted to heuristics, and described an improvement algorithm to discover interesting total orders. Then, given this total order, we described an algorithm to discover a minimal partial order. We performed a sensitivity analysis on our algorithms, and showed that the performance is quite insensitive to this randomization.

Finally, we presented computations that illustrate the effectiveness of our algorithm, mainly comparing our results with the most closely related work of [17]. We showed that the partial order found by us is significantly better than partial orders presented in [17] for the same data set, for a variety of benchmarks. In particular, our partial order dominates all partial orders presented in [17].

For future work, we intend to continue in two distinct directions. In the former, we wish to focus on this application, and plan to study the problem of chronologically ordering fossil sites in more detail. In particular, we would like develop to more effective exact techniques for solving problems P1 and P2 (possibly using better LP formulations). In the latter direction, we wish to apply our methodology to other domains, on other ordering problems.

References