Some Heuristics for the 2-Poset Cover Problem

Gabriel Alberto A. Sanchez
Ateneo de Manila University
Loyola Heights, Quezon City
Philippines 1108
+632 4266071
gabriel.sanchez@obf.ateneo.edu

Proceso L. Fernandez
Ateneo de Manila University
Loyola Heights, Quezon City
Philippines 1108
+632 4266071
pfernandez@ateneo.edu

John Paul C. Vergara
Ateneo de Manila University
Loyola Heights, Quezon City
Philippines 1108
+632 4266071
jpvergara@ateneo.edu

ABSTRACT

Posets are abstract models that may be considered as generating a set of linear orders, which are permutations on some base set. The problem of determining a minimum set of posets that can exactly generate a specified input set of linear orders is referred to as the Poset Cover Problem, and this problem is NP-Hard in the general case. In this study, we investigate a constrained version of the problem, the 2-Poset Cover Problem, where there are exactly 2 posets that can generate a given input. We develop some heuristics for this and examine some properties related to the problem. Our heuristics are able to provide the correct solutions for a significant majority of the tested random instances. From the instances where the heuristics have failed, some insights were derived which may be helpful in determining the correct complexity class to which the 2-Poset Cover Problem belongs.

1. INTRODUCTION

In data mining, numerous patterns can be discovered for analyzing huge amounts of data by performing efficient algorithms. Previous data mining research found patterns such as sequential patterns given a collection of lists [1] (an example of this is finding frequent episodes in a given sequence of events [18]), and finding order constraints from a given collection of total orders. The latter is a data mining task called poset mining. It focuses on generating a partially ordered set (or sets) given a list of linear orders.

Formally, a partially ordered set (poset) is defined as an ordered pair $P = (V, \leq_P)$ where V is a finite set, and $\leq_P$ is a binary relation over V, i.e., $\leq_{Pc} \subseteq V \times V$. Every poset has three main properties, namely reflexivity, antisymmetry and transitivity. For any $u,v \in V$, we say that $u \leq_P v$ if $(u,v) \in \leq_P$. Note that not all elements in the set V need to be related [7,11,25].

A poset P with binary relation $\leq_P$ is said to be a strict poset, written as $P = (V, <_P)$, if the binary relation is antisymmetric and transitive but irreflexive [7,11,25]. From here on, all posets in this paper refer to strict posets.

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 $u \parallel_P v$.

Poset mining is the complete reverse of the well-researched problem that generates a complete list of linear orders, denoted as $L(P)$, based on a given poset P [6,13,19,21,22,24]. In graph theory terms, this is the well-known problem of constructing a topological sort of a given Directed Acyclic Graph (DAG) $G=(V,E)$. Poset mining basically attempts to construct a single DAG or a set of DAGs given a list of topological sorts. Here, both the poset $P=(V,<_P)$ and the DAG $G=(V,E)$ use the same set of elements V, and each binary relation $(u,v) \in <_P$ in P corresponds to an edge $(u,v) \in E$ in G [25]. The DAG G however is a transitive reduction of poset P because the relations in P are transitive. This can be then be drawn to a Hasse diagram $H(P)$.

For example, let $V=\{0,1,2,3,5,6,7,8,9\}$, and let

$$L(P)=\{(6,7,8,0,1,2,3,9,5),(6,7,8,0,2,1,3,9,5),(6,7,8,0,2,3,1,9,5),$$
$$(7,8,6,0,2,1,3,9,5),(7,8,6,0,1,2,3,9,5),(7,8,6,0,2,3,1,9,5),$$
$$(8,7,6,0,1,2,3,9,5),(8,7,6,0,2,3,1,9,5),(8,7,6,0,2,1,3,9,5)\},$$

where the nine linear orders are given in permutation notation. The resulting Hasse diagram $H(P)$ of the poset P that generates $L(P)$ is shown in Figure 1. The set $L(P)$ of linear orders is more properly called the *linear extensions* of the poset P.

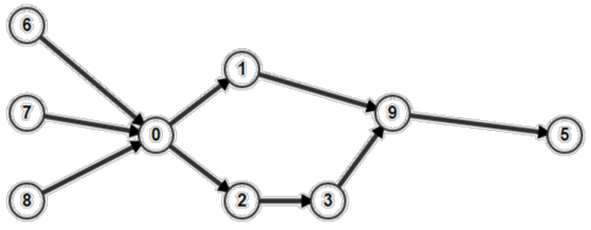

Figure 1: A Hasse diagram of the example poset

In this study, we investigate the poset mining task where exactly two posets are needed to generate a given input set of linear orders. This problem, called the 2-Poset Cover Problem, is formally defined as follows:

2-POSET COVER PROBLEM
INPUT: A set $\Upsilon = \{l_1, l_2, \ldots, l_m\}$ of linear orders over the set $V = \{1, 2, 3, \ldots, n\}$.
OUTPUT: A pair of distinct posets $P_1 = (V, \leq_{P1})$ and $P_2 = (V, \leq_{P2})$ such that $L(P_1) \cup L(P_2) = \Upsilon$ and $L(P_1) \cap L(P_2) \neq L(P_1)$ or $L(P_2)$, if such a pair exists

2. RELATED LITERATURE

There are existing works pertaining to the mining of partial orders and posets. Manilla and Meek [17] proposed a solution in discovering partial orders by viewing partial orders as generative models and used a mix of those models to describe a set of sequences. However, they restricted themselves to a particular class of partial orders known as series-parallel partial orders. They did this in order to efficiently compute the total number of extensions of a partial order without experiencing too much difficulty.

Fernandez et al. [9,10,11] and Ukkonen [26] presented polynomial-time algorithms for generating a single poset, if such a poset exists, that generates a set of linear extensions that are exactly the same as the input. Their algorithms run in $O(mn^2)$ time. Fernandez [8] extended the work and arrived at a better algorithm that runs in $O(mn+n^3)$ time. Tan [25] later optimized the solution and presented an algorithm that runs in $O(mn+n^2)$ time.

If there is no such poset that can be generated based on the input, then the problem of generating a set of posets that covers the input is explored. This problem, known as the Poset Cover Problem, has been proven by Heath and Nema [12] to be NP-complete. Therefore, Dispo-Ordanel [7], Tan [25], Fernandez [8], and Fernandez et al. [9,10,11] investigated polynomial-time solvable variants of the Poset Cover Problem by restricting their attention to specific classes of posets. By formalizing these problems, they have opened the door to greater research which may lead into more sufficient solutions and/or approximations for these problems and their variants. However, the complexity class for constrained version 2-Poset Cover Problem is still undetermined.

3. METHODOLOGY

3.1 Development of Experiment Tools

To automate the experimentation process, some basic tools were first created. The first of these is the solution verifier, a Java program framework for implementing a heuristic and for processing input test cases from an input file to generate the corresponding solutions to an output file. This program is the main tool being used to verify the correctness of the heuristics.

The general strategy used in this study for coming up with the heuristics is to begin with simple, smaller cases and then slowly work with more complex ones. Before testing a proposed heuristic against highly complex input data, it is better to guarantee first that the heuristic satisfies all basic and small test cases. To perform this, input files containing linear extensions of generated posets containing entities and relations of small sizes were generated. These input files either contained a set of arbitrary cases, or an exhaustive set of cases given a chosen, practical bound.

Two different methods were used to generate the input files. The first method was a random approach through a random poset generator. This program generates an arbitrary number of pairs of posets, each through randomly generating two distinct DAGs. This program takes in two variables, namely n (number of entities of the poset) and e (number of relations of the poset). The program then randomly generates two distinct DAGs, each containing the specified properties by maintaining a lower-triangular adjacency matrix and using a union-find disjoint set data structure.

The second method was an exhaustive approach. This was done by generating all possible pairs of posets that involve only a certain value of n and an optional value for e. Instead of randomly generating DAGs, the second approach generated all possible non-empty DAGs satisfying the specified n and e values. The input file generated by this approach was then based on the collection of the set of linear orders generated by each all possible distinct pairings.

Four different input files, corresponding to the four different sets of test cases, were used during the course of the study. Table 1 describes the properties of these input files, including the number of entities and relations of the posets involved, the generation method, and the total number of generated test cases. With the exception of the second input file, the generation method also restricted the number of relations of the expected posets to a certain value of e.

To speed up the process of the analysis, four log files were also generated. These log files contain each input instance's case number and the String representation of the two expected posets involved.

Table 1. The four input files used in the study

	Number of Entities (n)	**Number of Relations (e)**	**Generation Method**	**Number of Test Cases**
File 1	4	2	Random	100
File 2	4	0 to 4	Exhaustive	146,611
File 3	5	3	Exhaustive	441,330
File 4	6	4	Random	100,000

All output files generated by the solution verifier program contain the answers of the heuristic being verified. For each input test case being processed, the program outputs either the String representation of the posets that were generated by the heuristic, or "null" if the solution failed to generate a solution. A separate program was then created to simply count the number of test cases which contained "null" and extract the test case numbers of those cases from the output file. These represent the specific problem instances where a given heuristic fails. This data is then used to extract the counter-cases, if they exist, from the log file.

3.2 Development of Heuristics for the 2-Poset Cover Problem

In this study, we attempted to develop a polynomial-time algorithm to solve the 2-Poset Cover Problem. In the course of searching for such a solution, three heuristics were developed. The design of the proposed heuristics was done iteratively, with each succeeding heuristic addressing some discovered weaknesses from a previous heuristic.

Initially, the first heuristic was developed based on the concept of an *anchor pair* (to be discussed in Section 4). After this heuristic was tested on some random problem instances and found to fail on some instances, a second heuristic was developed to address the unsolved instances.

To challenge the correctness of the second heuristic, the test cases were expanded. This revealed a few instances where the second heuristic failed. Similar to the previous iteration, the second heuristic was then modified to address this, thus producing a third heuristic.

The third heuristic was shown to solve all instances used to test the first two heuristics. However, after the test cases were expanded further, it was observed that the third heuristic still does not solve the problem completely. Only an extreme minority (less than 0.1%) of the random test cases remain unsolved. We investigate these instances and explain why the third heuristic still fails for these. Hopefully, we can use the insights gained from this iterative development of heuristics in order to come up with better heuristics or even a polynomial-time exact solution to the 2-Poset Cover Problem.

3.3 Examining the Performance of the Heuristics

To examine the performance of the heuristics, each heuristic was implemented on the solution verifier program. To determine the accuracy of the heuristics, both the total count of the solved cases and the total count of the unsolved cases were recorded based on the results in the solution verifier's output file. As soon as one unsolved case was found, both its set of linear orders and its String representation of the two expected posets were extracted from the respective input/output files and then placed into a new data file. This data was then used to derive new insights and possible fixes for the next heuristic. If the output did not contain any unsolved cases in any of the provided input files, then the solution verifier processed input files containing complex test cases until an unsolved case was detected.

The average runtime (in milliseconds) of each heuristic was also recorded to determine the efficiency of each of them over the four input files used in the study. The average running time was based on the list of recorded execution times of the solution verifier program on a Supermicro SS1027R-WRF Server machine. Table 2 lists down all the relevant specifications.

Table 2. Supermicro SS1027R-WRF Server Specifications

Processor	2 x 2.1GHz 6-Core Intel Xeon E5-2620V2
Chipset/FSB	Intel C602
Memory	2 x 8GB DDR3
Operating System	Windows Server 2008 R2
Compiler	Java 7 Update 71

4. RESULTS AND ANALYSIS

The three heuristics developed in this study are all based on the observation that for any pair of distinct posets P_1 and P_2 over the same base set, there exists a pair of elements (a, b) such that the relationship between the two elements is different in one poset as compared to in the other poset. For example, it is possible that a is incomparable to b in one poset but comparable in the other. Another scenario is that $a<b$ in one poset but $b<a$ in the other.

We refer to such pair of elements as an *anchor pair*. The term is coined as such because we conjecture that one such pair can be used as an anchor in the partitioning of the input into two sets of linear orders. The goal of this partitioning is to use each set as a generator of a candidate (cover) poset. The generation of such a candidate poset uses the exact algorithm for solving the 1-Poset Cover (see Figure 2). This algorithm involves obtaining a candidate poset by computing the intersection of relations from the linear orders [9,10,11,26].

```
ALGORITHM: GEN_POSET
INPUT: A set $\Upsilon = \{l_1, l_2, \dots, l_m\}$ of linear orders
          over the set $V = \{1, 2, 3, \dots, n\}$.
OUTPUT: A poset $P = (V, \leq_P)$ such that $L(P_1) = \Upsilon$,
          if one exists
1   $<_P \leftarrow \cap_{L \in \Upsilon} <_L$
2   if $\Upsilon = L(P)$ then
3       return $P$
4   else return $null$
```

Figure 2: Polynomial-time algorithm to solve 1-Poset Cover [9,10,11,26]

4.1 First Heuristic

```
ALGORITHM: First Formulated Heuristic to the Two Poset
          Cover Problem
INPUT: A set $\Upsilon = \{l_1, l_2, \dots, l_m\}$ of linear orders
          over the set $V = \{1, 2, 3, \dots, n\}$.
OUTPUT: A pair of distinct posets $P_1 = (V, \leq_{P1})$ and
          $P_2 = (V, \leq_{P2})$ such that $L(P_1) \cup L(P_2) = \Upsilon$ and
          $L(P_1) \cap L(P_2) \neq L(P_1)$ or $L(P_2)$, if such a pair exists
1   for $a \leftarrow 1$ to $n - 1$
2       for $b \leftarrow a + 1$ to $n$
3           $S \leftarrow \{ l \in \Upsilon \mid a <_l b\}$
4           $S' \leftarrow \{ l \in \Upsilon \mid b <_l a\}$
5           $P_1 \leftarrow GEN\_POSET(S)$
6           $P_2 \leftarrow GEN\_POSET(S')$
7           if $P_1 \neq null$ and $P_2 \neq null$ then
8               return $\{P_1, P_2\}$
9           else if $P_1 \neq null$ then
10              $P_{2*} \leftarrow MOD\_GEN\_POSET(S', a, b)$
11              if $P_{2*} \neq null$ then
12                  return $\{P_1, P_{2*}\}$
13          else if $P_2 \neq null$ then
14              $P_{1*} \leftarrow MOD\_GEN\_POSET(S, a, b)$
15              if $P_{1*} \neq null$ then
16                  return $\{P_{1*}, P_2\}$
17  return $null$
```

Figure 3: First Formulated Heuristic for the 2-Poset Cover Problem

The first heuristic formulated by this study is stated in Figure 3. To start off, the heuristic loops through every possible pair (a, b). For every pair, the heuristic generates the two sets S and S'. The *GEN_POSET* function is then used to check if each of the two sets has a generating poset. If the heuristic succeeds in generating two valid posets P_1 and P_2, then this implies that $S = L(P_1)$ and $S' = L(P_2)$, and we have found the desired pair of posets that cover the input linear orders. If at least one of these posets is invalid, then further processing is performed.

Suppose that exactly one of the candidate posets is valid. Let the poset returned by *GEN_POSET*(S) be the valid poset. This implies that $a \parallel_{P2} b$. Since all linear orders in S' contain the relation $b <_l a$, that relation must be ignored when generating P_{2*} through the function *MOD_GENPOSET*(S', a, b), a modification of GEN_POSET that discards $b <_l a$. If poset P_{2*} is valid, then the heuristic succeeded in generating both posets.

If the heuristic fails to return two valid posets using the two possible cases of the anchor pair, then the distinct entity (a, b) is not the desired anchor pair. The heuristic exhausts all other distinct entities until it finds a suitable anchor pair that generates the two valid posets. If the heuristic fails to find any such pair of posets, then the heuristic returns *null*.

The heuristic was successful in 96/100 cases of the first input file. This indicated that there were 4 counter-examples encountered. The Hasse diagrams of the two expected posets in one such counter-example are shown in Figure 4.

Upon further analysis, it was observed that the heuristic only succeeded if the input set consists of linear orders from two posets that have no overlapping linear extensions. In all cases where there is an overlap, the heuristic incorrectly returned *null*. This is because stray linear orders (i.e., linear orders that are supposed to be also part of the other set) are contained in the set that generated the first valid poset.

To illustrate this, suppose S={(3, 1, 2, 4), (3, 1, 4, 2), (3, 4, 1, 2), (4, 3, 1, 2), (4, 1, 3, 2), (1, 4, 3, 2), (1, 3, 4, 2), (4, 1, 2, 3), (1, 4, 2, 3)} and that (1, 2) is the current pair being processed. The linear order (3, 1, 2, 4) in S is a stray linear order because only (3, 1, 2, 4) is not part of the same expected poset, which is P_2 in Fig. 4, as the other linear orders in S.

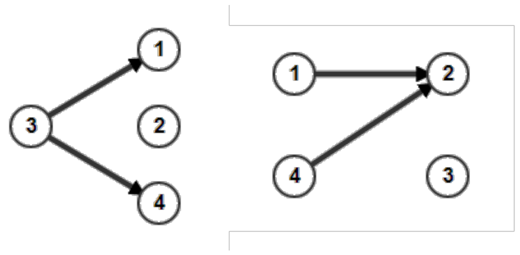

Figure 4: Hasse diagrams of the expected posets P1 and P2 in Case #26 of the first input file

4.2 Second Heuristic

As shown in Fig. 5, the second heuristic aimed to fix the mistake of the first heuristic with regards to solving the overlapping cases. Instead of applying the *MOD_GEN_POSET* function in the set that generated an invalid poset, it is instead used in the set that generated the valid poset. Suppose S is the set that generated the valid poset, and P_{1*} is the newly generated poset using *MOD_GEN_POSET*(S, a, b). If the set of linear orders of P_{1*} is a subset of ϒ, then S_{new} is generated using the difference of ϒ and $L(P_{1*})$. This is done in order to transfer the stray linear order to the proper set. P_{2*} is then generated using *GEN_POSET*(S_{new}). If P_{2*} is not *null*, and the union of the two regenerated posets P_{1*} and P_{2*} exactly matches ϒ, then the heuristic successfully generated the two valid posets.

The heuristic was successful in 98/100 cases of the input file. This indicated that there were 2 counter-examples encountered. Both pairs of Hasse diagrams of the expected posets of the two counter-examples are shown in Figures 6 and 7.

Upon further analysis, it was seen that the *MOD_GEN_POSET* function does not necessarily remove all relevant relations. Only the immediate pair (a, b) was removed, leaving the other pairs that involve the ancestors of a and the descendants of b intact. The edited function, referred to as *REDUCED_MOD_GEN_POSET*, achieves this by making the generated poset undergo transitive reduction.

ALGORITHM: Second Formulated Heuristic to the Two Poset Cover Problem
INPUT: A set $\Upsilon = \{l_1, l_2, \ldots, l_m\}$ of linear orders over the set $V = \{1, 2, 3, \ldots, n\}$.
OUTPUT: **A** pair of distinct posets $P_1 = (V, \leq_{P1})$ and $P_2 = (V, \leq_{P2})$ such that $L(P_1) \cup L(P_2) = \Upsilon$ and $L(P_1) \cap L(P_2) \neq L(P_1)$ or $L(P_2)$, if such a pair exists

```
1   for $a \leftarrow 1$ to $n-1$
2       for $b \leftarrow a+1$ to $n$
3           $S \leftarrow \{ l \in \Upsilon \mid a <_l b \}$
4           $S' \leftarrow \{ l \in \Upsilon \mid b <_l a \}$
5           $P_1 \leftarrow GEN\_POSET(S)$
6           $P_2 \leftarrow GEN\_POSET(S')$
7           if $P_1 \neq null$ and $P_2 \neq null$ then
8               return $\{P_1, P_2\}$
9           else if $P_1 \neq null$ then
10              $P_{1*} \leftarrow MOD\_GEN\_POSET(S, a, b)$
11              if $L(P_{1*}) \subseteq \Upsilon$ then
12                  $S'_{new} \leftarrow S' - L(P_{1*})$
13                  $P_{2*} \leftarrow GEN\_POSET(S'_{new})$
14                  if $P_{2*} \neq null$ and $L(P_{1*}) \cup L(P_{2*}) = \Upsilon$ then
15                      return $\{P_{1*}, P_{2*}\}$
16          else if $P_2 \neq null$ then
17              $P_{2*} \leftarrow MOD\_GEN\_POSET(S', a, b)$
18              if $L(P_{2*}) \subseteq \Upsilon$ then
19                  $S_{new} \leftarrow S - L(P_{2*})$
20                  $P_{1*} \leftarrow GEN\_POSET(S_{new})$
21                  if $P_{1*} \neq null$ and $L(P_{1*}) \cup L(P_{2*}) = \Upsilon$ then
22                      return $\{P_{1*}, P_{2*}\}$
23  return $null$
```

Figure 5: Second Formulated Heuristic for the 2-Poset Cover Problem

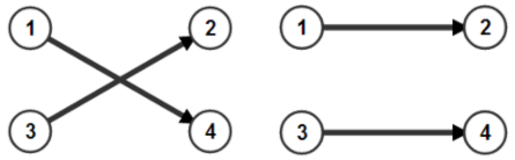

Figure 6: Hasse diagrams of the expected posets *P*1 and *P*2 in Case #39 of the first input file

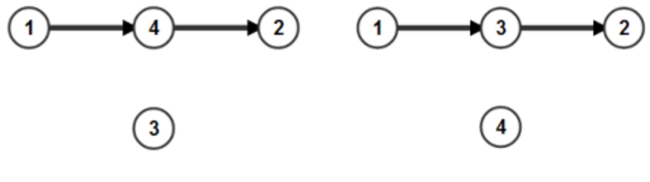

Figure 7: Hasse diagrams of the expected posets *P*1 and *P*2 in Case #54 of the first input file

4.3 Third Heuristic

The formal description of the third heuristic is provided in Fig. 8. This heuristic involves the *REDUCED_MOD_GEN_POSET* function in order to fix the mistakes of the second heuristic. However, the original poset was still generated in case the transitive reduced poset contained linear orders that are not part of Υ.

ALGORITHM: Third Formulated Heuristic to the Two Poset Cover Problem
INPUT: A set $\Upsilon = \{l_1, l_2, \ldots, l_m\}$ of linear orders over the set $V = \{1, 2, 3, \ldots, n\}$.
OUTPUT: A pair of distinct posets $P_1 = (V, \leq_{P1})$ and $P_2 = (V, \leq_{P2})$ such that $L(P_1) \cup L(P_2) = \Upsilon$ and $L(P_1) \cap L(P_2) \neq L(P_1)$ or $L(P_2)$, if such a pair exists

```
1   for $a \leftarrow 1$ to $n-1$
2       for $b \leftarrow a+1$ to $n$
3           $S \leftarrow \{ l \in \Upsilon \mid a <_l b \}$
4           $S' \leftarrow \{ l \in \Upsilon \mid b <_l a \}$
5           $P_1 \leftarrow GEN\_POSET(S)$
6           $P_2 \leftarrow GEN\_POSET(S')$
7           if $P_1 \neq null$ and $P_2 \neq null$ then
8               return $\{P_1, P_2\}$
9           else if $P_1 \neq null$ then
10              $P_{1*} \leftarrow MOD\_GEN\_POSET(S, a, b)$
11              $P_{1*r} \leftarrow REDUCED\_MOD\_GEN\_POSET(S, a, b)$
12              if $L(P_{1*r}) \subseteq \Upsilon$ then
13                  $P_{1*} \leftarrow P_{1*r}$
14              if $L(P_{1*}) \subseteq \Upsilon$ then
15                  $S'_{new} \leftarrow S' - L(P_{1*})$
16                  $P_{2*} \leftarrow GEN\_POSET(S'_{new})$
17                  if $P_{2*} \neq null$ and $L(P_{1*}) \cup L(P_{2*}) = \Upsilon$ then
18                      return $\{P_{1*}, P_{2*}\}$
19          else if $P_2 \neq null$ then
20              $P_{2*} \leftarrow MOD\_GEN\_POSET(S', a, b)$
21              $P_{2*r} \leftarrow REDUCED\_MOD\_GEN\_POSET(S', a, b)$
22              if $L(P_{1*r}) \subseteq \Upsilon$ then
23                  $P_{2*} \leftarrow P_{2*r}$
24              if $L(P_{2*}) \subseteq \Upsilon$ then
25                  $S_{new} \leftarrow S - L(P_{2*})$
26                  $P_{1*} \leftarrow GEN\_POSET(S_{new})$
27                  if $P_{1*} \neq null$ and $L(P_{1*}) \cup L(P_{2*}) = \Upsilon$ then
28                          return $\{P_{1*}, P_{2*}\}$
29  return $null$
```

Figure 8: Third Formulated Heuristic for the 2-Poset Cover Problem

The algorithm was successful in 100/100 cases of the first input file. This indicated that all counter-cases presented in the earlier sections were solved. The heuristic was also successful in all 146,611 cases of the second input file. This meant that the algorithm gave a solution to all possible distinct pairs of posets when *n = 4*.

The set of test cases was further expanded. The algorithm was then able to solve all of the 441,330 cases from the third input file, and 99948/100000 cases from the fourth input file. Figure 9 shows the Hasse Diagrams of the two expected posets of Case #1463, one of the 52 counter-examples found from the fourth input file.

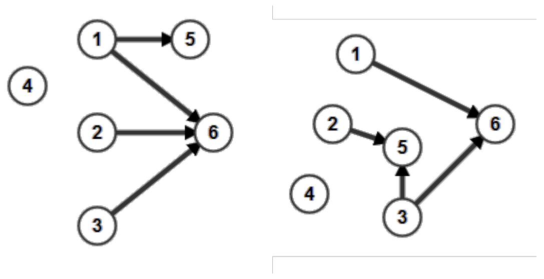

Figure 9: Hasse diagrams of the expected posets P_1 and P_2 in Case #1463 of the fourth randomly generated input file

Upon analysis of the counter-cases, it was discovered that the counter-examples can be solved if the candidate poset has undergone partial transitive reduction. This indicates that the solution might need to exhaust all possible variants which, unfortunately will need an exponential running-time complexity.

Overall, the three heuristics performed relatively well. A summary of the empirical results in terms of the accuracy and average running time (over 30 runs) is provided in Table 3 and Table 4.

Table 3. Summary of the total number of solved cases per input file of each heuristic

	First (100)	Second (146,611)	Third (441,330)	Fourth (100,000)
Heuristic 1	96.00%	99.93%	81.01%	48.68%
Heuristic 2	98.00%	99.96%	97.68%	95.13%
Heuristic 3	100.00%	100.00%	100.00%	99.95%

Table 4. Summary of average running time of each heuristic on each input file (in milliseconds)

	First	Second	Third	Fourth
Heuristic 1	4.63	3,003.80	161,156.60	394,471.47
Heuristic 2	3.70	2,212.97	76,895.43	126,670.40
Heuristic 3	2.07	2,520.23	48,445.60	63,074.47

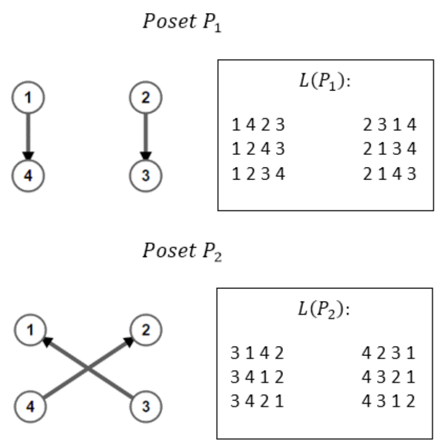

Figure 10: Hasse diagrams and the respective set of linear orders of the sample posets P_1 and P_2 of a counter-example of the anchor pair concept

In the course of the experiments, a counter-example was also discovered that disproves our conjecture about the existence, for all cases, of an anchor pair for effective partitioning of the input linear orders. As can be seen from Fig. 10, $L(P_1) \cap L(P_2) = \emptyset$ and yet there is no pair (a, b) that can properly partition the input so as to generate the two posets P_1 and P_2. Exhausting all possible pairs of elements from the base set will easily validate this claim. This shows, definitively, that the strategy of searching for a single anchor pair is not sufficient to solve the 2-Poset Cover Problem. It even seems to provide hints of a possible NP-Completeness for this problem, although much further investigation is needed to determine its correct complexity class. Nonetheless, a specific class of posets can possibly be classified, such that the presented anchor-pair strategy can be solved in polynomial time. This is subject to future studies.

5. CONCLUSION

In this study, we explore the 2-Poset Cover Problem and develop three heuristics for this. The heuristics are all based on the idea of searching for an appropriate anchor pair of elements that can be used to partition the input set of linear orders into 2 sets, and then generating a candidate poset cover for each of the sets. While the three polynomial-time heuristics do not solve the 2-Poset Cover Problem for all instances, it has been shown that these heuristics are able to return a correct 2-Poset Cover for a significant majority of the instances. More importantly, the instances on which the heuristics have failed have enabled us to have a better understanding of the problem space. The insights gained here can, hopefully, eventually lead to a polynomial-time solution to the problem, if it is indeed in the class P, or a proof that it is a NP-Complete.

6. ACKNOWLEDGMENTS

We would like to express our special thanks to the following people for providing their direct and indirect support in making this study a success: John Boaz Lee, Carlo Fransisco Adajar, Jessica Sugay, Walfrido David Diy, Jenina Isabel Sanchez, Nicko Reginio Caluya, and Ma. Mercedes Rodrigo.

7. REFERENCES

[1] Agrawal, R., and Srikant, R. 1995. Mining Sequential Patterns. In Proceedings of the Eleventh International Conference on Data Engineering, (1995), 3-14.

[2] Arkin, A., Shen, P., and Ross, J. 1997. A test case of correlation metric construction of a reaction pathway from measurements. Science 277, 5330 (1997), 1275-1279.

[3] Baker, B.S., and Coffman, E.G. 1996. Mutual Exclusion Scheduling. Theoretical Computer Science 162, 2 (1996), 225-243.

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

[5] Brightwell, G., and Winkler, P. Counting linear extensions. Order 8. 3 (1991), 225-242.

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

[7] Dispo-Ordanel, I. 2011. On two restricted cases of the poset cover problem. Master's thesis, Ateneo de Manila University.

[8] Fernandez, P. 2008. On the complexities of the block sorting and poset cover problems. PhD thesis, Ateneo de Manila University.

[9] Fernandez, P., Heath, L., Ramakrishnan, N., and Vergara, J.P. 2006. Reconstructing Partial Orders from Linear Extensions. In Proceedings of the Fourth SIGKDD Workshop on Temporal Data Mining: Network Reconstruction from Dynamic Data, (2006).

[10] Fernandez, P., Heath, L., Ramakrishnan, N., and Vergara, J.P. 2009. Mining Posets from Linear Orders. Technical Report TR -09-16, Department of Computer Science, Virginia Tech, (2009).

[11] Fernandez, P., Heath, L., Ramakrishnan, N., Tan, M. and Vergara, 2013. Mining Posets from Linear Orders. Discrete Mathematics, Algorithms and Applications 5, 4 (2013).

[12] Heath, L., and Nema, A. 2007. The Poset Cover Problem. Open Journal of Discrete Mathematics 3, 3 (2013), 101-111.

[13] Korsh, J.F., and Lafollette, P. 2002. Loopless Generation of Linear Extensions of a Poset. Order 19, 2 (2002), 115-126.

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

[15] Lee, A.K., and Wilson, M.A. 2004. A Combinatorial Method For Analyzing Sequential Firing Patterns Involving An Arbitrary Number Of Neurons Based On Relative Time Order. Journal of Neurophysiology 92, 4 (2004), 2555–2573.

[16] Mannila, H. 2008. Finding Total and Partial Orders from Data for Seriation. Lecture Notes in Computer Science 5254. 2008, 16-25.

[17] Mannila, H., and Meek, C. 2000. Global partial orders from sequential data. In Proceedings of the 6th Int'l Conf. on Knowledge Discovery and Data Mining, (2000), 161-168.

[18] Mannila, H., Toivonen, H., and Verkamo, A.I. 1997. Discovery of Frequent Episodes in Event Sequences. Data Mining and Knowledge Discovery 1, 3 (1997), 259-289.

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

[20] Patnaik, D., Butler, P., Ramakrishnan, N., Parida, L., Keller, B.J. and Hanauer, D.A. 2011. Experiences With Mining Temporal Event Sequences From Electronic Medical Records: Initial Successes And Some Challenges. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, (2011), 360-368.

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

[22] Pruesse, G., and Ruskey, F. 1994. Generating Linear Extensions Fast. SIAM Journal on Computing 23, 2 (1994), 373-386.

[23] Puolamaki, K., Fortelius, M., and Mannila, H. 2006. Seriation in paleontological data: Using Markov Chain

Monte Carlo methods. PLoS Computational Biology 2, 2 (2006).

[24] Ruskey, F. 1992. Generating Linear Extensions of Posets by Transpositions. Journal of Combinatorial Theory Series B 54, 1 (1992), 77-101.

[25] Tan, M. 2010. Polynomial-time solutions to three poset cover problem variations. Master's thesis, Ateneo de Manila University.

[26] Ukkonen, A. 2004. Data mining techniques for discovering partial orders. Master’s thesis, Helsinki University of Technology.

[27] Unnikrishnan, K.P., Ramakrishnan, N., Sastry, P.S., and Uthurusamy, R. 2006. Network Reconstruction from Dynamic Data. ACM SIGKDD Explorations Newsletter 8, 2 (2006), 90-91.

[28] West, D.B. 1993. Generating Linear Extensions by Adjacent Transpositions. Journal of Combinatorial Theory Series B 58. 1 (1993), 58-64.

[29] Wiggins, C.H., and Nemenman, I. 2003. Process pathway inference via time series analysis. Experimental Mechanics 43, 3 (2003), 361-370.Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= http://doi.acm.org/10.1145/161468.16147.