On Poset Merging[*]

Peter Chen[†] Guoli Ding[‡] Steve Seiden[†]

Abstract

We consider the follow poset merging problem: Let X and Y be two subsets of a partially ordered set S. Given complete information about the ordering within both X and Y, how many comparisons are needed to determine the order on $X \cup Y$? This problem is a natural generalization of the list merging problem. While searching in partially ordered sets and sorting partially ordered sets have been considered before, it seems that the problem of poset merging is an unexplored one. We present efficient algorithms for this problem, and show the first lower bounds. In certain cases, our algorithms use the optimal number of comparisons.

Keywords: Merging, Partial Order, Lower Bounds.
AMS Classification: 68W40

1 Introduction

Role based access control is a system of security which has generated a lot of recent interest within the database community, since it is able to model many sorts of security requirements [14].

In the role based model of Nyanchama and Osborn [13], security privileges are modeled using a directed acyclic graph called a *role graph*. Each user is a node in the graph. There is a directed path from user s to user t if and only if the privileges of s are a superset of those of t. The role graph induces a partial order on the set of users.

Database interoperability and federated databases [15] are two important current research topics in databases. A basic requirement for both interoperability and federation is that one should be able to integrate access control information [14]. The study of interoperability in role based access control was initiated by Osborn [14], who gave an algorithm for merging two role graphs. Inspired by [14], we further study the problem of merging role graphs.

Problem Description: Let S be a finite set. A subset R of $S \times S$ is called a *binary relation* on S. In this paper, we write $x \preceq y$ if $(x, y) \in R$. As usual, we call the binary relation $\preceq$ a *partial order* on S if it has the following three properties, for all $x, y, z \in S$:

(1) Reflexivity: $x \preceq x$;
(2) Antisymmetry: $x \preceq y$ and $y \preceq x$ imply $x = y$;
(3) Transitivity: $x \preceq y$ and $y \preceq z$ imply $x \preceq z$.

[*]Supported in part by NSF grant ITR-0326387.

[†]Address: Department of Computer Science, 298 Coates Hall, Louisiana State University, Baton Rouge, LA 70803. Email: `chen@bit.csc.lsu.edu` and `sseiden@acm.org`.

[‡]Address: Department of Mathematics , Louisiana State University, Baton Rouge, LA 70803-4918. Email: `ding@math.lsu.edu`.

S will be called a *partially ordered set*, or simply, a *poset*, if $\preceq$ is a partial order on S. We will write $(S;\preceq)$ if we need to specify $\preceq$. We also write $s \prec t$ if $s \preceq t$ but $s \neq t$. We use the notation $s|t$ to indicate that s and t are *incomparable*, i.e. $(s,t) \notin R$ and $(t,s) \notin R$. A poset where all pairs of elements are comparable is called a *total order*. A subset T of S where all elements are comparable is called a *chain*. A poset can be represented by a directed acyclic graph $G(S;\preceq)$, where there is a directed path from s to t in $G(S;\preceq)$ if and only if $s \preceq t$. We also define $G(T;\preceq)$ analogously for any subset T of S. A *comparison* between two elements $s,t \in S$ returns one of three distinct values, namely $s \preceq t$, $s \succ t$ or $s|t$. Suppose we have a partially ordered set S, which is partitioned into X and Y. In this paper, we examine the question "How many comparisons are required to construct $G(S;\preceq)$ given $G(X;\preceq)$ and $G(Y;\preceq)$?" We call this the *poset merging* problem. If S is totally ordered, then this is exactly the ordinary *list merging* problem.

Two closely related problems are as follows: In the *poset sorting* problem, one is given no initial information and asked to determine $G(S;\preceq)$. In the *poset searching* problem, we have a partially ordered set S and a subset T of S. We are given $G(T;\preceq)$. We wish to answer queries of the form "Does a given element $s \in S$ belong to T?" As with poset merging, in both of these problems the goal is to use the minimum number of comparisons.

Three natural ways to categorize posets are *width*, *dimension* and *number of ideals*, all measure how close a poset is to being totally ordered. We explain these measures briefly. An *antichain* of poset $(S;\preceq)$ is a subset T of S where all elements are pairwise incomparable. The width of $(S;\preceq)$ is the cardinality of the largest antichain. A well known result of Dilworth [2] tells us that any poset of width w can be partitioned into w chains. A poset has dimension d if it is the intersection of d total orders. An *ideal* in $(S;\preceq)$ is a subset T of S closed under $\preceq$. Algorithms that operate on posets often have running times bounded as some function of their size and one or more of three parameters just described.

Previous Results: To the best of our knowledge, the poset merging problem have not been previously investigated. The role-graph merging algorithm of Osborn [14] is a naïve poset merging algorithm. It requires $\sum_{i=n}^{n+m-1} i$ comparisons to merge two role-graphs (acyclic digraphs) on m and n vertices.

Next, we explore briefly what is known about the list merging problem. For a more complete treatment, we refer the reader to Section 5.3.2 of Knuth [10]. We denote the minimum number of comparisons to merge two sorted lists of lengths n and m by $M(m,n)$. As is standard in the merging literature, we always assume that $m \leq n$. The LINEAR merge algorithm [10] implies that $M(m,n) \leq n+m-1$. Knuth [10] credits Graham and Karp with independently showing that $M(m,m) \geq 2m-1$ and $M(m,m+1) \geq 2m$. Stockmeyer and Yao [16] show that $M(m,m+d) \geq 2m+d-1$ for all $m \geq 2d-2$. Therefore, LINEAR is optimal for these cases. Knuth [10] generalizes the argument of Graham and Karp to obtain lower bounds for small values of n and m. The information theoretic bound [10] states that for all n and m

$$M(m,n) \geq \left\lceil \log_2 \binom{m+n}{m} \right\rceil .$$

When $m = 1$, the information theoretic bound is tight, since binary search can be used to find the position for a single element using $\lceil \log_2(n+1) \rceil$ comparisons. Graham [6] and independently Hwang and Lin [8] show that

$$M(2,n) = \left\lceil \log_2 \tfrac{7}{12}(n+1) \right\rceil + \left\lceil \log_2 \tfrac{14}{17}(n+1) \right\rceil .$$

The first general merging procedure to outperform LINEAR was proposed by Hwang and Lin [9]. Their algorithm is a combination of LINEAR and binary search. They show that the number of comparisons it uses is at most

$$\left\lceil \log_2 \binom{m+n}{m} \right\rceil + m - 1.$$

A series of increasingly complicated and specialized algorithms have been developed by Hwang and Deutsch [7], Manacher [12], and Christen [1]. These algorithms provide improved bounds for small m.

Poset searching was first investigated by Linial and Saks [11]. They generalize binary search by defining the concept of a *central element* in a poset. By comparing the candidate element with the central element, we can eliminate a constant fraction of the ideals in the poset. They show that such elements always exist, and that therefore searching can be accomplished using $O(\log N)$ comparisons, where N is the number of ideals in the poset. Unfortunately, there is no known polynomial time algorithm for finding central elements [3].

The problem of poset sorting has also received some attention. The case where there is a total order is, of course, the standard sorting problem, which has been studied extensively [10]. Poset sorting was first studied by Faigle and Turán [4]. Let w be the width of a poset and N be the number of ideals. Faigle and Turán show a number of upper and lower bounds, the main upper bound being

$$O\left(\min\{n \log N, wn \log n\}\right)$$

on the number of comparisons required for sorting. The situation with lower bounds is not very good. If the poset is an antichain, then $\binom{n}{2}$ comparisons are required. However, if the width is w, the best lower bound on the number of comparisons is

$$n \log_3 n - n \log_3 w - 5n.$$

Further, the $O(n \log N)$ upper bound relies on the recognition of central elements [11]. Felsner, Kant, Rangan and Wagner [5] improve the lower order terms in the $O\left(wn \log n\right)$ upper bound on poset sorting.

Our Results: In this paper, we will give the first set of lower bounds and the first two non-trivial algorithms for poset merging. Our lower bounds generalize the adversary argument of Graham and Karp [10], and the information theoretic bound [10]. Our first algorithm is a generalization of the LINEAR algorithm. We will prove that this algorithm is optimal when merging partial orders consisting of two equal sized chains. Our second algorithm uses any list merging algorithm $\mathcal{A}$ as a subroutine. When $\mathcal{A}$ is the Hwang-Lin algorithm, we show that the number of comparisons used is at most $4m-3$ (recall that $m \leq n$) more than optimal when two total orders are merged. Most of our results bound the number of comparisons using some function of the widths of the posets being merged.

2 Preliminaries

For a binary relation R on S, for any two elements x and y of S, *adding* (x, y) to R or *deleting* (x, y) from R simply results in a new binary relation $R \cup \{(x, y)\}$ or $R - \{(x, y)\}$, respectively.

In this paper, the posets to be merged are always denoted by X and Y. We define $m = |X|$, $n = |Y|$ and assume that $0 < m \le n$. We define $v \le m$ and $w \le n$ to be the widths of X and Y, respectively. In this case, we always assume that X and Y are given decomposed as chains $X_1, \dots, X_v$ and $Y_1, \dots, Y_w$. If X and Y are not in this format, they can be decomposed in polynomial time without using any comparisons (all relations within X and Y are already known). Clearly, the width of $X \cup Y$ is at most $v + w$.

In general, if $(S; \preceq)$ is a poset of width w then $G(S; \preceq)$ can be represented using $O(w|S|)$ space. If $C \subset S$ is a chain in $(S; \preceq)$ and $s \in S$, then the *greatest lower bound* of s in C is an element c such that $c \preceq s$ and $c' \not\preceq s$ for all $c' \in C$ such that $c' \succ c$. If such a c exists, then it is unique. If c does not exist, we define the greatest lower bound to be $-\infty$. The *least upper bound* of s in C is the unique element c such that $s \preceq c$ and $s \not\preceq c'$ for all $c' \in C$ such that $c' \prec c$. Analogously, if c does not exist, we define the least upper bound to be ∞. Suppose $S_1, \dots, S_w$ is a decomposition of S. Each element $s \in S$ is completely defined by its w least upper bounds and w greatest lower bounds with respect to the chains $S_1 \dots S_w$. We use $2w$ pointers for each element, and w global pointers to indicate the head of each chain. We call these pointers $\mathrm{glb}(s, S_i)$ and $\mathrm{lub}(s, S_i)$ for $s \in S$ and $1 \le i \le w$.

So we can think of the problem of merging X and Y as that of correctly assigning $\mathrm{glb}(x, Y_i)$ and $\mathrm{lub}(x, Y_i)$ for all $x \in X$ and $1 \le i \le w$ and $\mathrm{glb}(y, X_i)$ and $\mathrm{lub}(y, X_i)$ for all $y \in Y$ and $1 \le i \le v$.

3 Generalizing the Linear Algorithm

We start by presenting an algorithm PLinear for merging two chains of a poset. PLinear is a natural generalization of the Linear merging algorithm and, as shown later, the number of comparisons used by PLinear is optimal when the two chains have equal size. At the end of this section, we will show how PLinear can be adapted to merge two general posets.

Suppose we are given a poset $(X \cup Y; \preceq)$, where $X = \{x_1, x_2, ..., x_m\}$ and $Y = \{y_1, y_2, ..., y_n\}$. Let us also assume that $x_i \preceq x_j$ and $y_i \preceq y_j$ for all $i \le j$. Algorithm PLinear, which is given in Figure 1 below, operates in two phases. In the first phase, we recognize all relations of the form $x_i \preceq y_j$, while in the second phase we recognize all relations of the form $y_i \preceq x_j$. Other than bookkeeping, the second phase works in the same way as the first one does.

Proposition 3.1 PLinear *is correct.*

Proof. Since X and Y are symmetric, we only need to consider Phase 1. We will show, for each i, that $x_i \preceq y_j$ if and only if the algorithm declares so. First, suppose i is the kind of index for which the algorithm detects $x_i \preceq y_j$, for some j, and also declares $x_i \preceq y_k$, for all $k \ge j$. Since $\preceq$ is transitive, the algorithm is obviously correct for all $k \ge j$. For each $k < j$, notice that when the subscript of y turns from k to $k + 1$ in the algorithm, there exists $i' \le i$ such that $x_{i'} \not\preceq y_k$. By transitivity, we must have $x_i \not\preceq y_k$, as we wanted.

Next, let i be an index such that $x_i \preceq y_j$ is never detected, for any j. There are two possible situations here, either x_i is not compared with any y_j, or x_i is compared with some y_j but $x_i \preceq y_j$ never occurred. In both cases, it is clear that the algorithm terminates because the subscript of y turns from n to $n + 1$. Consequently, there exists i' with $x_{i'} \not\preceq y_n$. Notice that $i > i'$ in the first situation and $i = i'$ in the second situation. Now by transitivity, it is easy to see that $x_i \not\preceq y_j$ for all j. ∎

Algorithm PLinear:

Phase 1. We start with $i = j = 1$ and terminate when $i > m$ or $j > n$. In a general step, we compare x_i with y_j and record the result. If $x_i \preceq y_j$, declare $x_i \preceq y_k$ for all $k \geq j$, set $i = i + 1$, and then repeat. If $x_i \not\preceq y_j$, set $j = j + 1$ and repeat.

Phase 2. We start with $i = j = 1$ and terminate when $i > n$ or $j > m$. In a general step, we compare y_i with x_j, in case these two were not compared in Phase 1 (if they have been compared in Phase 1, we use our record and this saves one comparison). If $y_i \preceq x_j$, declare $y_i \preceq x_k$ for all $k \geq j$, set $i = i + 1$, and then repeat. If $y_i \not\preceq x_j$, set $j = j + 1$ and repeat.

Figure 1: The PLinear algorithm.

In order to analyze the complexity of PLinear, we need to distinguish between comparisons that are *made* or *requested* by the algorithm. They are the same in Phase 1. But in Phase 2, when a comparison is requested, the result might be given by the record.

Proposition 3.2 *When the comparison of x_i and y_j is requested, within that phase,* PLinear *has requested, including this pair, $i + j - 1$ comparisons.*

Proof. This is clear when $i = j = 1$. Notice that, after each comparison, one of i and j is increased by one and the other remains the same. This behaves exactly the same as the formula $i + j - 1$. Thus the proposition is proved. ■

Theorem 3.1 *If $m + n > 2$, then* PLinear *makes at most $2m + 2n - 4$ comparisons when it terminates.*

Proof. Let C_1 and C_2 be the number of comparisons requested in Phases 1 and 2, respectively. Let C be the number of comparisons made in both phases and let δ be the number of comparisons requested but not made in Phase 2. Then it is clear that $C = C_1 + C_2 - \delta$.

Notice that both phases start with the request of comparing x_1 with y_1. It follows that $\delta \geq 1$. By the last proposition, it is clear that $C_1 \leq m + n - 1$ and $C_2 \leq m + n - 1$. If at least one of them holds with strict inequality, then $C \leq 2m + 2n - 4$ and we are done. Thus we may assume $C_1 = C_2 = m + n - 1$. But, by Proposition 3.2, each phase terminates only after the request of comparing x_m with y_n. Since $m + n > 2$, we have $(1, 1) \neq (m, n)$. Therefore, $\delta \geq 2$ and so $C \leq 2m + 2n - 4$, as we wanted. ■

Clearly, when $m + n \leq 2$, that is, when $m = n = 1$ (as $0 < m \leq n$), PLinear uses only one comparison, which is obviously the best possible. In fact, it will follow from Theorem 5.2 that, in the comparison model, PLinear is also optimal when $m = n \geq 2$.

Finally, we explain how to use PLinear to merge two general posets X and Y. As discussed earlier in Section 2, we assume that both X and Y are decomposed into chains $X_1, \ldots, X_v$ and $Y_1, \ldots, Y_w$, where v and w are the widths of X and Y, respectively. Now PLinear adapts easily to the general situation: We simply apply the algorithm to (X_i, Y_j) for all $1 \leq i \leq v$ and $1 \leq j \leq w$.

To count the number of comparisons used by the algorithm, we need a couple of more definitions. Let S be a poset of width k and let $S_1, \ldots, S_k$ be a partition of S into chains. The partition is *balanced* if the cardinality of $\{S_i : |S_i| = 1\}$ is minimized, over all partitions of S into k chains. We denote this minimum value by $\lambda(S)$.

Theorem 3.2 *If* PLinear *is used to merge X and Y, as suggested above, than it makes at most $2wm + 2vn - 4vw + \lambda(X)\lambda(Y)$ comparisons.*

Proof. To merge X and Y using PLinear, we need to first decompose them into chains. For the purpose of counting the number of comparisons used by the algorithm, we assume that the partitions are balanced. Since we know the complete information on both X and Y, finding such a decomposition does not cost any comparisons.

Let $m_i = |X_i|$ for $1 \leq i \leq v$ and $n_i = |Y_i|$ for $1 \leq i \leq w$. Let $\lambda(X_i, Y_j) = 1$ if $m_i = n_j = 1$, and $\lambda(X_i, Y_j) = 0$ if otherwise. Then, by Theorem 3.1, we need at most $2m_i + 2n_j - 4 + \lambda(X_i, Y_j)$ comparisons to merge X_i and Y_j. Thus, the total number of comparisons used is at most

$$\sum_{i=1}^{v}\sum_{j=1}^{w}\{2m_i + 2n_j - 4 + \lambda(X_i, Y_j)\} = 2wm + 2vn - 4vw + \lambda(X)\lambda(Y),$$

as we wanted. ■

4 Upper Bounds for Small m

The poset merging algorithm given in the previous section is based on Linear. As we will see in the next section (Theorem 5.2) that Linear is optimal for merging partial orders consisting of two equal sized chains. In this section, we develop algorithms, which perform better than Linear when merging a large chain with a small one. To facilitate this, we present a general method for applying a list merging algorithm to poset merging. Again, we examine first the case of merging two total orders.

Given any algorithm $\mathcal{A}$ for list merging, we define an algorithm PMerge$(\mathcal{A}, X, Y)$ for merging two total orders: We assume $\mathcal{A}$ uses a total order comparison operation $\trianglelefteq$ in the process of merging X and Y. The algorithm appears in Figure 2 below. As before, this algorithm can be easily generalized to merge general posets, just apply the algorithm to all pairs (X_i, Y_j) of chains.

Let $a(m, n)$ be the maximum number of comparisons used by $\mathcal{A}$ in merging a list of size m with one of size n. We say a merging algorithm $\mathcal{A}$ is *consistent* if for all m and n there exist i and j such that for all $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$ the first comparison made by $\mathcal{A}$ is $x_i \leq y_j$. Note that all merging algorithms in the literature are consistent.

Theorem 4.1 PMerge *is correct. Further, if $\mathcal{A}$ is consistent it uses at most $2a(m, n) - 1$ comparisons.*

```
Algorithm PMerge($\mathcal{A}, X, Y$):
  1. Phase($\mathcal{A}, X, Y$).
  2. Phase($\mathcal{A}, Y, X$).
```

```
Phase($\mathcal{A}, U, V$):
  1. Define $\trianglelefteq$ as follows: If $a|b$ then $a \trianglelefteq b$ if and only if $(a,b) \in U \times V$.
     If $a$ and $b$ are comparable then $a \trianglelefteq b$ if and only if $a \preceq b$.
  2. Use $\mathcal{A}$ to merge $U$ and $V$ to get list $Z = z_1, \ldots, z_{n+m}$.
  3. $z_0 \leftarrow -\infty$ and $\ell \leftarrow 0$.
  4. For $i \leftarrow 1, \ldots, n+m$:
         If $z_i \in V$ then $\ell \leftarrow i$ else $\mathrm{glb}(z_i, V) \leftarrow z_\ell$.
  5. $z_{n+m+1} \leftarrow \infty$ and $\ell \leftarrow n+m+1$.
  6. For $i \leftarrow n+m, \ldots, 1$:
         If $z_i \in U$ then $\ell \leftarrow i$ else $\mathrm{lub}(z_i, U) \leftarrow z_\ell$.
```

Figure 2: The PMerge algorithm.

Proof. During a call to Phase, the variables $\mathrm{glb}(u, V), u \in U$ are assigned in Steps 3 and 4. $\mathcal{A}$ merges U and V to get Z using the definition of $\trianglelefteq$ in Step 1 of Phase. Note that $\trianglelefteq$ is a total order on $U \cup V$. If $z_i \in U$ then $z_j \preceq z_i$ for all $j \leq i$ and $z_j \not\succeq z_i$ for all $j > i$. Define $V^* = V \cup \{-\infty\}$. It is easy to prove by induction that at each iteration of Step 4, ℓ is the maximum index $j \leq i$ such that $z_j \in V^*$. Clearly, z_ℓ is maximal in $V^* \cap \{z_1, \ldots, z_i\}$. Therefore z_ℓ is the greatest lower bound of z_i in V. Also during Phase, the variables $\mathrm{lub}(v, U), v \in V$ are assigned in Steps 5 and 6. Define $U^* = U \cup \{\infty\}$. It is easy to prove by induction that at each iteration of Step 6, ℓ is the minimum index $j \geq i$ such that $z_j \in U^*$. If $z_i \in V$ then $z_i \not\succeq z_j$ for all $j < i$ and $z_i \preceq z_j$ for all $j \geq i$. Therefore z_ℓ is minimal among $U^* \cap \{z_i, \ldots, z_{n+m}\}$ and it is the least upper bound of z_i in U.

During the two calls to Phase, all the variables $\mathrm{glb}(y, X)$, $y \in Y$; $\mathrm{lub}(x, Y)$, $x \in X$; $\mathrm{glb}(x, Y)$, $x \in X$ and $\mathrm{lub}(y, X)$, $y \in Y$ are assigned.

Obviously, PMerge uses at most $2a(m, n)$ comparisons, since it uses $\mathcal{A}$ to merge X and Y twice. If $\mathcal{A}$ is consistent, the first time we merge X and Y, we record the outcome of the first comparison and then recall it the second time we merge them. ∎

To understand how well the algorithm PMerge performs, let us define $T(m, n)$ to be the number of comparisons required to merge two total orders of sizes m and n. We point out in the following that the number of comparisons used by PMerge is not too far away from $T(m, n)$. In order to do so, we have to use not only Theorem 4.1, but also a result from the next section which tells us a lower bound on $T(m, n)$. We choose to present this result here, instead of waiting till the lower bounds have been established, because we do not like to put it too far away from the algorithm.

Corollary 4.1 PMERGE *uses at most* $T(m,n) + 4m - 3$ *comparisons.*

Proof. Let $\mathcal{A}$ be the Hwang-Lin algorithm [9], which is a generalization of both binary search and LINEAR. Then, by [9], we have

$$a(m,n) \leq \left\lceil \log_2 \binom{m+n}{m} \right\rceil + m - 1.$$

Now it follows from Theorem 4.1 that the number of comparisons used by PMERGE is at most

$$\begin{aligned}
2a(m,n) - 1 &\leq 2\left\lceil \log_2 \binom{m+n}{m} \right\rceil + 2m - 3 \\
&= 2\left\lceil \log_2 \frac{(m+n)(m-1+n)\cdots(1+n)}{m!} \right\rceil + 2m - 3 \\
&\leq 2\left\lceil \log_2 \frac{2^m(m+\lfloor n/2\rfloor)(m-1+\lfloor n/2\rfloor)\cdots(1+\lfloor n/2\rfloor)}{m!} \right\rceil + 2m - 3 \\
&= 2\left\lceil \log_2 \binom{m+\lfloor n/2\rfloor}{m} \right\rceil + 4m - 3 \\
&\leq T(m,n) + 4m - 3,
\end{aligned}$$

where the last step is by Theorem 5.3 from the next section. ■

It should be pointed out that Corollary 4.1 can be further improved by using more sophisticated algorithms for $\mathcal{A}$, for instance the algorithm of Christen [1]. It should also be emphasized again that, as with PLINEAR, PMERGE can be easily adapted to general poset merging.

5 Lower Bounds

In this section, we show the first lower bounds for poset merging. We begin by showing that if v and w are arbitrary, then nm comparisons are required. We then show lower bounds for the merging of two total orders. As we shall see, this special case turns out to be quite important. We are able to generalize these lower bounds for merging two total orders to get lower bounds which are functions of v and w.

Theorem 5.1 *If both* X *and* Y *are antichains then poset merging requires* nm *comparisons.*

Proof. We prove the theorem by using the following simple adversary argument: Each time the algorithm compares two elements, answer 'incomparable'. Suppose the algorithm outputs a candidate for $G(X \cup Y; \preceq)$ but does not compare some element $x \in X$ with some element $y \in Y$. Then $x \preceq y$ and $x|y$ are both consistent with all comparisons made, so the adversary can assign the relation between x and y in such a way that the algorithm's answer is wrong. ■

We now consider the merging of two total orders. Recall that $T(m, n)$ is the number of comparisons required to merge two total orders of sizes m and n. We start by considering $n = m$, and then move to the general case. When the total orders are the same size, we show

$$T(n,n) \geq \beta(n) = \begin{cases} n & \text{if } n = 0 \text{ or } 1 \\ 4n-4 & \text{if } n \geq 2. \end{cases}$$

Let $X = \{x_1, x_2, ..., x_m\}$ and $Y = \{y_1, y_2, ..., y_n\}$ be disjoint sets. Suppose a binary relation $\preceq$ on $X \cup Y$ satisfies the following:

$$(*) \begin{cases} \text{(i)} & x_i \preceq x_j \text{ if and only if } i \leq j; \\ \text{(i}'\text{)} & y_i \preceq y_j \text{ if and only if } i \leq j; \\ \text{(ii)} & x_i \preceq y_j \text{ implies } x_i \preceq y_{j'} \text{ for all } j' \geq j, \text{ and } x_{i'} \preceq y_j \text{ for all } i' \leq i; \\ \text{(ii}'\text{)} & y_i \preceq x_j \text{ implies } y_i \preceq x_{j'} \text{ for all } j' \geq j, \text{ and } y_{i'} \preceq x_j \text{ for all } i' \leq i; \\ \text{(iii)} & x_i \preceq y_j \preceq x_k \text{ implies } i < k; \\ \text{(iii}'\text{)} & y_i \preceq x_j \preceq y_k \text{ implies } i < k. \end{cases}$$

Then we prove a few propositions on $\preceq$.

Proposition 5.1 *The binary relation $\preceq$ is a partial order.*

Proof. First, since X and Y are disjoint, (i) and (i') directly imply reflexivity.

If $a, b \in X$ then (i) implies antisymmetry, while if $a, b \in Y$ then (i') implies it. If $a \in X$ and $b \in Y$, then $a \neq b$ since X and Y are disjoint. Suppose both $a \preceq b$ and $b \preceq a$. Condition (iii) gives us a contradiction. If $a \in Y$ and $b \in X$, the argument is analogous. Therefore, antisymmetry holds in all cases.

Now consider transitivity. Suppose $a, b, c \in X \cup Y$ with $a \preceq b \preceq c$. We need to show that $a \preceq c$. Since X and Y are symmetric, we may assume, without loss of generality, that $a = x_i$, for some i. We first consider the case $b = y_j$, for some j. If $c = y_k$, for some k, then, by (i'), $j \leq k$ and thus, by (ii), $a \preceq c$ holds. If $c = x_k$, for some k, then, by (iii), $i \leq k$ and thus, by (i), $a \preceq c$ holds. The case $b = x_j$, for some j, is similar to the last case. It follows from (i) that $i \leq j$. If $c = x_k$, for some k, then we deduce $a \preceq c$ from (i). If $c = y_k$, for some k, then we deduce $a \preceq c$ from (ii). ■

Proposition 5.2 *Suppose there are indices p and q such that $x_p \preceq y_q$ and neither $x_p \preceq y_{q-1}$ nor $x_{p+1} \preceq y_q$ holds. Then the binary relation $\preceq'$ obtained by deleting (x_p, y_q) from $\preceq$ also satisfies (*).*

Proof. It is clear that we only need to verify (ii), since all other conditions are obviously satisfied. Suppose, on the contrary, that, for some i and j, we have $x_i \preceq' y_j$ while either $x_i \not\preceq' y_{j'}$ for some $j' \geq j$ or $x_{i'} \not\preceq' y_j$ for some $i' \leq i$. By the definition of $\preceq'$ it is easy to see that $(i, j') = (p, q)$ in the first case, and $(i', j) = (p, q)$ in the second case. It follows that, in the first case, $q - 1 \geq j$ and thus $x_p \preceq y_{q-1}$ (as $\preceq$ satisfies (ii)), while in the second case, $p + 1 \leq i$ and thus $x_{p+1} \preceq y_q$ (also because $\preceq$ satisfies (ii)), both contradict our assumption. ■

Remark 5.1 *By the symmetry between X and Y, if there are indices p and q such that $y_p \preceq x_q$ and neither $y_p \preceq x_{q-1}$ nor $y_{p+1} \preceq x_q$ holds, then the binary relation obtained by deleting (y_p, x_q) from $\preceq$ also satisfies (*).*

Proposition 5.3 *Suppose $x_p \not\preceq y_q$ and $y_q \not\preceq x_p$. If both $x_p \preceq y_{q+1}$ and $x_{p-1} \preceq y_q$ hold, then the binary relation $\preceq'$ obtained by adding (x_p, y_q) to $\preceq$ also satisfies (*).*

Proof. It is clear that we only need to verify (ii), (iii), and (iii'), since all other conditions are obviously satisfied. If (iii) or (iii') is violated, then there exists either $k < p$ with $y_q \preceq x_k$ or $i > q$ with $y_i \preceq x_p$. In both cases, we deduce from (ii') that $y_q \preceq x_p$, a contradiction. Now it remains to prove (ii). Clearly, if (ii) does not hold, then we must have $(i, j) = (p, q)$. In addition, we must also have either $x_i \not\preceq y_{j'}$ for some $j' > j$ or $x_{i'} \not\preceq y_j$ for some $i' < i$. It follows that, in the first case, $q + 1 \leq j'$ and thus $x_p \not\preceq y_{q+1}$, while in the second case, $p - 1 \geq i'$ and thus $x_{p-1} \not\preceq y_q$. This contradiction completes our proof of the proposition. ■

Remark 5.2 *By the symmetry between X and Y, if $y_p \not\preceq x_q$, $x_q \not\preceq y_p$, $y_p \preceq x_{q+1}$, and $y_{p-1} \preceq x_q$ hold, then the binary relation obtained by adding (y_p, x_q) to $\preceq$ also satisfies (*).*

In the following, we consider a special poset. Let $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_n\}$, where $n \geq 2$. Let us define a binary relation $\preceq$ on $X \cup Y$ as follows:
(a) $x_i \preceq x_j$ and $y_i \leq y_j$ for all $i \leq j$;
(b) $x_i \preceq y_j$ if and only if $j - i \geq 2$;
(c) $y_i \leq x_j$ if and only if $j - i \geq 1$.
It is not difficult to see that $\preceq$ satisfies (*) and thus, by Proposition 5.1, $X \cup Y$ is partially ordered by $\preceq$.

Let $R_1 = \{(x_i, y_{i+2}) : i = 1, 2, ..., n-2\}$, $R_2 = \{(y_i, x_{i+1}) : i = 1, 2, ..., n-1\}$, $R_3 = \{(x_i, y_{i+1}) : i = 1, 2, ..., n-1\}$, and $R_4 = \{(y_i, x_i) : i = 1, 2, ..., n\}$.

Proposition 5.4 *If $(a, b) \in R_1 \cup R_2$, then deleting (a, b) from the partial order $\preceq$ results in a new partial order.*

Proof. It is straightforward to verify that the assumption here satisfies the assumptions of Proposition 5.2 or Remark 5.1. Then the result follows immediately from Proposition 5.1. ■

Proposition 5.5 *If $(a, b) \in R_3 \cup R_4$, then adding (a, b) to the partial order $\preceq$ results in a new partial order.*

Proof. This is very similar to the last proof. It is straightforward to verify that the assumption here satisfies the assumptions of Proposition 5.3 or Remark 5.2. Then the result follows immediately from Proposition 5.1. ■

Theorem 5.2 *The number of comparisons needed to recognize $(X \cup Y; \preceq)$ is at least $\beta(n)$.*

Proof. If an algorithm has made less than $\beta(n)$ comparisons, then for some $(a, b) \in R_1 \cup R_2 \cup R_3 \cup R_4$, a and b are not compared by the algorithm, as $|R_1 \cup R_2 \cup R_3 \cup R_4| = \beta(n)$. However, by Propositions 5.4 and 5.5, there is another partial order $\preceq'$ that agrees with $\preceq$ everywhere else, except for (a, b). Thus the algorithm cannot determine if the partial order is $\preceq$ or $\preceq'$. ■

As mentioned earlier in Section 3, by combining Theorem 3.1 and Theorem 5.2 we conclude that PLinear is optimal when merging two total orders of the same size. Theorem 5.2 also generalizes easily as follows: Suppose $n/w = m/v = k$ is integral. Then a lower bound for merging X and Y is $vw\beta(k)$. To see this, consider the case where $n_i = k$ for all $1 \le i \le w$ and $m_j = k$ for all $1 \le j \le v$. For all $x, x' \in X$, $x|x'$ if and only $x \in X_i$, $x' \in X_j$ with $i \neq j$. In other words, only elements within a chain are comparable in X. The same condition holds for Y. Then, for all pairs (i, j) and (k, ℓ), it is not difficult to see that X_i merges with Y_j independently of how X_k merges with Y_ℓ, since $G(X_i \cup Y_j; \preceq)$ and $G(X_k \cup Y_l; \preceq)$ are independent.

We now give a general lower bound for merging two total orders:

Theorem 5.3 *For all positive integers m and n,*

$$T(m,n) \ge \left\lceil \log_2 \binom{\lceil n/2 \rceil + m}{m} \right\rceil + \left\lceil \log_2 \binom{\lfloor n/2 \rfloor + m}{m} \right\rceil .$$

Proof. The proof is a variation on the information-theoretic lower bound for merging. Let X and Y be total orders with $|X| = n$ and $|Y| = m$. Let $k = \lfloor n/2 \rfloor$. We consider only instances X, Y where $\mathrm{glb}(y, X) \preceq x_k$ and $x_{k+1} \preceq \mathrm{lub}(y, X)$ for all $y \in Y$. We further require that for all $y, y' \in Y$ such that $y \preceq y'$, $\mathrm{glb}(y, X) \preceq \mathrm{glb}(y', X)$ and $\mathrm{lub}(y, X) \preceq \mathrm{lub}(y', X)$. It is easy to verify that the conditions (*) hold in this case, so we have a partial order. For instances of this type, merging X and Y involves solving two totally independent sub-problems. The first is to determine $\mathrm{glb}(y, X)$ for all $y \in Y$. The number of possible solutions to this sub-problem is $s = \binom{\lfloor n/2 \rfloor + m}{m}$. Note that in solving this sub-problem, the three outcomes of a comparison really only give us one bit of information. That is, when comparing an element $y \in Y$ with x_i for $i \le k$, we never get the answer $y \preceq x_i$. Therefore, the number of comparisons needed is at least $\lceil \log_2 s \rceil$. A similar argument holds for the sub-problem of determining $\mathrm{lub}(y, X)$ for all $y \in Y$. ■

6 Summary and Future Research Directions

Poset merging is an important but understudied research area. In this paper, we have presented the first non-trivial upper bounds, and the first lower bounds for this problem. We proved that our upper bounds are not too far away from our lower bounds, and in some cases, these two bounds are actually equal. In our model, we do not impose any condition on the union of the two posets to be merged. It will be very interesting to see whether tighter bounds exist if we do know some extra information on the union. In particular, the following problem is still open: Suppose we are merging X and Y and we are given that the width of $X \cup Y$ is $k < v + w$. How can we use this information to reduce the number of comparisons? For instance, for $n = m$ and $v = w = 1$ if $k = 2$ we require at least $2m + 2m - 4$ comparisons, while if $k = 1$ we need at most $n + m - 1$.

Acknowledgment. We are grateful to the two anonymous referees for carefully reading our paper and for helping us improving the paper.

References

[1] Christen, C. Improving the bounds on optimal merging. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science* (Oct. 1978), pp. 259–266.

[2] Dilworth, R. P. A decomposition theorem for partially ordered sets. *Annals of Mathematics 51* (1950), 161–166.

[3] Dubhashi, D. P., Mehlhorn, K., Ranjan, D., and Thiel, C. Searching, sorting and randomised algorithms for central elements and ideal counting in posets. In *Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science* (Dec. 1993), pp. 436–443.

[4] Faigle, U., and Turán, G. Sorting and recognition problems for ordered sets. *SIAM Journal on Computing 17*, 1 (Feb. 1988), 100–113.

[5] Felsner, S., Kant, R., Rangan, C., and Wagner, D. On the complexity of partial order properties. *ORDER: A Journal on the Theory of Ordered Sets and Its Applications 17*, 2 (2000), 179–193.

[6] Graham, R. L. On sorting by comparisons. In *Computers in Number Theory: Proceedings of the Science Research Council Atlas Symposium no. 2* (Aug. 1969), pp. 263–269.

[7] Hwang, F. K., and Deutsch, D. N. A class of merging algorithms. *Journal of the ACM 20*, 1 (Jan. 1973), 148–159.

[8] Hwang, F. K., and Lin, S. Optimal merging of 2 elements with n elements. *Acta Informatica 1*, 2 (1971), 145–158.

[9] Hwang, F. K., and Lin, S. A simple algorithm for merging two disjoint linearly ordered sets. *SIAM Journal on Computing 1*, 1 (Mar. 1972), 31–39.

[10] Knuth, D. E. *Sorting and Searching*, second ed., vol. 3 of *The Art of Computer Programming.* Addison-Wesley, Reading, MA, USA, 1998.

[11] Linial, N., and Saks, M. Every poset has a central element. *Journal of Combinatorial Theory, Series A 40* (1985), 195–210.

[12] Manacher, G. K. Significant improvements to the Hwang-Lin merging algorithms. *Journal of the ACM 26*, 3 (July 1979), 434–440.

[13] Nyanchama, M., and Osborn, S. The role graph model and conflict of interest. *ACM Transactions on Information and Systems Security 2*, 1 (1999), 3–33.

[14] Osborn, S. Database security integration using role-based access control. In *Proceedings of the IFIP WG11.3 Working Conference on Database Security* (Aug. 2000), pp. 423–437.

[15] Sheth, A. P., and Larson, J. A. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys 22*, 3 (1990), 183–236.

[16] Stockmeyer, P. K., and Yao, F. F. On the optimality of linear merge. *SIAM Journal on Computing 9*, 1 (1980), 85–90.