# Dynamic Implicit Strict Partially Ordered Sets

Andrew Chen
Department of Computer Science and Information Systems
College of Business and Industry
Minnesota State University Moorhead
Moorhead, Minnesota 56563
USA
chenan@mnstate.edu

***ABSTRACT:*** *Sets may be ordered, unordered, or partially ordered. Amongst partially ordered sets (posets), if the relation between the ordered elements is one of* $\geq$ *then this is a standard poset, but if the relation between the ordered elements is one of* $>$*then this is called a strict poset. It is the author's contention that many strict posets exist implicitly: for example, most examples of preference ordering are actually only partial orders, since one may easily rank one's most-favorite and least-favorite of anything, but of all that is inbetween, there has probably not been formed a complete total ordering. Furthermore, even if there was a complete total ordering, the solicitation of the preference order information often results in only a partial order being known: for example, it is easier to ask "which of these two do you like better" than it is to ask "please rank all of these from most favorite to least favorite" for large sets. Additionally, preferences change over time, and as a result, finding and identifying which piece of ordering information is now invalid is a challenge for a variety of reasons. This paper seeks to identify some initial possibilities for remedying and meeting that challenge.*

**Keywords:** Poset, Partially Ordered Set, Dynamic, Strict Partially Ordered Set

**Received:** 14 June 2011, Revised 17 July 2011, Accepted 21 July 2011



## 1. Introduction

In this paper, we introduce the phenomena of dynamic implicit strict partially ordered sets. After defining them and the primary problem associated with them that we are interested in, we provide some preliminary results into the investigation of this problem.

### 1.1 Definitions

Let $S$ be a set.

We say that S is a partially ordered set with respect to a relation $r$ if $r$ defines a (possibly empty) ordering between some elements of S but possibly not between other elements of $S$. We will be refering to relations as sets as well, since they are sets of the ordered pairs for which the relation holds, and thus we will be using relations in set operations as well.

For an example of a total ordering, the set of integers with respect to the relation >is a total ordering, since between any two different integers $x$ and $y$, either $x > y$ or $y > x$.

For an example of an unordered set, consider the set of colors with respect to the relation "earlier". Since we can not say that "red is earlier than green" we must conclude that the colors are unordered with respect to this relation.

For an example of a partially ordered set, consider what happens when one takes a deck of playing cards and tosses them into

a messy pile in the middle of a table. Consider the relation "is on top of" with respect to the playing cards in that messy pile. For some cards, we may be able to say "this card is on top of that card", but for other cards, we may not be able to say that. Thus, this is an example of a partially ordered set.

Amongst partially ordered sets (sometimes called *posets*), the relation between the ordered elements is often one of $\geq$. When the relation between the ordered elements is one of > then this is called a *strict* poset.

For more coverage of posets, see [1].

In a complete total ordering, the solicitation of the preference order information often results in only a partial order being known: for example, it is easier to ask "which of these two do you like better" than it is to ask "please rank all of these from most favorite to least favorite" for large sets. One may easily rank one's most-favorite and least-favorite of anything, but of all that is in-between, there has probably not been formed a complete total ordering. Thus, most examples of preference ordering are actually only partial orders, and given the many forms of preference ordering, we can see that many strict posets exist implicitly. These two characteristics, that the solicitation of the ordering information within these strict posets will only involve partial gathering of the ordering relations, and that these strict posets exist implicitly, are the two characteristics that the author will use to define an *implicit* strict partially ordered set.

Additionally, preferences change over time, and as a result, finding and identifying which piece of ordering information is now invalid is a challenge for a variety of reasons, the main one of which is that there is lack of knowledge of which piece of ordering information changed due to the fact that there is only partial gathering of the ordering relations. This paper seeks to identify some initial possibilities for meeting that challenge.

**1.2 Problem Statement**

At time $t$, $r(t)$ defines a total ordering on set $S$. At time $t$, we are randomly given $x \in r(t)$ through a function $f(t)$. At time $t+1$, $r(t+1)$ now may differ from $r(t)$ through some small adjustment such that $|r(t) \cup r(t+1)| \leq k$ where < is the symmetric difference and $k$ is some small positive integer constant. Note that we may have $|r(t) \cup r(t+1)| = 0$

The problem is to create the largest and best possible approximation of $r(t)$ given the orderings that f(t) has given us, for each t from when we began ($t=0$) until now ($t=t_{now}$).

(Note that if $k=0$ then the solution is trivial, since one would need to merely wait long enough before $f(t)$ would give us all of the elements of an unchanging r(t). Thus, we must assume $k \geq 1$.)

Before finding a largest and best possible approximation of $r(t)$, let us define more precisely what we mean by "largest" and "best".

Let us denote our approximation to $r(t)$ by $a(t)$.

By largest we mean with $|a(t)|$ maximal. Note that $|a(t)|$ is bounded by $|r(t)|$ if we know $|r(t)|$, and if we don't know $|r(t)|$, then $|a(t)|$ is bounded by the upper bound on $|r(t)|$ that we can infer from $|S|$. By best we mean two things, namely, $|a(t) \cap r(t)|$ being maximal and $|a(t) \cup r(t)|$ being minimal, where $\cap$ is intersection.

**2. Hypothesis**

If we model our knowledge of $S$ and $a(t)$with a directed graph $G$ in which each element of S is a vertex, and each ordering between two elements becomes a directed edge, then at each time t we are given another directed edge to add to $G$, and we can keep on doing this until such time as we obtain a directed cycle. Clearly, a directed cycle is an indication that there is some out-of-date information present. In response to encountering a directed cycle, one reasonable conclusion might be to discard the oldest piece of ordering information that we have that contributes to this cycle. To do this, it means that we must label each directed edge with the time t at which we added it to our graph $G$. Note that each edge will have a unique label as a result of the fact that we add the edges to the graph one-at-a-time.

Given that we have enountered a directed cycle that was formed by the addition of the most recent edge that we have encountered, finding the oldest piece of ordering information within that cycle would merely involve traversing the cycle. Unfortunately, however, it is entirely possible that we might have multiple cycles that all overlap on the new directed edge

that was just added. These multiple cycles might overlap in multiple other ways as well, resulting in a potential combinatorial explosion if we were to try to traverse all such possible cycles. Thus, we need something more efficient than traversing all cycles that involve the new edge that was just added.

Let us say that the newest edge was $e = (t, s)$. Since this completes a cycle, this means that there is at least one (possibly many) paths from $s$ to $t$ within the $G$ that corresponds to $a(t)$. Our goal is to find a way to cut the oldest edge in the paths from $s$ to $t$.

One approach that comes to mind would be to consider the subgraph $B$ induced by the paths from s to t and to apply a *min -cut* [2] algorithm to this, using the time labels of the edges as weights. Unfortunately, this yields one problem: we don't necessarily want the minimum cut, rather we want the cut for which the maximum value is minimum. In other words, a small cut that has one newer element is worse than a larger cut that consists of a large number of older elements. (This approach favors the "largest" criterion over the "best" criterion, and this problem that we mention with regard to this approach is that perhaps we want to value the "best" criterion over the "largest" criterion.)

Taking this into consideration, we consider another approach: let us consider $B'$. where $B'$. is like $B$ but with the following two modifications: (1) add in a new vertex $v$ and edges connecting $v$ to s and $v$ to $t$ and (2) also make this an undirected graph. If we are to focus on the goal of "best" then we can start with the tree formed by the two edges $(v, s)$and $(v, t)$and then greedily add new (greatest timestamp) edges to that tree that would not form a cycle so as to form a maximum spanning tree of $B'$. (this can be done by taking many minimum spanning tree algorithms and merely modifying them to be maximum spanning tree algorithms instead [2]) (maximum by sum of tim estamps). Take this maximum spanning tree of $B'$. and remove $v$ and its incident edges and the resulting graph will have a cut $C$ between the component of the tree that includes $s$ and the component of the tree that includes $t$. The edges on this cut $C$ will be the oldest edges (if there was another cut with older edges, then one of the edges in $C$ would have been chosen to be part of the maximum spanning tree instead because of the fact that the algorithm for choosing edges was greedy) which is precisely what will satisfy our desire for the "best" criterion.

It is our hypothesis that the spanning tree approach described in the previous paragraph is better (according to the "best" criterion described above) and that the *min-cut* approach described prior to that is "larger" according to the "largest" criterion described above.

## 3. Experiment

To test the previously mentioned hypothesis, these two approaches were implemented in a computer program (contact author for further details) and then run on random permutations with random neighboring elements in those permutations being swapped every time unit (and thus $k = 2$).

## 4. Implementation

The implementation of the *min-cut* approach used the probablistic approach given in [3]. The implementation of the spanning tree approach used a greedy approach that added edges in order of increasing age and ignored those that would cause cycles (contradictions in ordering information).

## 5. Results

For initial total orderings of 100 elements, with permutations occurring every time unit, and running these approaches for 101 time units (time stamps ranging 0 to 100 inclusive), both algorithms yielded the same results every time, for ten times, with the results as given in Table 1.

Each time $r$ and $f$ were determined, and then the two algorithms were run with $f$ as their input.

## 6. Conclusion

As was mentioned, there was no noticeable difference between the two algorithms of seeking the "best" versus the "largest". It is possible that this may be shown with larger data sets, and especially those for which there are multiple possible cuts, given that the *min-cut* approach that was chosen was a probabilistic approach.

| $\|a\|$ | $\|a \cap r\|$ | $\|a\text{-}r\|$ |
|---|---|---|
| 100 | 97 | 3 |
| 100 | 100 | 0 |
| 100 | 98 | 2 |
| 101 | 101 | 0 |
| 101 | 101 | 0 |
| 101 | 100 | 1 |
| 99 | 99 | 0 |
| 101 | 101 | 0 |
| 100 | 100 | 0 |
| 101 | 99 | 2 |

Table 1. Results of Experiment

These two algorithms were largely able to get an approximation that was rather good, sometimes have no incorrect information (quite possibly because *f* may not have sampled from any information that changed over time), and sometimes having only a very small amount of incorrect information (3 out of a 100).

Futhermore, as you can see, it wasn't often that contradictions were discovered and eliminated in these circumstances. If no contradictions were discovered, then $|a|$ is the number of timestamps involved, which was the case in half of the trial runs. The greatest number of edges needing to be removed in order to resolve contradictions was 2 as can be seen in the fact that one run had $|a| = 99$.

**7. Future Work**

Having identified the problem and begun to find possible solutions to this problem, our future work would include, not necessarily in this order:

• determining the likelihood that directed cycles formed as outlined in this paper will have multiple cuts

• determining a metric for measuring the amount of order within a strict poset

• determining appropriate data structures with which to represent $a(t)$so as to have minimum complexity with regard to the addition of a new edge with the passing of time

• developing a robust reference implementation that uses those data structures

• given the data structures that would be suitable, identifying the best way to parallelize the appropriate algorithms to work with those data structures

• developing a reference parallel implementation

**References**

[1] Peter, J., Cameron (1994). *Combinatorics: Topics, Techniques, Algorithms*. Cambridge University Press.
[2] Gary Chartrand and Ortrud, R., Oellermann. (1993). Applied and Algorithmic Graph Theory. McGraw-Hill, *Inc*.
[3] David, R., Karger and Clifford Stein (1996). A new approach to the minimum cut problem. *Journal of the ACM*, 43:601–640, July.

**Author Biography**

Andrew Chen obtained his B. S. degree in Computer Science and Mathematics in 1997 at the University of Richmond in Richmond, Virginia. He then went on to get his M. S. (1999) and Ph. D. (2005) from Michigan State University. He has been working as an assistant and now associate professor at Minnesota State University Moorhead since then. His research interests involve novel applications of graph theory.