Lattice Completion Algorithms for Distributed Computations

Vijay K. Garg*

Parallel and Distributed Systems Lab,
Department of Electrical and Computer Engineering,
The University of Texas at Austin,
Austin, TX 78712
garg@ece.utexas.edu http://www.ece.utexas.edu/~garg

Abstract. A distributed computation is usually modeled as a finite partially ordered set (poset) of events. Many operations on this poset require computing meets and joins of subsets of events. The lattice of normal cuts of a poset is the smallest lattice that embeds the poset such that all meets and joins are defined. In this paper, we propose new algorithms to construct or enumerate the lattice of normal cuts. Our algorithms are designed for distributed computing applications and have lower time or space complexity than existing algorithms. We also show applications of this lattice to the problems in distributed computing such as finding the extremal events and detecting global predicates.

1 Introduction

A distributed computation is usually modeled as a set of events ordered by the partial order relation called the happened-before [Lam78] relation. This relation can be tracked using Mattern [Mat89] and Fidge's vector clocks [Fid89] which provide an efficient implicit representation of the poset of events that happened in a distributed computation. There are numerous applications in distributed systems such as distributed debugging [CM91,GW94], and recovery of distributed programs [SY85], that track the happened-before relation using vector clocks.

Since the joins and meets are always defined for lattices but may not exist for a general poset, there are many fundamental and practical advantages of working with lattices rather than posets. Given any poset, there are usually two ways to complete it — completion by consistent cuts (or ideals) and completion by normal cuts. The lattice of consistent cuts captures the notion of consistent global states in a distributed computation and has been discussed extensively in the distributed computing literature [Mat89,CM91,GM01]. The lattice of normal

* supported in part by the NSF Grants CNS-1115808, CNS-0718990, CNS-0509024, and Cullen Trust for Higher Education Endowed Professorship.

cuts has not received much attention in distributed computing. For a poset P, its completion by normal cuts, or Dedekind-Macneille (DM) completion, denoted by $L_{DM}(P)$ is the smallest lattice that has P as its suborder [DP90]. Fig. 1(i) shows a distributed computation with four events. Its completion by normal cuts and consistent cuts is shown in Fig. 1(ii) and (iii), respectively.

The lattice of normal cuts is generally much smaller in size than the lattice of consistent cuts. In the extreme case, the lattice of consistent cuts may be exponentially bigger in size than the lattice of normal cuts. We show in this paper that some global predicates can be detected on the lattice of normal cuts instead of consistent cuts, thereby providing an exponential reduction in the complexity of detecting them.

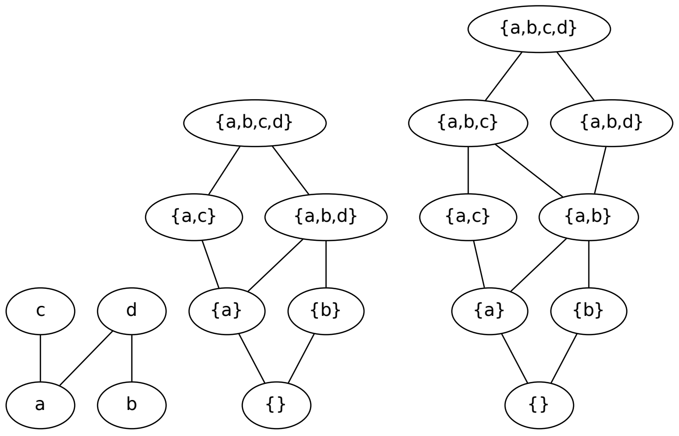

Fig. 1: (i) The original poset. (ii) Its lattice of normal cuts (iii) Its lattice of consistent cuts.

In this paper, we also discuss algorithms for constructing and enumerating $L_{DM}(P)$ for a distributed computation given as a finite poset P with implicit representation (i.e., represented using vector clocks). There has been extensive research in algorithms for the problems of DM-completion [NR99,NR02,GK98], construction of concept lattices[Gan84], construction of maximal antichain lattice [JRJ94], and construction of union-closed family of sets [NR99,NR02]. Our work differs in principally two ways. First, our focus is on implicit representation of posets and lattices. Most of the earlier work builds explicit cover relation of the lattice, whereas we represent the lattice implicitly using vector clocks. We note here that [Gar13] also uses vector clocks but for the lattice of maximal antichains. Second, our work is targeted towards distributed computing traces. For distributed computing traces, it is natural to assume that the number of events generated by a single process is significantly more than the number of processes, i.e., the width of the poset is much smaller than the height of the poset corresponding to the computation. Also, most events in a distributed computation are internal to the process, i.e., they do not have any interaction with other processes. The computational complexity of our algorithm is explicitly dependent on the width of the poset, and the number of message receive events in a distributed system.

There are principally two classes of algorithms for generation of lattices. Incremental algorithms take as input a poset P and its lattice completion L, and output the lattice completion of the poset P extended with an element x. The algorithms by Ganter and Kuznetsov [GK98] and Lourine and Raynaud [NR99,NR02] fall in this class. These algorithms store the entire lattice. The other class of algorithms, frequently used in concept analysis [GW97], only require enumeration of all elements of the concept lattice. They do not require storage of the entire lattice (which may be exponentially bigger than the poset itself). The algorithm by Ganter [Gan84] falls in this class. It enumerates all elements of the lattice in a lexicographical order. To distinguish between these two classes of lattice generation, we refer to the first class of algorithms as the lattice *construction* and the second class of algorithms as the lattice *enumeration*. In this paper, we propose algorithms for both lattice construction and lattice enumeration adapted to distributed computations.

Table 1: Algorithms for Lattice Construction and Enumeration of Normal Cuts.

Algorithm	Incremental	Time Complexity	Space Complexity
Ganter and Kuznetsov [GK98]	Yes	$O(mn^3)$	$O(mn)$
Nourine and Raynaud[NR99,NR02]	Yes	$O(mn^2)$	$O(mn)$
Algorithm IDML [this paper]	Yes	$O(rwm \log m)$	$O(mw \log n)$
BFS [this paper]	No	$O(mw^2(w + \log w_L))$	$O(w_L w \log n)$
DFS [this paper]	No	$O(mw^3)$	$O(h_L w \log n)$
Lexical by Ganter [Gan84]	No	$O(mn^3)$	$O(n)$

Table 2: The notation used in the paper

Symbol	Definition	Symbol	Definition
n	size of the poset P	m	size of the normal cuts lattice L
w	width of the poset P	r	number of elements with more than one lower cover
h_L	height of the lattice L	w_L	width of the lattice L.

We first propose an incremental Dedekind-Macneille lattice construction algorithm called IDML which compares favorably with the algorithms proposed by Nourine and Raynoud [NR99,NR02] for distributed computing. Let the size of the poset be n and the size of the DM-lattice be m, then the algorithm by Nourine and Raynoud takes $O(n^2m)$ time. The IDML algorithm takes $O(rwm \log m)$ time where w is the width of the poset and r is the number of receive events in the computation. For typical distributed computations, our algorithm has significantly smaller time complexity. Moreover, Nourine and Raynoud's algorithm require building a special structure called a lexicographic tree with space complexity $O(mn)$. Our incremental algorithm uses a balanced binary search tree of all the lattice elements with the space complexity $O(mw \log n)$.

For lattice enumeration, the existing algorithms use *lexicographical* enumeration of the lattice [Gan84]. In this paper, we propose techniques for breadth-first (BFS) and depth-first (DFS) enumeration of lattices. It is important to note that the algorithms for BFS and DFS enumeration of lattices are different from the standard graph-based BFS and DFS enumeration because our algorithms cannot store the explicit graph corresponding to the lattice. Hence, the usual technique of marking the visited nodes is not applicable. BFS-enumeration and DFS-enumeration may be semantically more meaningful and useful in distributed computing than lexical enumeration. For example, while searching for an event with a given property in a distributed computation, it is more useful to find one at the lowest level of the lattice. Note that BFS, DFS and lexical algorithms for enumeration of the lattice of consistent cuts (but not for the lattice of normal cuts) have already been proposed in the distributed computing literature. For example, BFS enumeration has been proposed by Cooper and Marzullo [CM91], DFS enumeration by Alagar and Venkatesan [AV01], and Lexical enumeration by Garg [Gar03]. Due to different structure of these lattices, the technique for BFS and DFS enumeration is quite different. For example, the problem of determining if an element of the lattice has already been enumerated is different for the two lattices. Table 1 summarizes the time and space complexity of the lattice construction and enumeration algorithms.

The ability to construct or enumerate the lattice of normal cuts has wide applications in many areas. We discuss distributed computing applications in Section 6. It has applications in other areas such as formal concept analysis [GW97] but will not be discussed in this paper.

2 Background: Posets with Implicit Representation

We assume that the reader is familiar with the basic concepts of posets and lattices [DP90]. A partially ordered set (or *poset*) is a pair $P = (X, \leq)$ where X is a set and $\leq$ is a reflexive, antisymmetric, and transitive binary relation on X. A *subposet* of P is a subset of X whose order relation is restriction of P to the subset. If either $x \leq y$ or $y \leq x$, we say that x and y are *comparable*; otherwise, we say x and y are *incomparable*. For any two elements x and y, y covers x if $x < y$ and $\forall z \in X : x \leq z < y$ implies $z = x$. A subset $Y \subseteq X$ is called an *antichain (chain)*, if every distinct pair of points from Y is incomparable (comparable) in P. The *width (height)* of a poset is defined to be the size of a largest antichain (chain) in the poset.

Given a subset $Y \subseteq X$, the *meet* of Y, if it exists, is the greatest lower bound of Y and the *join* of Y is the least upper bound. An element is *join-irreducible (meet-irreducible)* if it cannot be expressed as the join (meet) of other elements. A poset $P = (X, \leq)$ is a *lattice* if joins and meets exist for all finite subsets of X. It is a *complete lattice* if joins and meets exists for all subsets of X. The largest element of a lattice is called the *top* element.

Let P be a poset with a given chain partition of width w. In a distributed computation, P is the set of events executed under the happened-before relation where a chain corresponds to a total order of events executed on a single process. In such a poset, every element e can be identified with a tuple (i, k), the k^{th} event on the i^{th} chain. In this paper, we keep the order relation implicit using vector clock [Mat89] as explained next. For $e \in P$, let $D[e]$, the *down-set* of e be the elements in P, that are less than or equal to e. The set $D[e]$ can equivalently be captured using a vector $e.V$ such that $e.V[i] = j$ iff there are exactly j elements on chain i that are less than or equal to e. It is easy to verify that $e \leq f$ iff $e.V \leq f.V$. Fig. 2(i) and (ii) show a poset and corresponding vector clocks.

A subset Q is a consistent cut (an order ideal) of P if it satisfies the constraint that if f is in Q and e is less than or equal to f, then e is also in Q. For any element $e \in P$, $D[e]$ is always a consistent cut and is called a *principal ideal.* Any consistent cut Q of P can be represented using a simple vector $Q.V$ with the interpretation that $Q.V[i] = j$ iff exactly j smallest elements of chain i are in Q. Note that we have used vectors for representing events as well as set of events. Given two consistent cuts Q and R, their intersection (union) is simply the component-wise minimum (maximum) of the vectors for Q and R.

Just as we have constructed vectors using the down-sets, we can also use the dual *up-sets*. For $e \in P$, let $U[e]$, the *up-set* of e be the elements in P, that are greater than or equal to e. The notion of order filters which are duals of order ideals can similarly be defined.

3 Lattice Completion of a Computation

In this section, we discuss lattice-completion of a computation via normal cuts. Given $Q \subseteq P$, the set of lower bounds of Q, denoted by Q^l is given by

$$\{x \in P | \forall e \in Q : x \leq e\}$$

In Fig. 1, $\{c, d\}^l = \{a\}$. When Q is empty, Q^l is trivially the entire set P. When Q is singleton $\{e\}$, then it simply corresponds to $D[e]$. In general, we can compute Q^l using D as follows:

$$Q^l = \cap_{e \in Q} D[e]$$

It is easy to verify that Q^l is alway a consistent cut because $D[e]$ is a consistent cut for any e, and consistent cuts are closed under intersection. Similarly, the set of upper bounds of Q, denoted by Q^u can be computed. In Fig. 1, $\{a, b\}^u = \{d\}$. The set $(\{a, b\}^u)^l = \{a, b, d\}$.

Definition 1 (Normal Cut). *[DP90] A set $Q \subseteq P$ is a* normal cut *if* $(Q^u)^l = Q$.

We will use the simpler notation Q^{ul} for $(Q^u)^l$. It is easily shown [DP90] that computing Q^{ul} for any $Q \subseteq P$ is a closure operator, i.e., (1) $Q \subseteq Q^{ul}$ (it is extensive), (2) $Q_1 \subseteq Q_2 \Rightarrow Q_1^{ul} \subseteq Q_2^{ul}$ (it is monotone) (3) $(Q^{ul})^{ul} = Q^{ul}$ (it is idempotent). It is easy to verify that principal ideals are always normal. Indeed, if $Q = D[e]$, then $Q^u = U[e]$ and $Q^{ul} = D[e] = Q$. Since normal cuts correspond to a closure operator, they are closed under intersection.

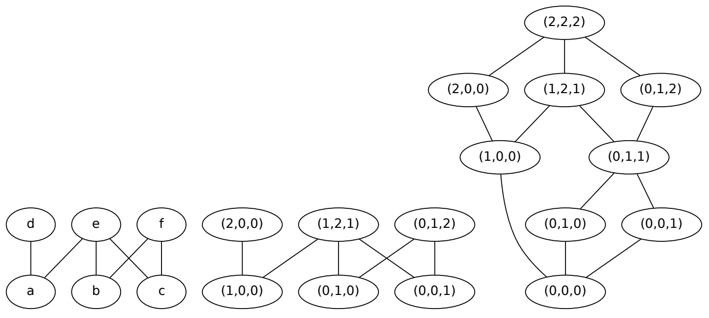

Fig. 2: (i) The original poset. (ii) Equivalent representation using Vector Clocks (iii) Its Lattice of normal cuts

Definition 2 (Dedekind–MacNeille Completion of a Poset). *For a given poset $P = (X, \leq)$, the Dedekind–MacNeille completion of P is the poset formed with the set of all the normal cuts of P under the set inclusion. Formally,*

$$DM(P) = (\{A \subseteq X : A^{ul} = A\}, \subseteq).$$

For the poset in Figure 1(i), the set of all normal cuts is:
$\{\{\}, \{a\}, \{b\}, \{a, c\}, \{a, b, d\}, \{a, b, c, d\}\}$.
The poset formed by these sets under the $\subseteq$ relation is shown in Figure 1(ii). This new poset is a complete lattice. The meet of normal cuts is same as the set intersection. The join of a set of normal cuts Q is defined as the meet of all the normal cuts that that are greater than or equal to all the normal cuts in Q. For example, the join of $\{a\}$ and $\{b\}$ is $\{a, b, d\}$ because it is the meet of all normal cuts which contain both $\{a\}$ and $\{b\}$. Our original poset P is embedded in this new structure such that x is mapped to the set $D[x]$.

For the poset in Fig. 2, the lattice of normal cuts has 9 cuts:
$\{\{\}, \{a\}, \{b\}, \{c\}, \{a, d\}, \{b, c\}, \{a, b, e, c\}, \{b, c, f\}, \{a, b, c, d, e, f\}\}$. Figure 2(iii) shows these 9 cuts in the vector clock representation. The lattice of consistent cuts has 19 elements (not shown in the figure).

4 IDML: An Incremental Algorithm for Lattice Completion

Let P be a poset and L be its Dedekind-Macneille lattice completion. In this section, we present a new incremental algorithm for lattice completion in implicit representation in which both P and L are represented using vectors. Suppose that a new element x is added to P with the constraint that x is not less than or equal to any of the existing elements. Our goal is to compute the lattice completion, L' of $P' = P \cup \{x\}$ given P and L. When P is a singleton, then its completion is itself. By adding one element at a time in any linear order that is consistent with the partial order, we can use the incremental algorithm for lattice completion of any poset.

Our incremental strategy for the lattice completion is as follows. We show that all the elements of L other than the top element of L are also contained in L'. The top element of L would either be retained or modified for L'. Therefore, except for the top, our algorithm will simply add elements to L to obtain L'.

Lemma 1. *Let S be a normal cut of $P = (X, \leq)$ such that $S \neq X$. Then S is also a normal cut of $P' := P \cup \{x\}$ where x is a maximal element of P'.*

Proof. Let $T = S^u$ in P. This implies that $T^l = S$ in P because S is a normal cut of P. Since $S \neq X$, T is nonempty (because if T is empty, $T^l = X$ which is not equal to S).

If $S \subseteq D[x]$, then S^u in P' equals $T \cup \{x\}$. We need to show that $S^{ul} = S$ in P', i.e., $(T \cup \{x\})^l = S$ in P'. The set $(T \cup \{x\})^l = T^l \cap D[x]$. Since x is a maximal element, we know that $x \notin T^l$. Since $T^l = S$ and $S \subseteq D[x]$, $T^l \cap D[x] = S$. Hence, S is a normal cut of P'.

If $S \not\subseteq D[x]$, then S^u in P' equals T. Since T is nonempty, and $T^l = S$ in P, we get that $T^l = S$ in P' as well. Hence, S is a normal cut of P'.

Our algorithm for DM-construction is shown in Fig. 3. Whenever a new element x arrives, we carry out three steps. In step 1, we process Y, the top element of L; in step 2, we add a normal cut corresponding to the principal ideal of x; and, in step 3, we process the remaining elements of L. The goal of step 1 is to ensure that L' has a top element. The goal of step 2 is to ensure that all principal ideals of P' are in L'. The goal of step 3 is to ensure that L' is closed under intersection. In step 3, we first check if x covers more than one element. If it does not, then we do not have to go over all normal cuts in L because of the following claim.

Lemma 2. *If x covers at most one element in P, then for any normal cut $W \in L$, $min(W, D[x]) \in L$ assuming $\{\} \in L$.*

Proof. If x does not cover any element of P, then $D[x] = \{x\}$ and $W \cap D[x] = \{\}$ which is assumed to be in L. Now suppose that x covers just one element y, then $D[x] = \{x\} \cup D[y]$. Therefore, $W \cap D[x] = W \cap D[y]$. Since both W and $D[y]$ are normal cuts of L, so is $W \cap D[y]$.

```
Input: a nonempty finite poset $P$, its DM-completion $L$, element $x$
Output: $L' :=$ DM-completion of $P \cup \{x\}$

$D[x] :=$ the vector clock for $x$;
$Y := top(L)$;
$newTop := max(D[x], Y)$;
// Step 1: Ensure that $L'$ has a top element
    if $Y \in P$ then $L' := L \cup \{newTop\}$;
    else $L' := (L - Y) \cup \{newTop\}$;
// Step 2: Ensure that $D[x]$ is in $L'$
    if $(D[x] \neq newTop)$ then $L' := L' \cup \{D[x]\}$;
// Step 3: Ensure that all meets are defined
    if $x$ does not cover any element in $P$ then
        $L' := L' \cup \{\mathbf{0}\}$; // add zero vector
    else if $x$ covers more than one element in $P$ then
        for all normal cuts $W \in L$ do
            if $min(W, D[x]) \notin L'$ then $L' := L' \cup min(W, D[x])$;
```

Fig. 3: Incremental Algorithm IDML for DM-construction

We now show the correctness of the algorithm, i.e., L' is precisely the DM-lattice for P'.

Theorem 1. *The algorithm IDML computes DM-completion of P' assuming that L is a DM-completion of P.*

Proof. We first show that all cuts included in L' are normal cuts of P'. In step 1, we add to L' all cuts of L except possibly $top(L)$, and $max(D[x], Y)$. All elements of L except possibly $top(L)$ are normal cuts of P' from Lemma 1. The cut $max(D[x], Y)$ is a cut of P', because it includes all elements of P'. In step 2, we add cut $D[x]$ to L' which is a normal cut of P' because it is a principal ideal of P'. In step 3, we only add cuts of the form $min(W, D[x])$. Since both W and $D[x]$ are normal cuts of P', and the set of normal cuts is closed under intersection, we get that $min(W, D[x])$ is also a normal cut of P'.

We now show that all normal cuts of P' are included in L'. Let S be a normal cut of P'. Let Q be the set of all principal ideals of P'. By our construction, L' includes all principal ideals of P' (because of step 2). It is sufficient to show that L' is closed under joins and meets. Since we have the top element in L', it is sufficient to show closure under meets. Let S and T be two normal cuts in L'. If both S and T are in L, then $S \cap T$ is in L and therefore also L'. Now, assume that $S \in L' - L$. Therefore, $S = W \cap D[x]$ for some $W \in L$. If $T \in L$, then $S \cap T = W \cap D[x] \cap T = (W \cap T) \cap D[x]$. Since $(W \cap T) \in L$, we get that $S \cap T \in L'$ because of step 3. If $T \in L' - L$, then it can be written as $W' \cap D[x]$ for some $W' \in L$. Therefore, $S \cap T = W \cap D[x] \cap W' \cap D[x] = (W \cap W') \cap D[x]$. Since $(W \cap W') \in L$, we again get that $S \cap T \in L'$. Since L' contains all principal ideals of P' and is closed under meet and join, we get that all normal cuts of P' are included in L'.

Note that our algorithm also gives an easy proof for the following claim.

Lemma 3. *The number of normal cuts of $P \cup \{x\}$ is at most twice the number of normal cuts of P plus two.*

Proof. For every cut in L, we add at most one more cut in Step 3 of the algorithm. Further, we add at most one cut in step 1 and one additional cut in Step 2.

We now discuss the time complexity of the IDML algorithm. Let m be the size of the lattice L. The time complexity of the IDML algorithm is dominated by step 3. Assuming that L is kept in a sorted order (for example, in the lexicographically sorted order) in a balanced binary search tree, the operation of checking whether $min(W, S) \in L$ can be performed in $O(w \log m)$, where w is the width of the poset P'. For any element for which we traverse the lattice L, we take $O(wm \log m)$ time. If the element x covers only one element (or no elements), then we take $O(w \log m)$ time. Suppose that there are r events in the poset that cover at least two events. In a distributed computation, only *receive* events would have this property. Then, to compute DM-Lattice of a poset P, we can repeatedly invoke IDML algorithm in any total order consistent with P. Therefore, we can construct DM-lattice of a poset P of width w with r elements of lower cover of size at least two in $O(rwm \log m)$. The algorithm by Nourine and Raynoud [NR99,NR02] takes $O(n^2m)$ time. Since $n \leq m \leq 2^n$, our algorithm takes time $O(rwn \log n)$ when $m = O(n)$.

We also note here that given a poset P, to construct its DM-lattice, we can restrict our attention to its subposet of irreducible elements because DM-completion of P is identical to DM-completion of the subposet containing all its join and meet irreducibles [DP90].

5 Traversal Based Algorithms for DM-completion

In some distributed computing applications, we may be interested not in storing the DM-Lattice but simply enumerating all the elements of the lattice or storing only those elements of the lattice that satisfy a given property. Recall that the size of the DM-Lattice may be exponential in the size of the poset in the worst case. Algorithm IDML has space complexity of $O(mw \log n)$ to store the lattice L (there are m elements in the lattice, and each element is represented using a w dimensional vector of entries of size $O(\log n)$). We now give an algorithm BFS-DML that does not require storing the entire lattice.

5.1 Breadth First Search Enumeration of Normal Cuts

The algorithm BFS-DML views the lattice as a directed graph and generates its elements in the breadth-first-order. It is different from the traditional BFS algorithm on a graph because we do not store the graph or keep data that is proportional to the size of the graph (such as the nodes already visited). Let $Layer(k)$ be the set of nodes in the graph that are at distance k from the bottom element of the lattice. Let w_L be the size of the largest set $Layer(k)$. Then, the space required by BFS-DML is $O(w_L w \log n)$.

The algorithm BFS-DML is shown in Figure 4. The set $\mathcal{S}$ is used to store the set of nodes that have been generated but have not been explored yet. The set is kept in a balanced binary tree so that it is easy to check if some element is already contained in the set. We maintain the invariant that the set $\mathcal{S}$ contains only the normal cuts of the poset P. The elements in the binary search tree are compared using the function *levelCompare* shown in Fig. 4. For any vector a corresponding to a consistent cut, the function $a.sum()$ returns the number of events in the consistent cut. At lines (1) and (2) of the function *levelCompare*, we define a consistent cut to be smaller than the other if it has fewer elements. Lines (3)-(5) impose a lexicographic order on all consistent cuts with equal number of elements. As a result, the function *levelCompare* imposes a total order on the set of all consistent and normal cuts.

The main BFS traversal of normal cuts, shown in lines (1) to (6), exploits the fact that there is a unique least normal cut that contains any consistent cut. The algorithm removes normal cuts from $\mathcal{S}$ in the *levelCompare* order. Let H be the smallest vector in this order (line 2). It finds all consistent cuts reachable from H by executing a single event e (line 4). We define an event e to be *enabled* in H if $H \cup \{e\}$ is a consistent cut. It adds all normal cuts that corresponds to "closure" of consistent cuts $H \cup \{e\}$ at line (5). We need to ensure that no normal cut is enumerated twice. At line (6), we check if a normal cut is already part of $\mathcal{S}$. It can be shown that this check is sufficient to ensure that no normal cut is enumerated twice (due to the definition of *levelCompare* and the BFS order of traversal).

We now discuss the complexity of the BFS algorithm. At line (4), since there are w processes, there can be at most w events enabled on any normal cut H. Checking whether an event is enabled in H requires that the events that happened-before e in poset P are included in H. This check requires $O(w)$ comparisons in the worst case.

To find the smallest normal cut containing $Q := H \cup \{e\}$, we simply compute Q^{ul}. Since $f \leq g$ is equivalent to $U[g] \subseteq U[f]$, we can restrict our attention to maximal elements of Q, i.e.,

$$Q^u = \cap_{f \in maximal(Q)} U[e].$$

Since P is represented using w chains, there are at most w maximal elements and therefore we can compute Q^u in $O(w^2)$ operations. We now take $R := Q^u$ and compute R^l, again using $O(w^2)$ operations. Thus, step (5) can be implemented in $O(w^2)$.

To check if the resulting normal cut K is not in $\mathcal{S}$, we exploit the tree structure of $\mathcal{S}$ to perform it in $O(w \log |\mathcal{S}|)$ which is $O(w \log w_L)$ in the worst case. Hence the total time complexity of Algorithm BFS is $O(mw(w^2 + w \log w_L))$ $= O(mw^2(w + \log w_L))$. The main space complexity of the BFS algorithm is the data structure $\mathcal{S}$ which is $(w_L w \log n)$. Note that the size of $\mathcal{S}$ is proportional to the size of the layer of the lattice in BFS enumeration (w_L) and is much smaller than the size of the lattice m used in the IDML algorithm.

```
    Input: a finite poset $P$
    Output: Breadth First Enumeration of elements of DM-completion of $P$
    $G$ := bottom element ;
    $\mathcal{S}$ := TreeSet of VectorClocks initially $\{G\}$ with levelCompare order;
(1)     while ($\mathcal{S}$ is notEmpty)
(2)         $H$ := remove the smallest element from $\mathcal{S}$;
(3)         output($H$);
(4)         foreach event $e$ enabled in $H$ do;
(5)             $K$ := the smallest normal cut containing $Q := H \cup \{e\}$;
(6)             if $K$ is not in $\mathcal{S}$, then add $K$ to $\mathcal{S}$;

        int function levelCompare(VectorClock $a$, VectorClock $b$)
(1)     if ($a.sum() > b.sum()$) return 1;
(2)     else if ($a.sum() < b.sum()$) return -1;
(3)     for (int $i = 0$; $i < a.size()$; $i$++)
(4)         if ($a[i] > b[i]$) return 1;
(5)         if ($a[i] < b[i]$) return -1;
(6)     return 0;
```

Fig. 4: Algorithm BFS-DML for BFS Enumeration of DM-Lattice

5.2 Depth First Search Enumeration of Normal Cuts

Another useful technique to enumerate elements of the lattice is based on the depth first search order. In BFS enumeration, the storage required is proportional to the width of the lattice whereas in DFS enumeration the storage required is proportional to the height of the lattice. Given any poset with n elements, the width of its lattice of normal cuts may be exponential in the size of the poset, but the height is always less than or equal to n. Hence, the DFS enumeration may result in exponential savings in space.

The algorithm for DFS enumeration is shown in Fig. 5. From any normal cut, we explore all enabled events to find the normal cuts. There are at most w enabled events and for each event it takes $O(w^2)$ time to compute the normal cut K at line (3). Since we are not storing the enumerated elements explicitly, we need a method to ensure that the same normal cut is not visited twice. For example, in Fig. 1, the normal cut $\{a, b, d\}$ is reachable from $\{a\}$ as well as $\{b\}$. Let *pred*(K) be the set of all normal cuts that are covered by K in the lattice. We use the total order *levelCompare* defined in Section 5.1 on the set *pred*(K). We make a recursive call on K from the normal cut G iff G is the maximum normal cut in *pred*(K) in the *levelCompare* order. Line (4) finds the maximum predecessor M using the traversal on the dual poset P^d. The dual of a poset $P = (X, \leq)$ is defined as follows. In the poset P^d, $x \leq y$ iff $y \leq x$ in P. It is easy to verify that S is a normal cut in P iff S^u is a normal cut in P^d. The function *get-Max-Predecessor*, shown in Fig. 5 uses expansion of a normal cut in the poset P^d to find the maximum predecessor.

```
    Input: a finite poset $P$, starting state G
    Output: DFS Enumeration of elements of DM-completion of $P$
(1)         output($G$);
(2)         foreach event $e$ enabled in $G$ do
(3)             $K$ := smallest normal cut containing $Q := G \cup \{e\}$;
(4)             $M$ := get-Max-predecessor($K$) ;
(5)             if $M = G$ then
(6)                 DFS-NormalCuts(K);

    function VectorClock get-Max-predecessor(K) {
    //takes K as input vector and returns its maximum predecessor normal cut
(1)         $H$ = MinimalUpperBounds($K$); // $H := K^u$
(2)         // find the maximal predecessor using normal cuts in the dual poset
(3)         foreach event $f$ enabled in the cut $H$ in $P^d$ do
(4)             $temp_f := H - \{f\}$; // advance on event $f$ in $P^d$ from cut $H$;
(5)             // get the set of lower bounds on $temp_f$
(6)             $pred$ := MaximalLowerBounds($temp_f$) using $H^l$;
(7)             if (levelCompare($pred$, $maxPred$) = 1) then $maxPred = pred$;
(8)         return $maxPred$;
```

Fig. 5: Algorithm DFS-DML for BFS Enumeration of DM-Lattice

The function *get-Max-predecessor* works as follows. At line (1), we compute $H = K^u$, which is the normal cut in P^d corresponding to K. Our goal is to compute all predecessors of K in P which corresponds to all successors of H in P^d. To find successors of H, we consider each event f enabled in H in P^d. At line (4), we compute the consistent cut $temp_f$. The closure of $temp_f$ in P^d equals $temp_f^{ul}$ in P^d. Equivalently, we can compute $temp_f^{lu}$ in P. The closed set $temp_f^{lu}$ in P^d corresponds to the closed set $temp_f^{lul}$ in P. However, we know that $temp_f^{lul}$ is equal to $temp_f^l$. Therefore, by computing $temp_f^l$ for each f enabled in H, we get all the predecessors of K in P^d. Since there can be w events enabled in H in P^d, and it takes $O(w^2)$ time to compute each predecessor, it would take $O(w^3)$ to determine the maximum predecessor. However, since $temp_f$ and H differ on a single event, we can compute $temp_f^l$ using $H^l = K$ in $O(w)$ time. By this observation, the complexity of computing max-predecessor reduces to $O(w^2)$, and the total time complexity to determine whether K can be inserted is $O(w^2)$.

In line (5) of DFS-DML, we traverse K using recursive DFS call only if M equals G. Since the complexity of step (3) and step (4) is $O(w^2)$, the overall complexity of processing a normal cut G is $O(w^3)$ due to the *foreach* at line (2). Since there are m normal cuts, we get the total time complexity of DFS-DML algorithm as $O(mw^3)$.

The main space requirement of the DFS algorithm is the stack used for recursion. Every time the recursion level is increased, the size of the normal cut increases by at least 1. Hence, the maximum depth of the recursion is n. Therefore, the space requirement is $O(nw \log n)$ bits because we only need to store vectors of dimension w at each recursion level. Hence, the DFS algorithm takes significantly less space than the BFS algorithm.

6 Applications of Normal Cuts in Distributed Systems

6.1 Finding the Meet and Join of Events

Suppose that there are two events x and y on different processes that correspond to faulty behavior. It is natural to determine the largest event, z, in the computation that could have affected both x and y. The event z is simply the meet of events x and y if it exists in the underlying computation. For example, in Fig. 2(a), suppose that the faulty events are $\{d, e\}$. In this case, the "root" cause of faults of these events could be event a. In the vector clock representation, the root cause is $(1, 0, 0)$ in the DM-Lattice. Now consider the case when the set of faulty events is $\{e, f\}$. In this case, the underlying computation does not have a unique maximum event that affects both e and f. It can be seen in Fig. 2(a) that both the events b and c could be the "root" cause of the events e and f. This is exactly what we would get from the lattice of normal cuts. The largest normal cut that is smaller than both events e with vector clock $(1, 2, 1)$ and event f with vector clock $(0, 1, 2)$ equals the vector $(0, 1, 1)$ which correctly identifies the set of events that affect both e and f.

Dually, we may be interested in the smallest event z that happened-after a subset of events. In a distributed system, an event z can have the knowledge of event x only if x happened-before event z. If two events x and y happened on different processes, the minimum event z that knows about both x and y corresponds to their join.

6.2 Detecting Global Predicates in Distributed Systems

A global predicate on a distributed computation is a boolean function B defined on its set of consistent cuts. If B is true on a consistent cut G, then we denote it as $B(G)$. The problem of detecting a global predicate *possibly* : B corresponds to determining if there exists a consistent cut G in the computation that satisfies B. The global predicate detection problem is NP-complete [CG98] even for the restricted case when the predicate B is a singular 2CNF formula of local predicates [MG01]. The key problem is that the lattice of consistent cuts L_{CGS} may be exponential in the size of the poset. The lattice of normal cuts, L_{DM} of a poset P is a suborder of the L_{CGS} (every normal cut is consistent, but every consistent cut may not be normal). Its size always lies between the size of the poset P and the size of the lattice of consistent cuts of P. In particular, it may be exponentially smaller than L_{CGS}. We now show that a class of predicates can be efficiently detected by traversing the lattice of normal cuts rather than L_{CGS}.

The class of predicates we discuss are based on the idea of knowledge in a distributed system[HM84]. We define knowledge predicates based on the happened-before relation. We use the notation $G[i]$ to refer to events of G on process i.

Definition 3. *Given a distributed computation, or equivalently a poset $(P, \leq)$, we say that every one knows the predicate B in the consistent cut G, if there exists a consistent cut H such that H satisfies B and for every process i there exists an event e in $G[i]$ such that all events in H happened before e. Formally, $E(B, G) \equiv \exists H : B(H) \wedge \forall i \exists e \in G[i] : \forall f \in H : f \leq e$.*
We also define $E(B) \equiv \exists G : E(B, G)$

Intuitively, the above definition says that a predicate is known to everyone in the system if every process has a consistent cut in its past in which B was true. The definition captures the fact that in a distributed system, a process can know about remote events only through a chain of messages.

We now show that instead of traversing L_{CGS} we can traverse L_{DM} to detect $E(B)$ for any global predicate B.

Theorem 2. *Let B be any global predicate and G be a consistent cut such that $E(B, G)$. Then, there exists a normal cut N such that $E(B, N^u)$.*

Proof. Since everyone knows B in G, by the definition of "everyone knows", we get that there exists a consistent cut $H \subseteq G$ such that B is true in H and every process in G knows H. Let $\mathcal{K}$ be the set of all consistent cuts that know H. The set is nonempty because $G \in \mathcal{K}$. Furthermore, it is easy to show that the set $\mathcal{K}$ is closed under intersection. The least element K of the set $\mathcal{K}$ corresponds to the minimal elements of the filter H^u. Hence, we conclude that $E(B, K)$.

Define N to be the consistent cut corresponding to H^{ul}. It is clear that N is a normal cut because it corresponds to the closure of H. Moreover, $N^u = H^{ulu} = H^u = K$. The first equality holds by the definition of N and the second equality holds due to properties of u and l operators. Since K equals N^u, from $E(B, K)$ we get that $E(B, N^u)$.

7 Conclusions and Future Work

We have proposed algorithms for the construction and enumeration of the lattice of normal cuts of a poset of a distributed computation. We have also shown their application to distributed computing.

It is clear that enumeration or construction of a lattice of size m in which each element is represented using $w \log n$ bits requires $\Omega(mw \log n)$ time. The problem of finding an algorithm that matches the lower bound is open.

8 Acknowledgements

I am thankful to Bharath Balasubramanian for discussions on the topic.

References

[AV01] Sridhar Alagar and Subbarayan Venkatesan. Techniques to tackle state explosion in global predicate detection. *IEEE Transactions on Software Engineering*, 27(8):704–714, 2001.

[CG98] Craig M. Chase and Vijay K. Garg. Detection of global predicates: Techniques and their limitations. *Distributed Computing*, 11(4):191–201, 1998.

[CM91] R. Cooper and K. Marzullo. Consistent detection of global predicates. In *Proc. of the Workshop on Parallel and Distributed Debugging*, pages 163–173, Santa Cruz, CA, May 1991.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, Cambridge, UK, 1990.

[Fid89] C. J. Fidge. Partial orders for parallel debugging. *Proc. of the ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging*, 24(1):183–194, January 1989.

[Gan84] B. Ganter. Two basic algorithms in concept analysis. Technical Report 831, Techniche Hochschule, Darmstadt, 1984.

[Gar03] Vijay K. Garg. Enumerating global states of a distributed computation. In *Intl Conf. on Parallel and Distributed Computing and Systems*, pages 134–139, November 2003.

[Gar13] Vijay K. Garg. Maximal antichain lattice algorithms for distributed computations. In *Proc. of Distributed Computing and Networking - 14th International Conference, ICDCN 2013*, January 2013.

[GK98] Bernhard Ganter and Sergei Kuznetsov. Stepwise construction of the dedekind-macneille completion. In *Conceptual Structures: Theory, Tools and Applications*, volume 1453, pages 295–302. 1998.

[GM01] V. K. Garg and N. Mittal. On slicing a distributed computation. In *21st Intnatl. Conf. on Distributed Computing Systems (ICDCS' 01)*, pages 322–329, Washington - Brussels - Tokyo, April 2001. IEEE.

[GW94] V. K. Garg and B. Waldecker. Detection of weak unstable predicates in distributed programs. *IEEE Trans. on Parallel and Distributed Systems*, 5(3):299–307, March 1994.

[GW97] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1997.

[HM84] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. In Tiko Kameda, Jayadev Misra, Joseph Peters, and Nicola Santoro, editors, *PODC*, pages 50–61. ACM, 1984.

[JRJ94] Guy-Vincent Jourdan, Jean-Xavier Rampon, and Claude Jard. Computing on-line the lattice of maximal antichains of posets. *Order*, 11:197–210, 1994.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. of the ACM*, 21(7):558–565, July 1978.

[Mat89] F. Mattern. Virtual time and global states of distributed systems. In *Proc. of the Intl. Workshop on Parallel and Distributed Algorithms*, pages 215–226, 1989.

[MG01] N. Mittal and V. K. Garg. On detecting global predicates in distributed computations. In *21st Intnatl. Conf. on Distributed Computing Systems (ICDCS' 01)*, pages 3–10, Washington - Brussels - Tokyo, April 2001. IEEE.

[NR99] Lhouari Nourine and Olivier Raynaud. A fast algorithm for building lattices. *Inf. Process. Lett.*, 71(5-6):199–204, 1999.

[NR02] Lhouari Nourine and Olivier Raynaud. A fast incremental algorithm for building lattices. *J. Exp. Theor. Artif. Intell.*, 14(2-3):217–227, 2002.

[SY85] R. E. Strom and S. Yemeni. Optimistic recovery in distributed systems. *ACM Trans. Comput. Syst.*, 3(3):204–226, 1985.