String Realizers of Posets with Applications to Distributed Computing

Vijay K. Garg*
garg@ece.utexas.edu

Chakarat Skawratananond
skawrata@ece.utexas.edu

Electrical and Computer Engineering Department
The University of Texas at Austin
Austin, Texas 78712.

ABSTRACT

In this paper, we show the connection between vector clocks used in distributed computing and dimension theory of partially ordered sets. Based on this connection, we provide lower bounds on the number of coordinates for timestamping events in a distributed computation for capturing the happened-before relation. To this end, we introduce the notion of a string realizer and the string dimension of a poset. For distributed computing and other applications, the concept of string realizer is more natural than the chain realizer used in the classical dimension theory. We establish the relationship between the string dimension and the chain dimension of a poset. Using this relationship and Dilworth's theorem for the chain dimension of finite distributive lattices, we obtain the desired lower bound. The concept of strings also has applications in efficient encoding of partial orders because it requires fewer bits to encode a string realizer than a chain realizer.

1. INTRODUCTION

A distributed computation has been widely modeled as a partially ordered set (poset) $(E, \rightarrow)$ where E is the set of events in the computation and $\rightarrow$ is the happened-before relation[12]. Fidge[10] and Mattern[13] independently introduced vector clocks to timestamp events such that happened-before relationship between any two events can be determined by examining their timestamps. In particular, in a distributed computation of N processes, vector timestamps provide the following property:

$$\forall e, f \in E : e \rightarrow f \iff v(e) < v(f)$$

where $v(x)$ is the N-dimensional vector timestamp of any event x. In other words, the poset of events is isomorphic to the set of vectors in dimension N. The vector clock mechanism requires each process to send its vector clock on all its outgoing messages. For large N, this mechanism introduces significant overhead during the computation. It is natural to ask if there is an alternative mechanism with lower overhead for timestamping events.

The first lower bound argument on the size of the vector clocks is due to Charron-Bost[4]. Her result states that for all N, there exists a computation on N processes such that any assignment of events to $\mathcal{R}^k$ which captures the happened-before relation (and its complement) must have $k \geq N$. Note that this result excludes possibility of lower dimensional vector clocks only for that particular computation on N processes. However, it does not exclude timestamps which may use less than N coordinates for other computations on N processes. In fact, it is easy to show distributed computations on N processes that do not require N-dimensional vector timestamps to capture happened-before relation and concurrency. We show that N is indeed a lower bound on the size of vector clocks if an additional property is required from vector clocks. This property states that vector clocks can also be used to determine the relationship between consistent cuts of the computation. We show that any vector clock mechanism that satisfies this property must have dimension at least N *for all* distributed computations on N processes (such that poset corresponding to the computation has width N). Indeed, Mattern and Fidge's vector clocks satisfy this additional property and therefore must have dimension at least N.

Our results are based on drawing connections between vector clocks used in distributed computing and dimension theory of partially ordered sets [15]. The dimension of a partially ordered set, first introduced by Dushnik and Miller[8], is defined as the least number of total orders such that the partial order is the intersection of these total orders. One of the advantages of this concept is that it provides an encoding scheme (or a timestamping scheme) for a partial order. If the dimension of a partial order on n elements is k, then each element can be assigned a code of size $k \log n$ such that the ordering between any two elements can be derived in k comparisons. Essentially, each element is represented by a

*supported in part by the NSF Grants ECS-9907213, CCR-9988225, Texas Education Board Grant ARP-320, an Engineering Foundation Fellowship, and an IBM grant.

k-tuple representing its position in each of the k orders. Although, the concept of dimension has been used successfully for many mathematical applications (for example in characterizing planar posets [15]), we argue that the concept of string dimension introduced in this paper is more useful for distributed computing applications.

We first generalize the concept of a total order to that of a string. Let (X, P) be a poset where X is a set, and P is a reflexive, antisymmetric, and transitive binary relation on X. (X, P) is a string if and only if there exists a mapping f from X to $\mathcal{N}$ (the set of natural numbers) so that $\forall x, y \in X : x < y$ iff $f(x) < f(y)$. The main difference from a chain is that we let $f(x) = f(y)$ when x and y are concurrent. Every chain and every anti-chain is a string. Mathematically, a string is a lexicographic sum of chains and anti-chains. We then introduce the concept of a string realizer of a poset. Informally, a string realizer of a poset is a set of strings such that the relationships in the poset can be derived using the strings. The vector clocks that have been used in distributed computation correspond to realizers using strings rather than total orders. We then introduce the notion of a string dimension of a poset as the least number of strings required to realize the poset. We show, somewhat surprisingly, that the string dimension of a poset is exactly equal to the dimension of the poset whenever the string dimension is at least 2. This establishes a relationship between dimension theory and vector clock mechanisms which are more like strings.

We then use standard results in dimension theory to derive results about vector clocks. We show that the theorem by Charron-Bost is a corollary of a result by Dushnik-Miller. Further, by using Dilworth's theorem on dimension of distributive lattices, we determine the string dimension of the lattice of consistent cuts induced by a distributed computation.

There are other advantages of strings and string dimension. For example, we show that a string realizer leads to a more efficient encoding scheme for a partial order than a chain realizer. We also define the notion of a string extension of a poset. A string extension is generalization of the concept of topological sort of a poset. We show that for every poset there exists a string extension whose length is equal to the height of the poset.

In summary, the paper makes the following contributions.

- We introduce the concepts of string, string realizer and string dimension and show that they are more natural for distributed computing applications than those based on chains.
- We establish that the string dimension of a poset is same as the chain dimension for any poset that is not a string.
- We show that Charron-Bost's result follows from a result by Dushnik and Miller[8].
- We show that Fidge and Mattern's vector clock also provides ordering information on the lattice of consistent cuts induced by the partial order. We show that any such mechanism must have at least dimension N using Dilworth's theorem.
- We show that, in general, string encoding of partial orders is more efficient than chain encoding. We also give an algorithm that generates a code with at most $\log(height(P)+1) * width(P)$ bits for any poset (X, P). This method is superior to code based on chain realizers when the height of the poset is small.
- We introduce and prove the existence of a normal string extension of a poset, and show its application to task scheduling.

Although this paper is concerned only with distributed computing applications, we note that encoding of partial orders also have applications in Databases[1], Artificial Intelligence[9] and programming languages development[5].

2. BACKGROUND: DIMENSION THEORY

2.1 Partially Ordered Set and Lattices

A pair (X, P) is called a partially ordered set or poset if X is a set and P is a reflexive, antisymmetric, and transitive binary relation on X. We call X the *ground set* while P is a *partial order* on X. We write $x \leq y$ and $y \geq x$ in P when $(x, y) \in P$. Also, $x < y$ and $y > x$ in P means $x \leq y$ in P and $x \neq y$.

We use *Hasse diagrams*[1] to represent finite posets. If $x < y$ in P, then x appears lower than y in the diagram.

Let $x, y \in X$ with $x \neq y$. If either $x < y$ or $y < x$, we say x and y are *comparable*, and write $x \perp y$. On the other hand, if neither $x < y$ nor $x > y$, then we say x and y are incomparable, and write $x || y$. A poset (X, P) is called *chain* if every distinct pair of points from X is comparable in P. Similarly, we call a poset an *antichain* if every distinct pair of points from X is incomparable in P. A point $x \in X$ is called a *maximal* point (minimal point) if there is no point $y \in X$ with $x < y$ in P ($x > y$, respectively). We denote the set of all maximal points by $max(X, P)$, while $min(X, P)$ denotes the set of all minimal points.

A chain C of a poset (X, P) is a *maximum chain* if no other chain contains more points than C. We use similar definition for *maximum antichain*. The height of the poset P, denoted by $height(P)$, is the number of points in the maximum chain. Similarly, the width of the poset P, denoted by $width(P)$, is the number of points in a maximum antichain. We say (X, P) and (Y, Q) are isomorphic, if there exists a $1-1$ and onto map $f : X \longrightarrow Y$ so that $x_1 \leq x_2$ in P if and only if $f(x_1) \leq f(x_2)$ in Q.

An element $y \in X$ is called an *upper bound* for S if $s \leq y$ in P, for every $s \in S$. An upper bound y for S is the *least upper bound* for S, abbreviated l.u.b.(S), provided $y \leq y'$ in P for every upper bound y' of S. *Lower bounds* and *greatest lower bounds* are defined similarly. The poset is called a *lattice* if every nonempty finite subset $S \subseteq X$ has the least upper bound as well as the greatest lower bound.

[1]The formal definition of Hasse diagrams can be found in [6].

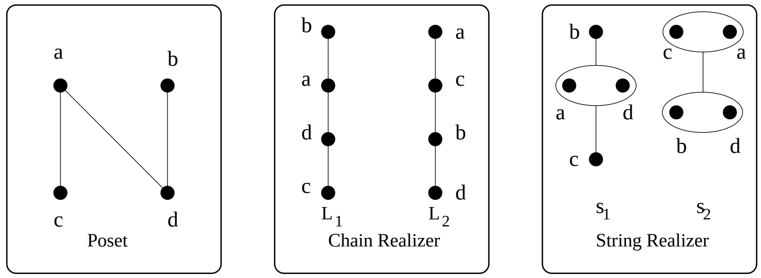

Figure 1: (X, P)

When $\mathbf{L} = (X, P)$ is a lattice, for any $x, y \in P$, we define

$$x \vee y = l.u.b\{x, y\} \qquad \text{(join)}$$

$$x \wedge y = g.l.b\{x, y\} \qquad \text{(meet)}$$

$\mathbf{L}$ is called *distributive lattice* if for all $x, y, z \in X$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

Next we provide the definition of *down-set* and the poset of the form $\mathbf{2^P}$. Let $\mathbf{P} = (X, P)$ be a poset and let $S \subseteq X$. S is called a *down-set* in (X, P) if $x \in S$ whenever $y \in S$ and $x \le y$ in P. Let D denote the family of all down-sets of $\mathbf{P}$. Define a partial order Q on D by $D_1 \le D_2$ in Q if and only if $D_1 \subseteq D_2$. Then the poset $\mathbf{Q} = (D, Q)$ is isomorphic to $\mathbf{2^P}$.

2.2 Dimension

A family $\mathcal{R} = \{L_1, L_2, \ldots, L_t\}$ of linear orders on X is called a *chain realizer* of a poset (X, P) if $P = \cap\mathcal{R}$. $x < y \in L_i \cap L_j$ if $x < y$ in both L_i and L_j. We also say that $\mathcal{R}$ *realizes* P. Figure 1 shows a poset P in which $\{L_1, L_2\}$ realizes P.

It can be shown [15] that $\mathcal{R}$ is a realizer of P iff for every $x, y \in X$ with $x \parallel y$ (x incomparable to y) in P, there exists distinct integers i, j with $1 \le i, j \le t$ for which $x < y$ in L_i and $y < x$ in L_j. In the following, we write $x <_c y$ when $x < y$ in L_c,

Definition 1. *[15] For any poset (X, P), the dimension of (X, P), denoted by $dim(X, P)$, is the least positive integer t for which there exists a family $\mathcal{R} = \{L_1, L_2, \ldots, L_t\}$ of linear extensions of P so that $P = \cap\mathcal{R} = \cap_{i=1}^{t} L_i$.*

The dimension of the poset in Fig. 1 is 2. The concept of dimension provides us a way to encode a partial order. The elements of a partial order with dimension t can be encoded with a t-dimensional vector as follows. For any element x, the vector v_x is defined as follows: $v_x[i]$ = number of elements less than x in L_i, for $1 \le i \le t$. Given code for two elements v_x and v_y, we have the following order:

$$v_x < v_y \iff \forall i : v_x[i] < v_y[i] \tag{1}$$

For example, the code for a and b in the poset in Figure 1 is $(2, 3)$ and $(3, 1)$ based on the realizer. Based on the code and (1), it can be easily determined that a and b are concurrent. We call the order given by (1) the *chain order*.

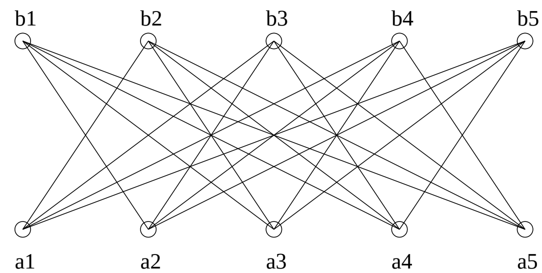

Figure 2: S_5

The dimension of a poset can be arbitrarily large. Consider a poset (X, P) where $X = \{a_1, a_2, \ldots, a_n\} \cup \{b_1, b_2, \ldots, b_n\}$, and $a_i < b_j$ in P if and only if $i \ne j$, for $i, j = 1, 2, \ldots, n$. This class of posets is known as *the standard example* and denoted by S_n. Figure 2 shows the diagram for S_5. The following Theorem is due to Dushnik and Miller[8].

Theorem 2. *[8] $dim(S_n) = n$.*

Let $L_i = [a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, b_i, a_i, b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n]$, where a_1 is the lowest element, and b_n is the highest element in chain L_i Then $\mathcal{R} = \{L_1, L_2, \ldots, L_n\}$ is a realizer of S_n.

3. STRING

In Section 2 we saw that classical dimension theory provides lower bounds on the dimension of vectors when the comparison is based on the *chain order*. On the other hand, the vector clocks in distributed computing use vector ordering given by the following (2) which we call *vector order*.

$$u < v \equiv \begin{array}{l} \forall k : 1 \le k \le N : u[k] \le v[k] \wedge \\ \exists j : 1 \le j \le N : u[j] < v[j] \end{array} \tag{2}$$

Consider a distributed system in which the code of elements is determined in a decentralized fashion. In this case the relationship between two events may not be known globally. Thus, if event e happened before f, this relationship may be known only to a single process. From the perspective of other processes, e and f may be indistinguishable (for example, when both are internal to the process). This is more easily captured in the vector order where a vector u is deemed as smaller than vector v even when u is smaller than v in just one component and same in all the other components. Since chain order requires that all the coordinates in code of event e are strictly less than all the respective coordinates in code of event f, it is difficult to use chain order in a distributed system. In this section, we generalize the concepts in dimension theory so that the ordering used between codes is identical to (2).

We first give the definition of a *string.*

Definition 3 (string). *A poset (X, P) is a string if and only if $\exists f : X \rightarrow \mathcal{N}$(the set of natural numbers) such that $\forall x, y \in X : x < y$ iff $f(x) < f(y)$*

The set of elements in a string which have the same f value is called a *knot*. For example, a poset (X, P) where $X = \{a, b, c, d\}$ and $P = \{(a, b), (a, c), (a, d), (b, d), (c, d)\}$ is a string because we can assign $f(a) = 0, f(b) = f(c) = 1$, and $f(d) = 2$. Here, b and c are in the same knot. The difference between a chain and a string is that a chain requires existence of a *one-to-one* mapping such that $x < y$ iff $f(x) < f(y)$. For strings, we drop the requirement of the function to be *one-to-one*. We represent a finite string by the sequence of knots in the string. Thus, P is equivalent to the string $\{(a), (b, c), (d)\}$.

A chain is a string in which every knot is of size 1. An anti-chain is also a string with exactly one knot. Note that a string drops the distinction between elements which have the same order relationship with all other elements. Thus, two elements x and y have the same code $f(x) = f(y)$ iff for any element z, (1) $x < z$ iff $y < z$, and (2) $z < x$ iff $z < y$. This is a more natural concept for ordered sets.

A string gives more efficient encoding of the partial order than the use of chains. At an extreme, the range of f may be finite even when the domain of f is infinite. For example, the following order { all even numbers } < { all odd numbers } on natural numbers can be encoded by assigning 0 to all even numbers and 1 to all odd numbers. Such a poset cannot be assigned codes using the classical dimension theory.

We write $x \leq_s y$ if $x \leq y$ in string s, and $x <_s y$ if $x < y$ in string s.

Definition 4 (String Realizer). *For any poset (X, P), a set of strings $\mathcal{S}$ is called a string realizer iff $\forall x, y \in X$: $x < y$ in P if and only if*

1. *$\forall s \in \mathcal{S} : x \leq_s y$, and*
2. *$\exists t \in \mathcal{S} : x <_t y$.*

The definition of less-than relation between two elements in the poset based on the strings is identical to the less-than relation as used in vector clocks. This is one of the motivation for defining string realizer in the above manner. A string realizer for the poset in Fig. 1 is given by two strings

$$s_1 = \{(c), (d, a), (b)\} \qquad s_2 = \{(d, b), (c, a)\}$$

There are two important differences between definitions of string realizers and chain realizers. First, if $\mathcal{R}$ is a chain realizer of a poset P, then P is simply the intersection of linear extensions in $\mathcal{R}$. This is not true for a string realizer (see Fig. 1). Secondly, all the total orders in $\mathcal{R}$ preserve P, i.e., $x < y$ in P implies that $x < y$ in all chains in $\mathcal{R}$. This is not true for string realizer. For example, $d < a$ in poset P of Fig. 1, but (d, a) appears as a knot in the string s_1. We are only guaranteed that a will not appear lower than d in any string - they may appear in the same knot.

Now, analogous to the dimension we define

Definition 5 (String Dimension). *For any poset (X, P), the string dimension of (X, P), denoted by $sdim(X, P)$, is the size of the smallest set of strings $\mathcal{S}$ such that $\mathcal{S}$ is a string realizer for (X, P).*

Example 6. *Consider the standard example S_n. The following function f can be used to create a string realizer of S_n. For all $k, i = 1, 2, \ldots, n$,*

$$f_k(a_i) = \begin{cases} 0 & \textit{if } k \neq i \\ 1 & \textit{otherwise} \end{cases}$$

$$f_k(b_i) = \begin{cases} 0 & \textit{if } k = i \\ 1 & \textit{otherwise} \end{cases}$$

For example,

$$\begin{array}{ll} a_1 = (1, 0, 0, \ldots, 0), & b_1 = (0, 1, 1, \ldots, 1) \\ a_2 = (0, 1, 0, \ldots, 0), & b_2 = (1, 0, 1, \ldots, 1) \end{array}$$

In this example, the length of each string is 2 and thus each element requires only n bits for encoding. If we use classical dimension based on total orders, each element would require $n * \log n$ bits.

Example 7. *Consider the poset (X, P) as follows.*
$X = \{\emptyset, \{a\}, \{b\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$
$P = \{(A, B) \in X \times X : A \subseteq B\}$.

A string realizer for the poset can be obtained as follows. For each set $A \in X$, we use a bit vector representation of the set A. Thus, $\{a, c\}$ is represented by $(1, 0, 1)$ and the set $\{a, b\}$ is represented by $(1, 1, 0)$. This representation gives us a string realizer with three strings such that every string has exactly two knots.

We now establish the relationship between string dimension and chain dimension. It may appear, at first, that the string dimension of a poset may be much smaller than the chain dimension. However, this is not the case as shown by the following result.

Theorem 8 (Equivalence Theorem). *For any poset (X, P) such that $sdim(P) \geq 2, sdim(P) = dim(P)$*

Proof. There are two cases.
$\underline{sdim(P) \leq dim(P)}$.

It is sufficient to show that for any chain realizer of size k, there exists a string realizer of equal or smaller size. Given a chain realizer $\mathcal{C}$, we construct the string realizer as follows. Each chain is simply viewed as a string. Our obligation is to show that the order generated from the string realizer is the same as the one based on chain realizer (recall that the definition of *less than* for string realizer is different from *less than* in a chain realizer.) In this proof, let $x \mapsto y \iff \forall s \in \mathcal{S} : x \leq_s y \ \wedge \ \exists t \in \mathcal{S} : x <_t y$. It is sufficient to show that $x < y \iff x \mapsto y$. First, we show that $x < y \implies x \mapsto y$.

$$\begin{array}{rcl} x < y & \implies & \forall c \in \mathcal{C} : x <_c y \\ & \implies & \forall s \in \mathcal{S} : x <_s y \\ & \implies & \forall s \in \mathcal{S} : x \leq_s y \ \wedge \ \exists t \in \mathcal{S} : x <_t y \\ & \implies & x \mapsto y \end{array}$$

Next, we show that $\neg(x < y) \implies \neg(x \mapsto y)$. There are two cases.

$$\begin{array}{lrcl} \text{case A:} & y < x & \implies & y \mapsto x \qquad \text{(From case I)} \\ & & \implies & \neg(x \mapsto y) \\ \text{case B:} & x \parallel y & \implies & \neg(x < y) \ \wedge \ \neg(y < x) \\ & & \implies & \exists c \in \mathcal{C} : x <_c y \ \wedge \ \exists d \in \mathcal{C} : y <_d x \\ & & \implies & \exists s \in \mathcal{S} : x <_s y \ \wedge \ \exists t \in \mathcal{S} : y <_t x \\ & & \implies & \neg(y \mapsto x) \ \wedge \ \neg(x \mapsto y) \\ & & \implies & \neg(x \mapsto y) \end{array}$$

$\underline{dim(P) \leq sdim(P)}$.

Given a string realizer of P, $\mathcal{S}$, we construct the corresponding chain realizer. We achieve this by untying knots of the string to form a chain.

First consider the case when two elements x and y belong to the same knot in all strings. We will combine these elements into one element say z. After finding the chain realizer of the new set, we replace z with x and y. Further, in one chain we keep x less than y and in another chain we keep y less than x. Observe that we can do this because there are at least two chains due to our assumption of $sdim(P) \geq 2$.

Now assume that there are no two elements as in the first case. Consider any knot $\{x_1, x_2, \ldots, x_m\}$ in any string s_1. Now we determine for all pairs (x_i, x_j) of the elements in the knot.

1. If $(\forall s \in \mathcal{S} - \{s_1\} : x_i \leq_s x_j) \ \wedge \ (\exists t \in \mathcal{S} - \{s_1\} : x_i <_t x_j)$, then we get $x_i < x_j$. or
2. If $\exists s, t \in S - \{s_1\} : (x_i <_s x_j) \ \wedge \ (x_j <_t x_i)$, then we get $x_i \parallel x_j$.

Then, we can untie this knot by performing the topological sort. By repeating this process, all knots on s_1 can be untied, and we obtain the chain. □

Figure 3 shows an example of this untying mechanism. Since d and e appear in the same knot in all strings, we first combine d and e into one element f. As a result we get strings in Figure 3(C). We then untie the knot (c, f) by keeping c less than f in s_1 and untie the knot (a, b) in s_2 by keeping a less than b. We now have the chain order. Now we replace f by d and e, keeping d less than e in s_1 and e less than d in s_2 to get the chains in Figure 3(D).

4. APPLICATIONS OF STRING REALIZERS AND DIMENSION

4.1 Lower Bound on Dimension of Vector Clocks

As we have mentioned before, the definition of a string realizer is identical to the definition for vector clocks in distributed systems. A distributed computation on N processes can be modeled as a poset of events $(E, \rightarrow)$ of width N. Fidge and Mattern's vector clocks are simply string realizers of the poset $(E, \rightarrow)$. For example, consider the poset in Fig. 3 which has width two. We can view it as a computation on two processes, the first process executes events a, b and d in that order, and the second process executes c and e in that order. By viewing b and c as send events received at e and d respectively, we get the following vector clocks for all events:

$$v(a) = (1, 0); v(b) = (2, 0); v(d) = (3, 1);$$

$$v(c) = (0, 1); v(e) = (2, 2)$$

This corresponds to two strings

$$s_1 = \{(c), (a), (b, e), (d)\} \quad \text{and} \quad s_2 = \{(a, b), (c, d), (e)\}$$

This is a different string realizer than shown in Figure 3, but has the same dimension.

Now that we have established equivalence of dimension and string dimension for non-string posets, we can use existing results from dimension theory to prove results on dimension of vector clocks.

We first consider lower bounds on the (string) dimension of vector clocks. Charron-Bost[4] has shown that we require at least N-dimensional vector timestamps to capture concurrency in the distributed computation consisting of N processes. The proof is by constructing a computation in which any timestamping scheme with less than N coordinates is not able to capture concurrency accurately. The following proof uses dimension theory and our equivalence theorem.

THEOREM 9. *For every N, there exists a distributed computation $(E, \rightarrow)$ on N processes such that any assignment from E to $\mathcal{N}^k$ that captures concurrency relation on E has $k \geq N$.*

PROOF. The result is trivially true for N equal to 1. For any $N \geq 2$, consider the standard example S_N shown in Figure 2. Define a_i and $b_{(i \mod N)+1}$ to be on process P_i. This computation is on N processes. By Dushnik and Miller's Theorem, this poset has dimension N. From Theorem 8, the computation has string dimension also equal to N. Any assignment from E to $\mathcal{N}^k$ that captures concurrency relation, results in a string realizer with k strings. Since the string dimension is N, it follows that $k \geq N$. □

Although this result proves that there cannot be a uniform timestamping mechanism of less than N coordinates, it does not exclude timestamping mechanism which may use less than N coordinates for a particular computation.

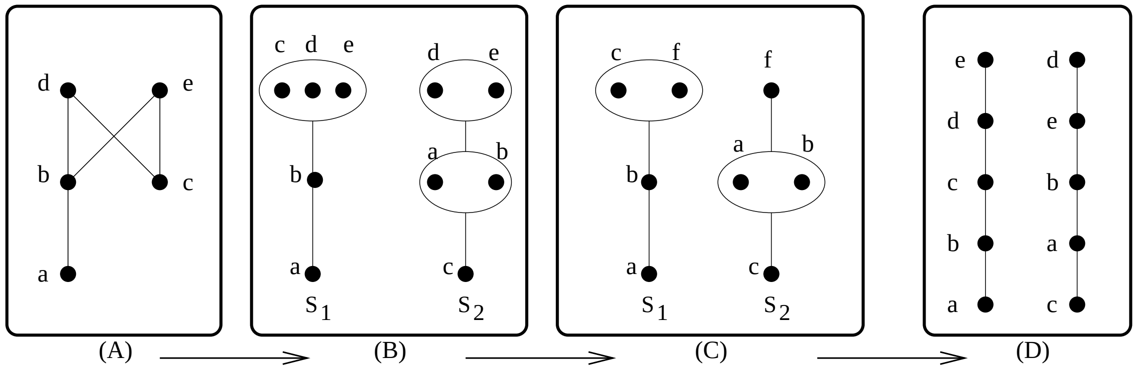

Figure 3: An example of untying mechanism.

As an extreme example, consider a system of N processes, where $N > 3$. Assume that processes do not send any messages to each other. We can timestamp each event j on process i by the vector $v_i(j) = (i, n-i, j)$. It is easy to see that this timestamping mechanism captures concurrency relation accurately[2].

Next we show that N-dimensional vector clocks of Fidge and Mattern (FM vectors for short) have an additional property that makes it necessary to have dimension N for all computations. In particular, FM vectors satisfy the following property. If f and g are two *distinct* events such that event f is on process $f.p$, then

$$v(f)[f.p] \le v(g)[f.p] \Rightarrow f \to g \qquad (3)$$

As a result of this property we show that FM vectors can also be used to timestamp elements of another poset - the lattice of consistent cuts of the computation $(E, \to)$. For notational convenience, we use $e.v[i]$ to denote the i^{th} component of the vector clock assigned to the event e. Recall that F is a consistent cut of $(E, \to)$ iff

$$(f \in F) \ \wedge \ (e \to f) \Rightarrow e \in F$$

For a consistent cut F, we define its timestamp as

$$F.v[i] = max\{e.v[i] \mid e \in F\}$$

Theorem 10 shows that the proposed vector timestamp for consistent cuts based on FM vector clocks captures the relation $\subseteq$ between consistent cuts.

Theorem 10. $F \subseteq G \iff F.v \le G.v$

Proof. It is easy to see that

$$F \subseteq G \implies F.v \le G.v$$

We show that $F.v \le G.v \implies F \subseteq G$. Let $\neg(F \subseteq G)$. This implies that there exists $f \in F - G$.

$$\begin{array}{lll} & & \{\text{definition of } \le\} \\ F.v \le G.v & \implies & \forall i \exists g \in G : f.v[i] \le g.v[i] \\ & & \{\text{definition of } f\} \\ & \implies & \exists g \in G : f \neq g \ \wedge \ f.v[f.p] \le g.v[f.p] \\ & & \{\text{FM vector clock property}\} \\ & \implies & \exists g \in G : f \to g \\ & & \{G \text{ is a consistent cut}\} \\ & \implies & f \in G \end{array}$$

Contradiction. □

We now explore the structure of the set of all consistent cuts under the relation $\subseteq$. Consistent cuts are identical to down-sets in lattice theory. A standard result in lattice theory states

Theorem 11. *[2, 14] Given any poset P, let 2^P be the poset formed by the set of its down-sets under $\subseteq$ order. Then, 2^P is a* distributive *lattice.*

Further, a result due to Dilworth tells us the dimension of a distributive lattice.

Theorem 12. *[7] Let $\mathbf{L}$ be a distributive lattice. Choose a poset $\mathbf{P} = (X, P)$ so that $\mathbf{L}$ is isomorphic to $\mathbf{2^P}$. Then $dim(\mathbf{L}) = width(P)$.*

Combining our equivalence theorem with these results, we get

Theorem 13. *Any vector clock mechanism that captures $\subseteq$ relation on the set of consistent cuts in a distributed computation of width N (equivalently, on N processes), must have at least N coordinates.*

Proof. Follows from Theorems 11, 12 and 8. □

4.2 Encoding Partial Orders

The concept of string realizer has the advantage over chain realizer that it generally requires less number of bits to encode a partial order using string realizer. Formally, consider

[2]In fact, this partial order can be encoded using vector clocks of dimension 2.

the following problem. Given a partial order (X, P), define a coding function $code : X \longrightarrow \{0,1\}^k$ and a binary relation $<$ on codes such that $\forall x, y \in X : x < y$ in $P \iff code(x) < code(y)$. Note that the order relation may be any arbitrary order (not necessarily vector order). The only requirement is that it can only use the bits in $code(x)$ and $code(y)$ to determine the order. It is clear that any partial order can be coded using $log(n) + n$ bits per element as follows. For every element, we store a binary array of size n. Further, each element is assigned a unique index into the array. Let $index(x)$ be the index of x in $1..n$ and $x.v$ be the n bit array for element x. Then, we determine the order between x and y as follows. $x < y$ iff $(x.v[index(y)] < y.v[index(x)])$.

Using dimension theory, partial orders of lower dimensions can be encoded much more succintly. If a partial order has dimension k, then it can be encoded using $k * log(n)$ bits. However, when the dimension is large (as for the standard example), this method may take upto $n/2 * log(n)$ bits per element.

String realizers typically result in a lower number of bits for encoding. From Theorem 8, we know that for coding purposes, the total number of coordinates based on total orders and strings are the same. The difference lies in the number of bits required to code a single coordinate. Given a string realizer R. If R has k strings each of length less than or equal to l, then (X, P) can be coded using $k \log l$ bits. l is clearly less than or equal to $|X|$. Depending upon the structure of the poset, l may be much smaller than $log(n)$ as seen for the case of the standard example.

In general, we have the following result.

THEOREM 14. *Every partial order (X, P) on $n \geq 2$ elements can be encoded using a string realizer in at most $\log(height(P) + 1) * width(P)$ bits.*

PROOF. For convenience, let $w = width(P)$. We use Dilworth's chain covering theorem which states that (X, P) can be partitioned into w chains $C_1, C_2, ..., C_w$. We then use the transitively reduced diagram of (X, P) with w processes as given by the chain decomposition. Further, we use Fidge and Mattern's algorithm to assign vector timestamp for each event when the poset diagram is viewed as a computation. These vector timestamps determine a string realizer with w coordinates such that no coordinate is greater than $height(P) + 1$. $\square$

There is a small change in application of Fidge and Mattern's algorithm in above construction. Their algorithm assumes that initial events of all processes are incomparable and assigns the initial event at process i a vector timestamp as follows:
$\forall j : j \neq i : v[j] = 0;$
$v[i] = 1;$

In our construction (in the proof of Theorem 14), all the initial events of chains may not be incomparable. To solve this problem, it is sufficient to add a special initial event for each chain whose smallest event is not a minimal event in the partial order. For example, consider the poset in Fig. 4. This poset can be decomposed into three chains $\{a, b, c\}, \{d, e\}$, and $\{f, g\}$. However, d is not a minimal element of the poset. Hence to apply, Fidge and Mattern's algorithm we may assume an event smaller than d which is incomparable to a and f in Process 2 with vector clock equal to $(0, 1, 0)$. Then, to compute the vector at d, we compute the maximum of vectors for a, f and $(0, 1, 0)$. Thus, the vector clock for all events can be derived as
$v(a) = (1, 0, 0); v(f) = (0, 0, 1); v(d) = (1, 1, 1);$
$v(b) = (2, 0, 0); v(c) = (3, 0, 0); v(e) = (2, 2, 1);$
$v(g) = (1, 1, 2).$

This results in the following string realizer:
$s_1 = \{(f), (a, d, g), (b, e), (c)\},$
$s_2 = \{(a, f, b, c), (d, g), (e)\}$, and
$s_3 = \{(a, b, c), (d, e, f), (g)\}.$

Observe that some strings may be longer than others and we need not use the same number of bits to encode positions in all the strings. The total number of bits required for a realizer with t strings is

$$\sum_{i=1}^{i=t} \lceil log(length(s_i)) \rceil$$

We note here that Bouchet[3] and Trotter[15] introduced a generalization of the original dimension by restricting the length of chains used in the realizer. This new dimension parameter is called k-dimension (denoted by $dim_k(P)$), when only the chains of length k are allowed in the realizer of P.

The k-dimension of P, $k \geq 2$, is the smallest positive integer t for which P is isomorphic to a subposet of K^t (ie. K^t is the product of t chains of length k). Therefore, the 2-dimension is the size of the smallest hypercube in which P can be embedded.

Obviously [11],

$$dim(P) \leq dim_{k-1}(P) \leq \ldots \leq dim_2(P)$$

One interesting question is to determine the smallest integer k, $2 \leq k \leq |P|$, such that $dim(P) = dim_k(P)$. Our procedure in the proof of Theorem 14, shows that

$$\forall k : k > height(P) : dim(P) = dim_k(P)$$

Habib et al.[11] went further by allowing chains of different length in the realizer of the poset. They defined a new dimension parameter called *encoding dimension* as follows.

The encoding dimension of a poset P, denoted by $edim(P)$, is the least integer t such that $t = \sum_{i=1}^{i=p} \lceil \log_2 k_i \rceil$ and P can be embedded into $K_1 \times K_2 \times \ldots K_p$, where K_i denotes a chain of length k_i.

It is shown in [11] that when P is an antichain, then $edim(P) = 2 \log |P|$. This is equal to the number of bits required in the Dushnik-Miller's dimension. However, by using string realizers, we can use only one bit to encode each element in an antichain.

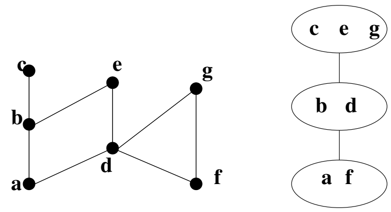

Figure 4: A Poset and its Normal String Extension

A key distinguishing feature of our work is that we allow order equivalent elements to have the same code. This is more natural concept for posets. Further, it allows hierarchical representation of orders. Two elements may have the same code at one level, but different at the other level when they are not distinguishable at coarser granularity but can be distinguished with finer granularity of the order. For example, in a distributed computation, all internal events between two external events may be assigned the same code at the coarser level of granularity.

4.3 Lower bound on Dimension of a Poset

We first define the notion of string length to derive a lower bound on dimension of any poset. The length of a realizer S for the poset P, denoted by $slength(P, \mathcal{S})$, is defined as the length of the longest string in the string realizer $\mathcal{S}$ of P. Let $slength(P)$ denote the length of the longest string in the string realizer with minimum number of strings. The following definition is useful in determining the lower bound on the dimension.

DEFINITION 15. *Let (X, P) be any poset. For $x, y \in X$, we say that x is order-equivalent to y (denoted by $x \sim y$) iff x is incomparable to y and for all $z \in X : x < z \equiv y < z$ and for all $z \in X : z < x \equiv z < y$*

Let $numeq(P)$ denote the number of equivalence classes of the relation $\sim$. The following lemma shows the relationship among $dim(P), slength(P)$ and $numeq(P)$.

LEMMA 16. $dim(P) \geq \log(numeq(P))/\log(slength(P))$.

PROOF. The proof follows from the fact that the total number of codes is equal to $slength(P)^{dim(P)}$. Further, two elements in different equivalence classes cannot have the identical code. This implies that $slength(P)^{dim(P)} \geq numeq(P)$. $\square$

The above lemma provides a lower bound on dimension of a poset P.

4.4 String Extension of Partial Orders

Many applications, for example, task scheduling with precedence constraints require that elements in a poset are processed in a order which does not violate precedence constraints. In general, topological sort of a partial order which produces a linear extension of partial order has been useful in this and other algorithmic applications. Similar to a linear extension, we can define a string extension of a partial order as follows.

DEFINITION 17. *A string s is a string extension of a partial order (X, P) if $\forall x, y \in X : x <_P y \Rightarrow x <_s y$.*

Note that in contrast to a chain realizer which contains linear extensions of a partial order, a string realizer does not necessarily contain string extensions.

We call s, a *normal* string extension of (X, P) if the length of s is equal to the height of (X, P). We have the following result.

THEOREM 18. *For every poset (X, P), there exists a normal string extension s.*

PROOF. The string s can be constructed by the following algorithm (that is implicit in Dilworth's anti-chain covering theorem). Remove all the minimal elements of the partial order and put them in the lowest knot. Get the next set of minimal elements and put them as the next knot. By repeating this procedure till all elements in (X, P) are removed we get the desired string. It can be easily verified that the string preserve order in (X, P) and has its length equal to the height of the poset. $\square$

For example, consider the poset in Fig. 4. The normal string extension produced using the construction in Theorem 18 is:

$$\{(a, f), (b, d), (c, e, g)\}$$

It is easily verified that the above string preserves the partial order.

If the poset (X, P) denotes tasks, then a normal string extension represents a processing schedule (assuming that concurrent tasks can be executed in parallel). The length of the string corresponds to a critical path in (X, P).

5. CONCLUSIONS

In this paper, we introduce a new class of posets called string and define the notions of string realizer and string dimension. We show that for distributed computing applications, these concepts are more natural than the corresponding classical concepts based on chains. In general, string encoding of partial orders is more efficient than chain encoding and easier to obtain in a distributed environment. We also establish that the string dimension of a poset is the same as the chain dimension for any poset that is not a string.

We show that Charron-Bost's result follows from the result by Dushnik and Miller[8]. We also show that Fidge and Mattern's vector clock provides ordering information on the lattice of consistent cuts induced by the partial order. By invoking Dilworth's theorem, we show that any mechanism that provides ordering information on the consistent cuts must have dimension equal to the width of the computation.

We also show applications of our theory in encoding partial orders and determining string extensions of a partial order.

6. REFERENCES

[1] R. Agrawal, A. Borgida and H.V. Jagdish. Efficient management of transitive relationships in large data bases including is-a hierarchies. *Proc. ACM SIGMOD* , 1989.

[2] G. Birkhoff. Lattice theory. *American Mathematical Society*, 25, 1967.

[3] A. Bouchet. Codages et dimensions de relations binaires. *Annals of Discrete Mathematics 23, Orders: Description and Roles, (M. Pouzet, D. Richard eds)*, 1984.

[4] B. Charron-Bost. Concerning the size of logical clocks in distributed systems. *Information Processing Letters*, 39:11–16, July 1991.

[5] Y. Caseau. Efficient handling of multiple inheritance hierarchies. *OOPSLA*, pages 271 – 287, 1993.

[6] B. Davey and H. Priestley. *Introduction to lattices and orders.* Cambridge University Press, 1991.

[7] R. Dilworth. Decomposition theorem for partially order sets. *Ann. Math.*, 51:161–165, 1950.

[8] B. Dushnik and E. Miller. Partially order sets. *American Journal of Mathematics*, 63:600–610, 1941.

[9] G. Ellis and F. Lehmann. Exploiting the induced order on type-labeled graphs for fast knowledge retrieval. *Proc. International Conference on Conceptual Structures, Lecture Notes in Artificial Intelligence*, Springer-Verlag 1994.

[10] C. Fidge. Partial orders for parallel debugging. In *Proceedings of the ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging*, pages 183–194, January 1989.

[11] M. Habib, M. Huchard, and L. Nourine. Embedding partially ordered sets into chain-products. In *Proceedings of symposium on Knowledge Retrieval, Use and Storage for Efficiency*, pages 147–161, Santa Cruz, 1995.

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

[13] F. Mattern. Virtual time and global states of distributed systems. In *Proceedings of the International Workshop on Parallel and Distributed algorithms*, pages 215–226, 1989.

[14] R. Stanley. *Enumerative Combinatorics Volumn 1.* Wadsworth and Brookes/Cole, Monterey, California, 1986.

[15] W. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory.* The Johns Hopkins University Press, 1992.