INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

An Alternate Representation of Distributed Computations

Céline VALOT
Dominique FORTIN

N° 1944
Juin 1993

PROGRAMME 1

Architectures parallèles,
bases de données,
réseaux et systèmes distribués

An Alternate Representation of Distributed Computations

Une Autre Représentation des Exécutions Réparties

Rapport de Recherche INRIA n^o

Céline Valot et Dominique Fortin

INRIA
B.P. 105, 78153 Le Chesnay Cédex, France
tel.: +33 (1) 39-63-59-00, telex: 697 033 F
email: Celine.Valot@nuri.inria.fr

Avril 1993

Abstract

This paper proposes another representation of the event partially ordered set of distributed executions which relies on diagrams, called *arc diagrams*. Arc diagrams are already in use within order theory, and appears to have useful properties which can be of interest to distributed algorithms. It is shown that arc diagrams describe efficiently event ordering relations as well as reachable global states. Access to both of those levels is done without switching between different representations. Furthermore, this representation allows to decompose the event set into chains, and yields to nearly optimal linear extensions. These linear extensions can be viewed as the *nearest* linear extensions of the poset that can be produced. Those can be efficiently used to detect a new kind of global predicates, called *greedy* predicates.

Résumé

Nous proposons une autre représentation de l'ensemble des événements d'une exécution répartie, qui repose sur des diagrammes, appelés *arc diagrammes*. Les arc diagrammes sont utilisés dans le cadre de la théorie des ordres et ont des propriétés utiles dans le domaine des algorithmes répartis. Les arc diagrammes décrivent efficacement les relations d'ordre entre événements ainsi que les états globaux atteignables. L'accès à ces deux niveaux se fait sans changer de représentation. De plus, cette représentation permet de décomposer l'ensemble des événements en chaînes, sans utiliser la décomposition par processus classique. Ces décompositions permettent d'obtenir des extensions linéaires presque optimales de l'ensemble des événements. Ces extensions linéaires peuvent êtres vues commes les extensions linéaires les plus proches de l'ensemble partiellement ordonné d'événements. Elle peuvent être efficacement utilisées pour détecter un nouveau type de prédicats globaux, que nous appelons des *prédicats greedy*.

Contents

1 Introduction **1**

2 The arc diagram representation **2**

2.1 Definitions and properties 2

2.2 Comparisons with classical representations 4

3 Chain decomposition of the execution poset **8**

3.1 A static chain decomposition 8

3.2 An incremental decomposition ? 11

4 Application to distributed computations **11**

4.1 Decomposing the events set into chains 11

4.2 Towards greedy predicates 12

5 Conclusion **14**

1 Introduction

The coding of the events partial order of a distributed execution has been widely studied by the research community working on distributed algorithms. Distributed algorithms require the ability to determine whether one of two events comes before or after the other. Events occurring on a single processor are totally ordered, and their ordering can be determined by reading the processor physical clock. Events occurring at different processes are just partially ordered, by processes exchanging messages. A sending of a message always precedes its receiving.

By means of timestamping mechanisms from Lamport, Mattern and Fidge, and the more recent interval clock [Diehl 1992b], one can have some knowledge on the order of event occurrences in a distributed execution, although the accuracy of these timestamping mechanisms vary [Valot 1993]. They have a cost, usually measured by the time needed to compute them and by the amount of memory used.

Another area of concern was the ability to observe the successive global states of a distributed execution (to check global properties on these states for example), to have a better understanding of the execution. This is a difficult task as no instantaneous global state is observable. Global states form also a partially ordered set, and to represent this poset, other data structures were introduced, with help of timestamping mechanisms to preserve causal consistency. Among these new data structures, the lattice of global states is a well-known representation.

Consequently, we can distinguish two different levels of abstraction: the first one aims at determining ordering relations between events. The second one applies to the observation of distributed global states. These levels are clearly related but had required different mechanisms to handle them.

One main question led to the work described in this paper, that is, the possibility to find another representation of an execution partial order, apart from the actual timestamping mechanisms. We argue that the arc diagram approach is a good alternate representation of an execution poset.

Arc diagrams were originally used to realize project analysis, solving constrained jobs scheduling problems. Penalizing cost is assigned to a job which is executed immediately after another one which is not constrained to precede it; each such occurrence is called a *jump*. The *jump number* problem is then to schedule the jobs to minimize the number of jumps [Sysło 1987]. Within order theory, the jump number problem is to find a linear extension of a partially ordered set, with the minimum number of jumps (ordered elements within the linear extension which are unordered in the original poset). Arc diagrams have been proved to be useful in solving this problem.

We show in this paper that the arc diagram representation allows the description of

the two distributed abstraction levels cited above. As a consequence, it can be considered as a good alternate representation by contrast to the pair timestamps/global states lattice. Furthermore, an arc diagram permits to efficiently decompose the poset elements into chains, and to build linear extensions of the poset. Those linear extensions are of great interest for tracing algorithms. Also they influence the detection of a new kind of global predicates, called greedy predicates, which are introduced in this document.

The next section of this document (Section 2) presents the arc diagram approach by first describing its definitions and properties in Section 2.1 and, by establishing comparisons between this representation and classical ones such as the event lattice in Section 2.2. Such comparisons are made by distinguishing the two abstraction levels noted above, for both of which the arc diagram appears to be an adequate representation. Section 3 introduces the way the jump number problem is approximated by decomposing the poset into various kind of greedy chains, using an arc diagram representation. This decomposition yields nearly optimal linear extensions of the execution poset. At this stage, our main contribution to the original work done by Syslo is the introduction of an on-the-fly decomposition algorithm, presented in Section 3.2. Section 4 describes the many interesting usages of the decomposition into greedy chains, for example, for tracing algorithms. Section 4.2 presents how global predicates can be advantageously detected on greedy linear extensions. Finally, Section 5 contains concluding remarks.

2 The arc diagram representation

We summarize in this section the arc diagram representation of an execution poset, introduced by [Syslo 1985].

2.1 Definitions and properties

We recall here briefly somme commonly used definitions on distributed executions.

The set of events E on a set of processes $P_1, P_2, \cdots, P_n$ forms a distributed computation. The events considered are the sending and receiving of messages. as internal events which remain local to a process. With the well-known transitive causality relation θ such that: $\theta(e, f) \iff e$ *causally precedes* f, (E, θ) forms a partially ordered set.

If neither e causally precedes f, nor f causally precedes e, they are said to be *concurrent* events ($e \parallel f$). The concurrency relation is a symmetric relation but not transitive.

A *chain* is a subset of the overall set of events E of pairwise comparable elements (a totally ordered set). An *antichain* is a subset of E containing pairwise incomparable elements, i.e., concurrent.

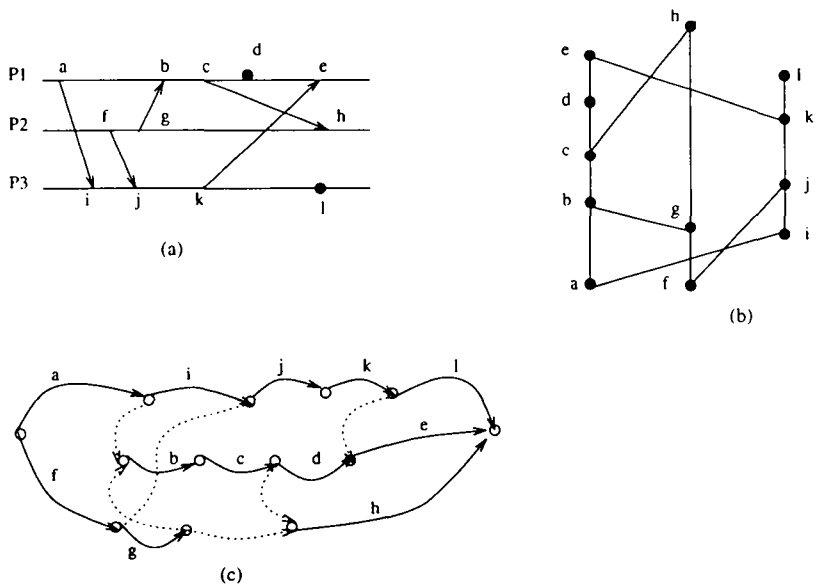

Figure 1: Two kind of diagrams of one computation

Time diagrams are a simplified and natural representation of a distributed execution, where dots represent internal events, arrows the sending and receiving of messages and horizontal lines the execution processes (see Figure 1 (a)).

An arc diagram $(D^A(E), \phi) = (X, R, t, h)$ is another representation of a partial order where X is a set of vertices, ϕ the mapping $\phi : E \rightarrow R$, R is the set of arcs of the form $a = (t(a), h(a))$ where t and h stands for *tail* and *head* respectively[1]. In this representation:

- the elements of the poset (E, θ) are used to label the arcs in such a way that $\theta(p, q)$ iff $(h(p), t(q)) \in R$. Labelled arcs are also called *poset arcs* and characterize the relations between the poset elements.
- the arc diagram vertices are not labelled and can be considered as some abstract states to which lead the arcs of the diagram.
- an arc diagram contains *dummy arcs*, which are not labelled, used to preserve the order relations, as poset arcs cannot always be added without changing the original partial order.
- a sequence of arcs $\pi = (a_1, a_2, \cdots, a_l)$ is a path of length l if $h(a_i) = t(a_{i+1})$ for $i = 1, 2, \cdots, l-1$.

[1]In the sequel, an arc a will be noted simply by a.

To whatever poset (E, θ), one cannot associate an unique arc diagram as dummy arcs can be created arbitrarily [Syslo 1985]. In the framework of distributed execution posets, it appears that dummy arcs need to be created at least when an event has two immediate predecessors which are unrelated, typically a receive event (see Figure 1).

The figure 1 illustrates the Hasse diagram (b) (also called *covering graph*) and one arc diagram (c) for a computation on three processes shown on (a).

An arc diagram is compact if: (1) it contains one source and one sink exactly; (2) every dummy arc is essential, i.e. if removed, it causes the loss of a precedence relation between some elements of the poset; (3) no dummy arc can be contracted (a dummy arc a can be contracted if $t(a)$ is not the tail of another arc or $h(a)$ is not the head of another arc).

A static algorithm to build an arc diagram from a given poset (E, θ) is presented in [Syslo 1985] and references are given to other such kind of algorithms. The arc diagram built is then compacted.

2.2 Comparisons with classical representations

Events ordering relations

Figure 2 recalls briefly the vector time timestamping mechanism and its main properties [Mattern 1988, Fidge 1989].

Mechanism	Properties
Vector time of size n V(e)[n]	$\forall\, e \in P_i, f \in P_j,\ \theta(e,f) \Leftrightarrow V_i(e)[j] < V_j(f)[j]$
	$e \parallel f \Leftrightarrow$ $\begin{cases} V_j(f)[i] < V_i(e)[i] \\ \textit{and} \\ V_i(e)[j] < V_j(f)[j] \end{cases}$

Figure 2: Vector time mechanism

Vector time has the property of completely characterizing causality as it is sufficient to compare two vectors to determine if the two corresponding events are related or not. The size of this timestamp is equal to the number of processes of the computation.

Proposition 2.1 *The arc diagram completely characterizes the partial ordering between events.*

Proof:

To prove this proposition, we have to show that the arc diagram has the same properties as vector time has. That means, $\theta(e,f) \Leftrightarrow \exists$ a path $\pi = (a_1, a_2, \cdots, a_l)$ such that $h(e) = t(a_1) \cdots h(a_l) = t(f)$ or if π is empty $h(e) = t(f)$. This follows from the definition of arc diagrams.

Also, $e||f$ iff $\nexists\ \pi$ such that $\theta(e,f)$ nor such that $\theta(f,e)$.

This shows that arc diagrams characterize causality and concurrency as vector time does. □

Moreover, the *covering relation* (noted $e \multimap f$, and such that, $f > x \geq e \Rightarrow x = e$) can be calculated without requiring extra data structures as it relies on the transitive reduction of poset arcs. In [Diehl 1992a] it is shown that vector time cannot decide if one event is an immediate predecessor of another. To calculate the covering graph, boolean vector clocks were introduced, in addition to the use of vector time and events are to be traced according to a linear extension of the causality relation. Building the covering graph of an execution with this method is very expensive. With an arc diagram, the covering information is immediate and we have $e \multimap f$ if $(h(e), t(f)) \in R$, i.e. is an arc of the arc diagram.

In order to consider an arc diagram as a useful representation of causality relationships, we have implemented the algorithm of [Sysło 1985] on-the-fly: the arc diagram is updated dynamically as event occurrences are read. To avoid an enormous growing of the arc diagram, the diagram is compacted as new arcs are inserted. We will not detail the numerous cases of the insertion/reduction of poset/dummy arcs but the main idea is that locality is preserved trough the insertion of new poset arcs; i.e. the transitive closure/reduction remain local to a given context.

Observing global states

Observing global states is another problem of distributed algorithms which had required the introduction of new data structures and new tools in order to be handled efficiently.

There is an exact correspondence between a consistent (w.r.t causality) global state and a *consistent cut*[2] as shown in [Schwarz 1991, Charron-Bost 1992] for example. The set of consistent cuts of a distributed execution is a partially ordered set, $(\mathcal{C}, \sqsubseteq)$.

The set $(\mathcal{C}, \sqsubseteq)$ can be represented in a geometric way (see for example [Charron-Bost 1992]). Consistent cuts appear to be interior points of a cone whose external edges correspond to broken lines drawn between consecutive events on each processes. Figure 3 (b) illustrates the geometric representation of consistent cuts of the computation of Figure 1 (c) considering the pairs of processes, $(P1, P2)$, $(P1, P3)$, and $(P2, P3)$.

[2] A consistent cut is a subset of the overall event set left closed with respect to the causality relation.

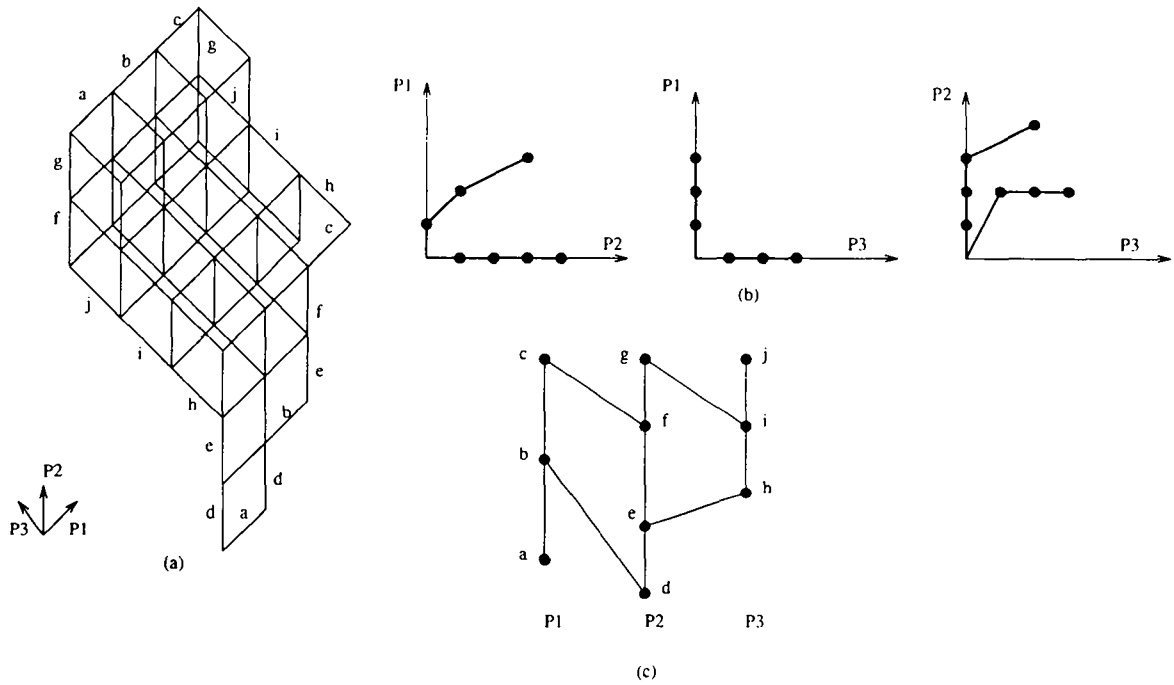

Figure 3: Representing reachable global states

However, one cannot speak of an execution global state without refering at the same time to the *observation* to which it pertains. Due to the non-determinism of distributed executions, two different observers can observe different execution courses. An observation is defined in the following way:

Definition 1 *[Schwarz 1991] An* **observation** *of a distributed execution is a linear extension* $(L, \ll)$ *of the poset* (E, θ) *such that for all events* $e \in E$ *the set* $\{e' \in E | e' \ll e\}$ *is finite.*

An observation will usually be done in a sequential manner, although in [Charron-Bost 1992] non-sequential observations are considered by allowing the simultaneous observations of concurrent events. All different observations that can be made correspond to all linear extensions that can be built from the original poset.

Lattices of global states were introduced to represent the overall set of reachable states within all observations of a distributed execution.

Proposition 2.2 *[Schwarz 1991, Charron-Bost 1992, Diehl 1992a] The ordered set of consistent cuts* $(\mathcal{C}, \sqsubseteq)$ *is a* **lattice** *and is another representation of the execution poset.*

Theorem 2.1 *[Bonnet 1982] Let E an ordered set and $\mathcal{I}(\mathcal{C}_E)$ the lattice of its consistent cuts.*
(1) If E' is a linear extension of E, then $\mathcal{I}(\mathcal{C}_{E'})$ is a maximal chain in $\mathcal{I}(\mathcal{C}_E)$.
(2) If C is a maximal chain of $\mathcal{I}(\mathcal{C}_E)$, then it exists a unique linear extension E' of E such that $\mathcal{I}(\mathcal{C}_{E'}) = C$.

The theorem 2.1, recalled above, says that to each path from a minimal element to a maximal one in the lattice, there corresponds a unique linear extension. The lattice represents then all linear extensions of the poset, therefore all possible observations that can be made. The Figure 3 (a) illustrates the lattice of reachable global states for the covering graph of Figure 3 (c). An on-the-fly algorithm for constructing the lattice of global states of a computation is presented in [Diehl 1992a].

Definition 2 *A* **greedy chain** *of (E, θ) is a chain C of E such that $Pred_{im}(p) \cup \{p\} = C$ with $p = sup\ C$ and for no $q \in Succ_{im}(p)$, the chain $C \cup \{q\}$ has this property.*

where $Pred_{im}(p)$ is the set of immediate predecessors of p and $Succ_{im}(p)$ the set of its immediate successors.

A greedy chain is then a maximal chain of covering elements. We know that any linear extension of E can be expressed as a linear sum $C_1 \oplus C_2 \oplus \cdots \oplus C_k$ of chains of E (with $sup\ C_i \not< inf\ C_{i+1}$ in E). A *greedy linear extension* is a linear extension such that each chain is a greedy chain. Moreover, by [Sysło 1987], we know that every linear extension can be transformed to a greedy one. The counterpart of greedy chains in an arc diagram is the concept of *greedy path*, which is a path $\pi = (a_1, a_2, \cdots, a_k)$ such that $\nexists a_j$ for which $t(a_j) = h(b), b \in R, (0 \leq j < k), a_k$ is a poset arc, and π cannot be extended to a path π' satisfying these properties; it exists a one-to-one correspondence between greedy chains and greedy paths of $(D^A(E), \phi)$.

Proposition 2.3 *The arc diagram represents all linear extensions of the execution poset.*

Proof:

There is a one-to-one correspondence between the lattice of an order (E, θ) and its compact arc diagram, $(D^A(E), \phi)$, which follows from the correspondence between greedy chains and linear extensions. Therefore all information within the lattice is preserved within the arc diagram representation.□

Access to all linear extensions from the arc diagram requires, however, some kind of unfolding process, as the arc diagram acts like an automaton over all linear extensions.

The consistent cuts geometry and the lattice of global states are both valuable representations of the execution poset. However, the former becomes cumbersome when the number of processes is greater than two, and the construction of the latter is extremely costly. No gain can be expected when building an arc diagram since the transitive closure of the causality relation has to be calculated, followed by the transitive reduction of this relation. Nevertheless it is a challenging candidate for representing distributed execution posets as it allows to describe ordering relations between events as well as reachable global states, by summarizing the linear extensions that can be built. For that purpose, with an arc diagram, the same data structure is used, without needs to switch between different representations.

Hopefully, in many cases, we only handle a subset of the set of linear extensions since we look for occurrences of global states according to some a priori knowledge on the execution (stable, regular predicates).

3 Chain decomposition of the execution poset

In addition to be an adequate representation, an arc diagram can be used to find chain decompositions of the execution poset. These decompositions are not based on the processes of the computation, as it is usually the case. In fact, in [Schwarz 1991, Diehl 1992a] the event set is decomposed into chains, according to the processes where the events have occurred. This decomposition is certainly the most immediate one, but it is far from an optimal decomposition (which should follow event chains without relying on the locality of events). The decompositions from an arc diagram are based on greedy chains instead and lead nearly optimal linear extensions.

We first describe a static chain decomposition algorithm. Section 3.2 sketches an on-the-fly version of this algorithm.

3.1 A static chain decomposition

Let's first define the jump number of a linear extension.

Definition 3 *Let (E, θ) a partially ordered set and $\mathcal{L}(E)$ the set of all its linear extensions. For $L \in \mathcal{L}$, the* **jump number** *$s(E, L)$ is the number of pairs $(a, b) \in E$ such that $a \prec\!\!\!- b$ in L and $a \not< b$ in E. This pair is called a jump of L.*

The jump number $s(E)$ of (E, θ) is the minimum of $s(E, L)$ over the set $\mathcal{L}$. The problem is to evaluate $s(E)$ and build linear extensions of (E, θ) with $s(E)$ jumps. These linear extensions, for which $s(E, \tilde{L}) = s(E)$, are called *optimal* linear extensions. The number of

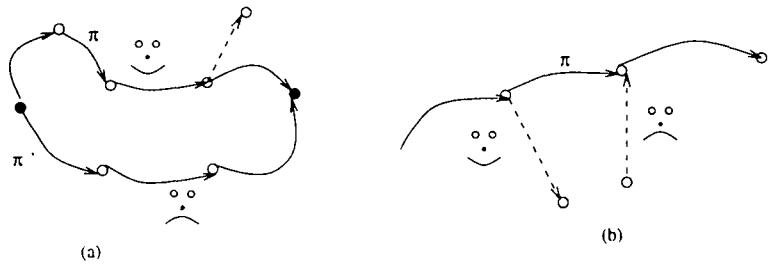

Figure 4: Strongly and semi-strongly greedy chains

jumps in a linear extension is obviously influenced by the number of unordered elements in the original poset, which are ordered within the linear extension.

Lemma 3.1 *[Sysło 1987] Every linear extension L of a poset E can be transformed to a greedy linear extension*[3] *$G(L)$ such that $s(E, G(L)) \leq s(E, L)$.*

The lemma 3.1 leads to a first approximation of an optimal linear extension. By appropriately arranging greedy chains to form a greedy linear extension, one can expect minimizing the number of jumps.

Furthermore, some kind of greedy paths allows a better approximation of the "nearest" linear extension of a given poset. By nearest, we mean the distance between the ordering relations of the poset elements and their image mirrored in a linear extension. The definitions of these greedy paths are introduced below.

Definition 4 *A greedy path π is* **strongly greedy** *if in addition to be greedy, (1) $h(\pi)$ is the sink of $D^A(E)$, or (2) $h(\pi)$ is the head of a poset arc b ($b \neq a_k$) such that no poset path ending in b has a vertex incident to a dummy arc.*

Definition 5 *A greedy path π is* **semi-strongly greedy** *if π has a vertex which is the tail of a dummy arc but not the head of a dummy arc.*

The Figure 4 illustrates strongly and semi-strongly greedy chains. The case (a) shows that π is a strongly greedy chain and that π' is not as π contains a dummy arc. The case (b) illustrates the condition to be a semi-strongly greedy chain highlighted by the smiling face.

A strongly greedy path π contributes one to the jump number of the poset; $h(\pi) = h(b)$ means that $a_k \in \pi$ is concurrent with $b_k \in b$, which can possibly create a jump (the elements involved in this jump depend however on the way greedy chains are arranged).

[3]See definition on page 7.

The main idea with a semi-strongly greedy path π is to suppress dummy arcs, whose existence could be the cause of a jump. These remarks give rise to theorem 3.1 and to theorem 3.2 enonced below.

Theorem 3.1 *[Sysło 1987] Let π a strongly greedy path of $D^A(E)$. Every greedy linear extension L of E can be transformed in a greedy linear extension L^* starting with the chain C_π with $s(E, L^*) \leq s(E, L)$.*

It is proved that an arc diagram which does not admit a strongly greedy path, contains a semi-strongly greedy path, which can be also used to reduce the jump number. The main result of Sysło's work is summarized by the following theorem :

Theorem 3.2 *[Sysło 1987] Every poset E has an optimal semi-strongly greedy linear extension and $s(E, L) \leq s(E) + \frac{1}{2}(a(D^A(E)) - 1)$ for every semi-strongly greedy linear extension L of E, where $a(D)$ is the number of dummy arcs in $D^A(E)$.*

This leads to a tight bound for evaluating the jump number of a poset.

By identifying in the arc diagram strongly and semi-strongly greedy chains, one can obtain approximations of optimal linear extensions[4]. The jumps of the linear extension break it into C_i chains; this leads to a chain decomposition $L = C_0 \oplus C_1 \oplus \cdots \oplus C_k$ of the execution poset with k jumps.

The static algorithm [Sysło 1987] for finding such a decomposition is :

1. Find a strongly greedy chain C_π in $(D^A(E), \phi)$. C_π is the starting chain of L, $L = C_\pi \oplus \cdots \oplus C_k$.
2. Remove the arcs of the path π in $(D^A(E), \phi)$.
3. Remove all dummy arcs with tails at non-terminal vertices of π.
4. Remove all dummy arcs d such that $(t(d), h(\pi)) \in R$ if there is no arc b such that $(h(b), h(\pi)) \in R$.
5. Remove all isolated vertices and compact the new arc diagram $D^A(E - C_\pi, \phi)$, contracting the remaining dummy arcs.
6. Return to (1) with $D^A(E - C_\pi, \phi)$.
7. If there is no strongly greedy chains, find semi-strongly greedy chains.

[4]Determining the exact jump number of a poset is known to be NP-complete for general posets.

Let's take an example and apply this decomposition algorithm to the arc diagram of Figure 1. We can first obtain a linear extension of the form $L_1 = ai \oplus fgbcd \oplus jkl \oplus e \oplus h$ which gives a jump number of 4. ai is a semi-strongly greedy chain, as is $fgbcd$, jkl a strongly greedy one, as is e and h. But we can obtain also $L_2 = ai \oplus fjkl \oplus gbcde \oplus h$ which has a smaller jump number (3), and is composed of strongly greedy chains, except the first one which is semi-strongly greedy.

3.2 An incremental decomposition ?

This decomposition algorithm works for posets which are statically given. In order to be of some utility in the framework of distributed executions, one must design an algorithm to calculate and update a decomposition on-the-fly.

The problem of an on-the-fly decomposition algorithm is the influence of the insertion of new arcs in the arc diagram being decomposed. To maintain the calculated decomposition as new arcs are inserted into the diagram, one has to know their influence on the strongly and semi-strongly attributes of chains, as the influence of these attributes changes on the rest of the decomposition.

The main idea is that, at a given stage, the current decomposition is stored as a sorted list of arc diagrams, each one corresponding to a chain of the decomposition. Whenever a new arc is to be inserted, one needs not redecompose the overall poset from the beginning, but just locate accurately the arc diagram involved. In fact, inserting arcs could only promote a semi-strongly greedy chain to a strongly greedy one; the previous decomposition could be updated from the located arc diagram upto the tail of the list. The worst case happens to be when the first arc diagram in the list has to be updated at every new arc insertion. We argue that there exists a locality property w.r.t. insertion into an arc diagram (due to causality constraints) but possibly no locality w.r.t. chain decomposition (even though the right location in the list can be found).

4 Application to distributed computations

We show in this section that the results enonced in the preceding section can be of great importance to distributed computations. In the framework of distributed executions, a semi-strongly greedy linear extension provides an efficient decomposition of the events set, which differs from the classical ones. Furthermore, this decomposition allows us to introduce greedy predicates which improve predicates classification (Section 4.2).

4.1 Decomposing the events set into chains

Within a distributed execution, the ability to decompose the events set E into $C_1, C_2, \cdots C_k$ chains is of interest.

À chain decomposition has a first usage when characterizing the partial order between events. It has been shown that Lamport clocks correspond to a decomposition composed of the overall events set itself. By contrast, Mattern and Fidge vector time corresponds to a decomposition where each C_i is composed of the events occurring on process P_i [Diehl 1992b]. The latter decomposition is the most immediate and explains the mandatory size of vector time when nothing is known about the computation. The idea of obtaining timestamps whose size could vary according to the decomposition considered has been studied in [Diehl 1992b] and [Valot 1993]; unfortunately, and thanks to Dilworth's theorem, which we recall hereafter:

Theorem 4.1 *[Dilworth 1950] An order (E, θ) of width[5] k can be decomposed into k disjoint chains $E = \uplus_{1 \leq i \leq k} E_i$, E_i chain*

the conclusion drawn, in order to obtain accurate timestamps, is that an arbitrary decomposition is useless; rather a chain decomposition close to the width of the partial order is better.

The chain decomposition on an arc diagram that we have presented in the preceding section appears to answer the problem of finding varying size timestamps. In the general case, $s(E) \geq w(E) - 1$ following Dilworth's theorem. By obtaining a greedy linear extension of k jumps, we could build timestamps of size k, the closest as possible to the optimal chain decomposition. We cannot assert that this k-size timestamp would have a smaller size than the n-size Mattern vector time (n being the number of processes). The reason is that we lose the process information in an arc diagram by contrast to other representations; however, this process information can be recovered by mapping the arc diagram on a set of processes but this cannot be done in an unique manner.

Nevertheless, one can be convinced that this greedy chain decomposition provides the "nearest" linear extension of the poset. Linear extensions have been widely used as they are easily represented and we argue that these nearest linear extensions can be of great utility to users tracing distributed algorithms, by providing an efficient trace of the execution.

An important application of linear extension is the evaluation of global predicates which is the subject of the next section.

[5]The order width is the cardinal of one of its largest antichains.

4.2 Towards greedy predicates

The main purpose of debugging or testing distributed executions is the evaluation of global predicates on such executions.

Predicates whose value depends on the local state of a process are said to be local and their detection is straightforward. Global predicates are predicates involving the state of various processes; the detection of such predicates means to find a global state in the execution path where the predicate holds.

Cooper and Marzullo introduced two predicate qualifiers to ease the detection of global predicates [Cooper 1991]. They are namely: *possibly*(ϕ) ($P(\phi)$) which is evaluated to true if ϕ holds for some observation; *definitely*(ϕ) ($D(\phi)$) evaluates to true if ϕ holds for all the observations. Those predicates are evaluated on the lattice of global states.

Detecting $D(\phi)$ is very expensive as it requires to examine all the linear extensions of the execution poset; in the worst case, detecting $P(\phi)$ is as expensive as detecting $D(\phi)$. Schwarz and Mattern presented an algorithm to reduce the cost of detecting $P(\phi)$ [Schwarz 1991]. The idea is to "navigate" through the event lattice, searching in the lattice, through one dimension, for a state where ϕ holds and changing the direction when this state is reached or due to causality requirements (when receiving a message). This sequential walk through the lattice is clearly a chain decomposition of the poset, which is done however on a process basis; indeed, the directions followed in the lattice are the processes axes. This algorithm could take benefit of the chain decomposition presented earlier, gaining in efficiency.

Charron-Bost et al. stated the following hierarchy of predicates, i.e. ($\phi \Rightarrow D(\phi) \Rightarrow P(\phi)$) and introduced a new kind of predicates for which $D(\phi)$ is equivalent to $P(\phi)$ enabling a reduced detection cost [Charron-Bost 1992]. These predicates are called *regular*. It is shown that stable predicates (S) are predicates for which $\phi = D(\phi) = P(\phi)$, while regular predicates are predicates for which only $D(\phi) = P(\phi)$ holds. Local predicates are proved to be regular; hence $\mathcal{S} \subset \mathcal{R} \subset \mathcal{P}$[6].

We introduce here two new predicates qualifiers, which fit in the predicates hierarchy enonced in [Charron-Bost 1992]. We derive then, from these new qualifiers, class of predicates, called *greedy predicates* and *strongly greedy predicates*.

Definition 6 **greedy** (ϕ) *($G(\phi)$) holds iff* $\forall c_g \in \mathcal{C}, \exists s \in c_g$ *for which* ϕ *is true,* c_g *being a greedy linear extension of the execution.*
strongly greedy (ϕ) *($SG(\phi)$) holds iff* $\forall c_{sg} \in \mathcal{C}, \exists s \in c_{sg}$ *for which* ϕ *is true,* c_{sg} *being a semi-strongly greedy linear extension.*

Proposition 4.1 *For a predicate* ϕ, $\phi \Rightarrow D(\phi) \Rightarrow SG(\phi) \Rightarrow G(\phi) \Rightarrow P(\phi)$ *holds.*

[6] $\mathcal{P}$ is the general class of global predicates

Proof: This follows from the definition of the G and SG qualifiers. □

Definition 7 *A predicate ϕ is said to be* **greedy** *if $P(\phi) \Rightarrow G(\phi)$. A predicate ϕ is said to be* **strongly greedy** *if $P(\phi) \Rightarrow SG(\phi)$.*

In an obvious way, local and stable predicates are greedy predicates, and are also strongly greedy predicates.

In order to improve the hierarchy $\mathcal{S} \subseteq \mathcal{G} \subseteq \mathcal{SG} \subset \mathcal{R} \subset \mathcal{P}$ into $\mathcal{S} \subset \mathcal{G} \subset \mathcal{SG} \subset \mathcal{R} \subset \mathcal{P}$, instances of greedy predicates which ae neither stable nor regular must be found. Since local predicates are regular and stable predicates follow causality, we devise that causal composition of local predicates are a good candidate to prove a strict hierarchy. Furthermore, we believe that there could be a relation between the jump number and the detection of greedy predicates. In other words, we think there must exist a correspondence between classes of posets and classes of predicates, which could be observed on a given execution. This is a direction worth pursuing.

5 Conclusion

We have shown in this paper that the arc diagram approach for representing distributed execution posets is a useful representation. On one hand, it characterizes correctly ordering relationships between events as vector time does, and on the other hand, it adequately summarizes the reachable global states of the distributed execution. One advantage of this approach is that the user does not need anymore to switch between different representations for handling these two abstraction levels. Also we showed that by using greedy chains (originally used for solving the jump number problem), one can obtain a chain decomposition of the event poset which does not rely on a classical process decomposition. These decompositions yield nearly optimal linear extensions, which can be viewed as the closest linear extensions that can be built with respect to the original poset. We introduced greedy predicates as an application of these optimal linear extensions, which appear to be a refinement of the predicates hierarchy introduced by Charron-Bost et al.

References

[Bonnet 1982] R. Bonnet and M. Pouzet. Linear extension of ordered sets. In I. Rival, editor, *Ordered Sets*, pages 125–170. D. Reidel Publishing Company, 1982.

[Charron-Bost 1992] Bernadette Charron-Bost, Carole Delporte-Gallet, and Hugues Fauconnier. Local and temporal predicates in distributed systems. Technical Report 92.36, LITP, 1992.

[Cooper 1991] R. Cooper and K. Marzullo. Consistent detection of global predicates. In *Proceedings ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 163–173, Santa Cruz, 1991.

[Diehl 1992a] C. Diehl, C. Jard, and X. Rampon. Reachability analysis on distributed execution. Technical Report 1720, INRIA, 1992.

[Diehl 1992b] Claire Diehl and Claude Jard. Interval approximations of message causality in distributed executions. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, pages 363–374, Cachan (France), 1992. Springer Verlag.

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

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

[Mattern 1988] Friedemann Mattern. Virtual time and global states in distributed systems. In *Proceedings of the International Parallel Conference on Distributed Algorithm*, pages 215–226. North-Holland, 1988.

[Schwarz 1991] Reinhard Schwarz and Friedemann Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. Technical Report 215/91, Department of Computer Science, University of Kaiserslautern, 1991.

[Sysło 1985] Maciej Sysło. A graph-theoretic approach to the jump number problem. In I. Rival, editor, *Graphs and Orders*, pages 185–215. D. Reidel Publishing Company, 1985.

[Sysło 1987] Maciej Sysło. Minimizing the jump number for partially-ordered sets: A graph-theoretic approach. *Discrete Mathematics*, 63:279–295, 1987.

[Valot 1993] Céline Valot. Characterizing the accuracy of distributed timestamps. In *Proceedings of the Third Workshop on Parallel and Distributed Debugging*, pages 43–52, San Diego, May 1993. ACM.

Unité de Recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

Unité de Recherche INRIA Lorraine Technopôle de Nancy-Brabois - Campus Scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 VILLERS LES NANCY Cedex (France)
Unité de Recherche INRIA Rennes IRISA, Campus Universitaire de Beaulieu 35042 RENNES Cedex (France)
Unité de Recherche INRIA Rhône-Alpes 46, avenue Félix Viallet - 38031 GRENOBLE Cedex (France)
Unité de Recherche INRIA Sophia Antipolis 2004, route des Lucioles - B.P. 93 - 06902 SOPHIA ANTIPOLIS Cedex (France)

EDITEUR
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

ISSN 0249 - 6399