Parallel N-free Order Recognition

Laurent Viennot
LITP/IBP
Université Paris 7 Denis Diderot
Case 7014, 2, place Jussieu
F-75251, Paris Cedex 05.
e-mail: lavie@litp.ibp.fr

Abstract

Parallel algorithms for recognizing and representing N-free orders are proposed for different models of parallel random access machines (PRAM). The algorithms accept as input a transitively reduced directed graph with n vertices and m edges. They respectively run in time $O(\log n)$ using $n + m$ processors in the EREW PRAM model and in constant time using n^2 processors in the CRCW PRAM model. Algorithms for distributed-memory machines are also proposed.

Key words: Parallel algorithms, recognition algorithms, PRAM, partial order, N-free order.

1. Introduction.

Due to the proverbial intractability (*i.e.* NP-completeness) of the majority of computational problems occurring in the algorithmic study of ordered discrete structures, much interest has been paid to classes of ordered sets that still admit efficient algorithms for otherwise intractable problems. The tractability of these classes is in most cases a consequence of rather strong structural properties not shared by arbitrary partial orders.

Many such algorithms have been developed by, for example, Mohring [Moh89], Golumbic [Gol80], Spinrad [Spi85], Pnuelli, Lempel and Even [PLE71], Papadimitriou and Yannakakis [PY79], Gabow [Gab81]. However, all of them are sequential algorithms. Like Bender, Gastaldo and Morvan [BGM93] who gave a parallel solution to interval order recognition, we are interested in developing parallel algorithms to complement the existing sequential algorithms. In this paper we focus on N-free orders.

N-free orders have been theoretically studied in depth for their numerous structural poperties [HJ85, Gri69, LM73, HB78, HN60]. One of their main and oldest applications is their use in project analysis, in particular in the techniques such as CPM or PERT, see *e.g.* [Elm77, MP64]. These techniques represent a

project by a directed graph in which the edges correspond to the activities of the project and vertices correspond to events (the completion of all activities entering the vertex). In order-theoretic terms, this so-called activity-on-edge representation or PERT-network is just the edge diagram of an N-free order. If the original partial order describing the technological precedence constraints of the project is not N-free, then dummy activities are added to make it N-free. Many techniques have been proposed for this task, *cf.* [Sys84, Sys85, Spi86] for further references.

The other major application of N-free orders arose with the investigation of the jump number. This classical parameter can be computed by a simple greedy algorithm in N-free orders [Riv82], and is also related to several structural properties of N-free orders.

The fastest known N-free recognition algorithms assume that the partial order P is given in transitively reduced form and construct an edge diagram if P is N-free. Their running time is $O(n+m)$, where n is the number of vertices and m is the number of edges in the transitive reduction of P. The first such algorithm is implicitly contained in the recognition algorithm for series-parallel partial orders in [VTL82]. The first "explicit" linear N-free recognition algorithm appeared in [Sys82]. Another important result is the algorithm of Ma and Spinrad [MS91] where no assumption on the input is made. It determines whether the transitive closure of a directed graph is an N-free order in $O(n+m_t)$ time where m_t is the number of edges of the transitive closure of the input.

In this paper we propose parallel algorithms for recognizing N-free orders, and parallel algorithms for determining edge diagram representations.

Section 2 introduces the basic definitions and properties of N-free orders, which will be exploited in our recognition algorithms. In section 3 we propose PRAM parallel N-free order recognition algorithms for two models of PRAM: with exclusive memory accesses or with concurrent memory accesses. In section 4 we propose PRAM parallel algorithms for constructing edge diagram representations. We present parallel recognition and representation algorithms for distributed memory machines in section 5.

2. Definition and properties of N-free orders.

A *partial order* will be denoted by $P=(V,<)$, where V is the (finite) *ground set* of *vertices* and $<$ is the *order relation*, *i.e.* an irreflexive and transitive relation whose pairs $(u,v)\in <$ are written as $u<v$ $(u,v\in V)$. If $u<v$ then u is called a *predecessor* of v and v is called a *successor* of u. If $u<v$ and there is no $w\in V$ with $u<w<v$ then v is said to *cover* u (denoted by $u\prec v$). We also say that v is an *immediate successor* of u (or u is an *immediate predecessor* of v). $\mathrm{ImSucc}(u)$ and $\mathrm{ImPred}(v)$ denote the set of all immediate successors and immediate predecessors of u.

Directed graphs may have parallel arcs but no loops. They are denoted by $G = (V, \mathcal{A})$, where V is the set of *vertices* and $\mathcal{A}$ is the set of *directed edges* or *arcs*. An arc $a \in \mathcal{A}$ is directed from its *tail* u to its *head* v. We write $a = (u, v)$. If there are parallel arcs, we consider that $\mathcal{A}$ is a multiset. A *sink* is a vertex which is not a head of any arc, and a *source* is a vertex which is not a tail of any arc.

A *dag* is a directed acyclic graph (possibly with parallel edges). It is *transitively closed* if $(u, v), (v, w) \in \mathcal{A}$ implies that $(u, w) \in \mathcal{A}$, and *transitively reduced* if $(u, w) \in \mathcal{A}$ implies that there is no $v \in V$ with $(u, v) \in \mathcal{A}$ and $(v, w) \in \mathcal{A}$. Every arc (u, w) violating this condition is called *transitive*.

Every partial order $P = (V, <)$ may be interpreted as a transitively closed dag with vertex set V and edge set $<$. Obviously, the transitively reduced form corresponds to the covering relation. A *Hasse diagram* of P is a drawing of this reduced form where the edges are implicitly directed from bottom to top (see Figure 1).

An order is *N-free* if its reduced form does not contain the subconfiguration N from Figure 1.(a) as induced subgraph.

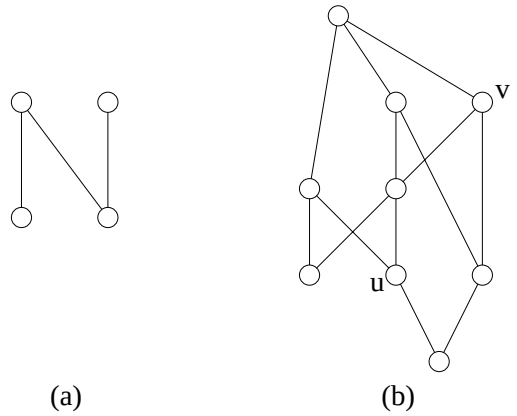


Figure 1. (a) The forbidden subconfiguration for N-free orders. (b) An example of N-free order represented by its Hasse diagram. The Hasse diagram is a full representation of the order since a vertex v is a successor of a vertex u if and only if there exists an always ascending path from u to v.

Every dag $G = (V, \mathcal{A})$ defines a partial order $P = (\mathcal{A}, <)$ over its arcs in the following sense: $a < b$ if and only if there exists a directed path from the head of a to the tail of b in G.

Notice that b covers a if and only if the head of a is the tail of b. The dag with vertex set $\mathcal{A}$ induced by the covering relation of P is called the *line-graph* of G. Such an order P is said to be *edge-induced* and G is called an *edge diagram* of P.

The following theorem (see [Moh89]) gives the fundamental structural properties of N-free orders needed for sequential recognition.

Theorem 1. *Given a partial order $P = (V, <)$, the following statements are equivalent:*

(1.1) P is N-free.

(1.2) For all $u, v \in V$, $\mathrm{ImSucc}(u) = \mathrm{ImSucc}(v)$ *or* $\mathrm{ImSucc}(u) \cap \mathrm{ImSucc}(v) = \emptyset$.

(1.3) P is edge-induced.

The optimal sequential recognition algorithm [Sys82] verifies property 1.2 by scanning incrementally each vertex and its set of immediate successors. It assumes that the order is given in transitively reduced form. Its running time is $O(|V| + |\prec|)$.

An equivalent graph-theoretic approach to Theorem 1 has been developed in [VTL82], *cf.* also [FGS85]. As we will see, it is more appropriate for parallel algorithmics. They define the class of *complete bipartite composite dags* (CBC dag for short) as the class of directed acyclic graph $G = (V, \mathcal{A})$ for which there exists a partition $B_1, \ldots, B_k$ of $\mathcal{A}$ such that:

(2.1) each B_i induces a complete bipartite subgraph of G (B_i is called a *bipartite component* of G),

(2.2) for each non-sink vertex v, all arcs leaving v belong to the same bipartite component,

(2.3) for each non-source vertex v, all arcs entering v belong to the same bipartite component.

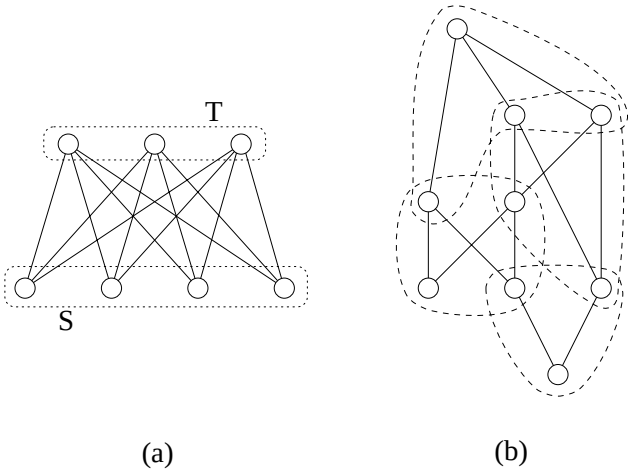


Figure 2. (a) A complete bipartite graph with source set S and sink set T. (b) The bipartite components of the transitive reduction of the N-free order represented in Figure 1.

Statements 2.2 and 2.3 are simply a condition of maximality of the bipartite components (see Figure 2). Using this definition, another characterization theorem is given:

Theorem 3 [VTL82]. *A dag is CBC if and only if it is the transitive reduction of an N-free order.*

Remarks: Let S_i and T_i respectively denote the source set and the sink set of the bipartite component B_i. Then for all $v \in T_i$, $\mathrm{ImPred}(v) = S_i$ and for all $u \in S_i$, $\mathrm{ImSucc}(u) = T_i$. Thus we have:

$$\{S_i,\ 1 \leq i \leq k\} = \{\mathrm{ImPred}(v),\ v \in V\}$$
$$\text{and} \quad \{T_i,\ 1 \leq i \leq k\} = \{\mathrm{ImSucc}(u),\ u \in V\}.$$

Each S_i (respectively T_i) will be called a *component source set* (respectively *component sink set*) of G whose bipartite component is B_i.

3. Parallel recognition algorithms.

In the following we develop two parallel algorithms for N-free order recognition. They are both based on Theorem 3 and they compute the bipartite components. They suppose that the input is given in transitively reduced form.

For the remaining of this paper, let n and m respectively denote the number of vertices and arcs of the input.

The first algorithm employs the *exclusive-read exclusive-write* (EREW for short) PRAM, where only one processor at a time can read from or write to a memory location. It runs in $O(\log n)$ time with $O(n+m)$ processors.

The second algorithm employs the *arbitrary concurrent-read concurrent-write* (arbitrary-CRCW for short) PRAM. In case of write conflict, only one arbitrary processor succeeds in writing its value. The second algorithm runs in constant time with $O(n^2)$ processors.

3.1 Data-structure independent algorithm.

The two parallel algorithms are based on the following data-structure independent algorithm:

Algorithm 1. N-free order recognition

Input: A transitively reduced graph $G = (V, \mathcal{A})$.

Output: **True** if the transitive closure of G is an N-free order and **false** if not.

Step 1. Let S_i and T_i respectively denote the source set and the sink set of the bipartite component B_i $(1 \leq i \leq k)$. Compute the bipartite components supposing that G is the transitive reduction of an N-free order as follows:

Select a vertex u_i in each component source set S_i using $\{S_i,\ 1 \leq i \leq k\} = \{\mathrm{ImPred}(v),\ v \in V\}$.

For all such vertex u_i,

identify $T_i = \mathrm{ImSucc}(u_i)$,
identify the B_i as the set of all arcs having their head in T_i.

Step 2. Check whether the three conditions 2.2, 2.3 and 2.1 for CBC dags are verified. If it is the case then G is the transitive reduction of an N-free order, else return false.

Remarks: Notice that in any case, the components computed in Step 1 form a partition of the arc set. Moreover they induce bipartite subgraphs of G because G is transitively reduced. Indeed, if it was not the case, a component B_i would contain two arcs of the form (u, v) and (v, w). It would also contains by construction the arc (u_i, v) and the arc (u_i, w) which would then appear to be a transitive arc. This would violate the fact that G is transitively reduced. Thus in Step 2, for condition 2.1 we just have to check that these subgraphs are complete.

Notice also that all the arcs entering a vertex $v \in T_i$ will be in the same bipartite component B_i by construction. Thus condition 2.3 will always be verified and we do not need to check it. On the other hand, when we successfully check, for some sink vertex u, that all the arcs leaving it are in the same bipartite component B_i (condition 2.2), we then know that $u \in S_i$. Thus the component source sets are computed at the same time.

As a final remark, let us explain how we will deal with disjoint subsets. The bipartite components are disjoint subsets of $\mathcal{A}$. Every B_i will be numbered by u_i. The bipartite components will be represented in array $\underline{B}$ such that an arc j is in B_i if and only if $\underline{B}[j] = u_i$. The S_i (respectively the T_i) are disjoint subsets of V. Every S_i (respectively T_i) will have same number as B_i. The S_i (respectively the T_i) will be analogously represented in an array $\underline{S}$ (respectively $\underline{T}$).

Theorem 4. *Algorithm 1 determines whether a transitively reduced graph G is N-free.*

The proof follows from theorem 3.

Proof: If the input G is the transitive reduction of an N-free order (*i.e.* a CBC dag), then the algorithm really computes its bipartite components in Step 1 and the three tests succeed in Step 2.

Anyway, no matter what partition of the arc set has been computed in Step 1, if the tests succeed in Step 2 then G is a CBC dag. If G is not the transitive reduction of an N-free order then we cannot have computed bipartite components in Step 1 and one of the tests will fail in Step 2. □

3.2 An EREW algorithm.

In the remaining of the paper, we mix up vertices and their numbers, more formally, we suppose that $V = \{1, \ldots, n\}$.

The first parallel N-free order recognition uses an *arc-array* data structure, where the arcs are explicitly stored as couples of vertex numbers in an array $\underline{A}$. During the algorithm, we sort couples of integer between 1 and n (the numbers of the vertices). As no efficient bucket sort running on EREW PRAM is known, we will always use the Cole Parallel Merge Sort [Col88] which can sort k elements in $O(\log k)$ time using k processors. These couples will be sorted according to the lexicographical order $<_{lex}$ defined by:

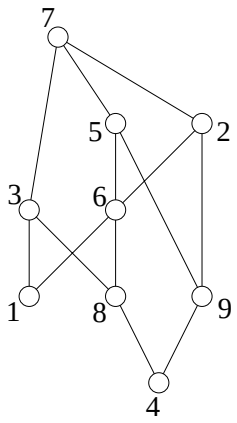
$$(u, v) <_{lex} (u', v') \Longleftrightarrow \begin{cases} \text{either } u < u' \\ \text{or} \qquad u = u' \text{ and } v < v'. \end{cases}$$

or according to the anti-lexicographical order $<_{anti}$ defined by:

$$(u, v) <_{anti} (u', v') \Longleftrightarrow (v, u) <_{lex} (v', u').$$

In a lexicographically (respectively anti-lexicographically) sorted array of couples, we call *blocks* the subarrays of consecutive couples having same first (respectively second) component (see Figure 3).

(a)



(b) | (1,3) (1,6) | (2,7) | (3,7) | (4,8) (4,9) | (5,7) | (6,2) (6,5) | (8,3) (8,6) | (9,2) (9,5) |

(c) | (6,2) (9,2) | (1,3) (8,3) | (6,5) (9,5) | (1,6) (8,6) | (2,7) (3,7) (5,7) | (4,8) | (4,9) |

Figure 3. (a) The N-free order represented in Figure 1 with numbered vertices. (b) The arc-array representation of its reduced form sorted lexicographically. The blocks which are the subarrays of couples having same first component are outlined. (c) The arc-array sorted anti-lexicographically. The blocks are the subarrays of couples having same second component.

In this first implementation of Step 1 of Algorithm 1, the chosen vertex in each component source set will be the one having minimal number. The idea will be to select these elements $u_1, \ldots, u_k$ without having computed the component source set $S_1, \ldots, S_k$ that respectively contain them. To do so, we anti-lexicographically sort the arc-array. Consider now the arcs of a block and let v be their common second component. Keeping only their tails, we obtain the sorted list of the elements of ImPred(v). Thus the first element of this list is the selected vertex of the component source set ImPred(v). Obviously this first element will be the same for any w such that ImPred(w) = ImPred(v). In the example of Figure 3, the selected vertices will be $6, 1, 2$ and 4. Then we can easily identify the corresponding T_i and B_i giving them number u_i.

The out-degree $D^+(u)$ of each vertex u of the input ($1 \leq u \leq n$) will be required in Step 2. We will verify that every B_i computed in Step 1 is complete by checking that all its sources have same out-degree. This is sufficient since $D^+(u_i) = |T_i|$ by construction. The out-degrees can easily be calculated in $O(\log n)$ time using for example, a lexicographical sorting of the arcs and a parallel prefix computation.

The algorithm is as follows:

Algorithm 2. EREW implementation

Input: A transitively reduced graph G.

Output: **True** if the transitive closure of G is an N-free order and **false** if not.

{ Implementation of Step 1 of Algorithm 1 }
1 Sort the arcs of array $\underline{A}$ anti-lexicographically.
2 **For all** $1 \leq j \leq m$ **do**
3 Let (u, v) be the arc in array position $\underline{A}[j]$.
4 **If** (u, v) is the first arc of its block (*i.e.* if the head of the arc in array position $\underline{A}[j-1]$ is different from v) **then** set $\underline{T}[v] := u$.
5 Set the bipartite component number of the arc (u, v) to $\underline{B}[j] := \underline{T}[v]$.

{ Implementation of Step 2 of Algorithm 1 }

{ Checking condition 2.2: are the arcs leaving each vertex in the same bipartite component? }
6 Sort the arcs lexicographically.
{ Check that in every block, all arcs have same bipartite component number: }
7 **For all** $2 \leq i \leq n$ **do**
8 Let (u_1, v_1) and (u_2, v_2) be the couples in array positions $\underline{A}[i-1]$ and $\underline{A}[i]$ respectively.
9 **If** $u_1 = u_2$ **then** ReturnValue[i] := $(\underline{B}[i-1] = \underline{B}[i])$
10 **Else** ReturnValue[i] := **true**
11 $\underline{S}[u_2] := \underline{B}[i]$
12 Let (u, v) be the couple in array position $\underline{A}[1]$, **set** $\underline{S}[u] := \underline{B}[1]$.

13 **If not**$(\bigwedge_{i=2}^{n} \text{ReturnValue}[i])$ **then return false**

{ Checking condition 2.1: are the bipartite components complete? }

14 Store the couples $(\underline{S}[u], D^+(u))$ lexicographically.

15 Check in the same way as previously that in every block all the couples have same second component.

16 **If** the two tests have succeeded **then return true**.

Theorem 5. *Algorithm 2 determines whether a transitively reduced graph G is N-free. It runs on EREW PRAM in $O(\log n)$ time using $n+m$ processors.*

Proof: Algorithm 2 is clearly equivalent to Algorithm 1. The concurrent read in line 5 can be implemented with a prefix computation in $O(\log n)$ time using m processors. All sorts and conjonctions can run in $O(\log n)$ time using $O(n+m)$ processors. The result follows. □

3.3 A CRCW algorithm.

We now propose a constant time algorithm. This is possible because we deeply use the arbitrary concurrent write ability to compute partitions instead of sorting operations. We will check that the bipartite components are complete thanks to the complementary graph: we will test if no arc is missing in any component. The use of the complementary graph induces a $O(n^2)$ workload. Algorithm 3 is based on an *adjacency-matrix* structure which allows to work on both the graph and its complement. The adjacency-matrix M for the input graph $G = (V, \mathcal{A})$ is defined as follows:

$$M[u,v] = \begin{cases} \textbf{true} \text{ if } (u,v) \in \mathcal{A} \\ \textbf{false} \text{ otherwise.} \end{cases}$$

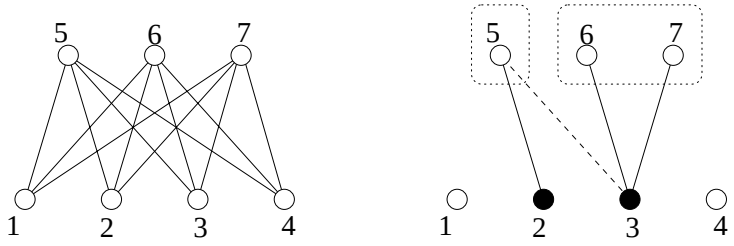
Notice that we can easily compute this representation from an arc-array data structure in constant time using n^2 processors.

To implement Step 1 of Algorithm 1, every bipartite component will be computed in two phases. Consider a component source set S_i. For all $v \in V$ such that $\text{ImPred}(v) = S_i$, we have to isolate the same vertex u_i. First we pick an arbitrary vertex $ST[v] \in \text{ImPred}(v)$ (lines 1-2). In each component source set, we keep only the picked vertex having minimal number (lines 3-6). Then for each such vertex u_i, we respectively compute T_i and B_i as the immediate successors set of u_i (lines 7-8) and as the set of the arcs entering T_i (line 9). Figure 4 illustrates lines 1-6 on an example.

Algorithm 3. CRCW implementation

Input: A transitively reduced graph G.

Output: **True** if the transitive closure of G is an N-free order and **false** if not.



(a) (b)

Figure 4. Selecting a source in each bipartite component. (a) A bipartite component with numbered vertices. (b) Here, an arbitrary concurrent write has determined $ST[5] = 2$ and $ST[6] = ST[7] = 3$ (line 2). Thus we mark 2 and 3 (line 4). The existence of the arc $(3, 5)$ proves that 3 and $2 = ST[5]$ are in the same source set $\mathrm{ImPred}(5) = \mathrm{ImPred}(6) = \mathrm{ImPred}(7)$ and thus that 3 is not the marked vertex with minimal number in this source set. Hence 3 will be unmarked (line 6).

```
{ Implementation of Step 1 of Algorithm 1 }
1   For all 1 ≤ u, v ≤ n do if M[u, v] then
2         ST[v] := u                          { arbitrary concurrent write }
3   For all 1 ≤ u ≤ n do Marked[u] := false
4   For all 1 ≤ v ≤ n do Marked[ST[v]] := true
5   For all 1 ≤ u, v ≤ n do if M[u, v] then
6         If u > ST[v] then Marked[u] := false    { concurrent write of same
            value }
7   For all 1 ≤ u, v ≤ n do if M[u, v] then
8         If Marked[u] then T[v] := u
9         B[u, v] := T[v]
{ Implementation of Step 2 of Algorithm 1 }
10  Result := true
{ Checking condition 2.2: are the arcs leaving each vertex in the same bipartite
  component? }
{ Compute the source set of every bipartite component: }
11  For all 1 ≤ u, v ≤ n do if M[u, v] then
12        S[u] := B[u, v]                     { arbitrary concurrent write }
13        If S[u] ≠ B[u, v] then Result := false   { check that the previous
            concurrent write was indeed a concurrent write of the same value }
{ Checking condition 2.1: are the bipartite components complete? }
{ Check that no arc is missing in any component: }
14  For all 1 ≤ u, v ≤ n do if not(M[u, v]) then
15        If S[u] and T[u] are defined and S[u] = T[v] then Result := false
```

16 **return** Result.

Theorem 6. *Algorithm 3 determines whether a transitively reduced graph G is N-free. It runs on CRCW PRAM in constant time using n^2 processors.*

Proof: The time and number of processors bounds are clear.

In line 13, all arcs (u, v) leaving u write their bipartite component number in $S[u]$ and check in line 14 that they have all written the same value, checking by the way condition 2.2. Notice that two vertices u and v are sources of the same bipartite component if and only if $S[u] = S[v]$.

We check that every bipartite component B_i joining vertices from S_i with T_i is complete by testing whether an arc (u, v) verifying $u \in S_i$ and $v \in T_i$ is missing in line 17.

We still have to prove that the implementation of Step 1 of Algorithm 1 is correct. We suppose that the input is a CBC dag as we do not matter what is computed otherwise.

Consider a bipartite component B_i joining vertices from S_i with T_i. For all $v \in T_i$, $ST[v]$ is set in line 2 to an arbitrary vertex in $S_i = \mathrm{ImPred}(v)$. The vertices of the form $ST[v]$ are marked in line 4. Let u_i be the marked vertex with minimal number in S_i and let v_i be a vertex such that $ST[v_i] = u_i$.

Consider now line 6. For all arc (u_i, v) leaving u_i, $v \in T_i$ and thus we have $ST[v] \geq u_i$. Hence u_i is still marked after this line. On the other hand for all $u \in S_i$ different from u_i, the arc (u, v_i) is present in the graph since the bipartite component is complete by hypothesis. As $u > ST[v_i]$, Marked$[u]$ is set to **false**. After line 6, u_i is the only marked vertex in S_i. Thus the write in line 8 is exclusive and $T_i = \mathrm{ImSucc}(u_i)$ is the set of vertices v verifying $T[v] = u_i$. The arcs (u, v) of B_i are those entering a vertex in T_i as computed in line 9. They can be identified by $B[u, v] = u_i$ the number of B_i. □

4. Parallel algorithms for constructing edge diagrams.

In the following section, we provide algorithms which determine an edge diagram edge-inducing a given N-free order.

4.1 Properties of the bipartite components.

Let $G = (V, \mathcal{A})$ be the transitive reduction of an N-free order P. By theorem 3, G is a CBC dag. Let $B_1, \ldots, B_k$ be the partition of its arc set in bipartite components. For every B_i, we denote by S_i the set of sources of B_i and by T_i the set of sinks of B_i.

Let T_0 be the source set of G and S_∞ be the sink set of G. Thus $T_0, T_1, \ldots, T_k$ and $S_1, \ldots, S_k, S_\infty$ form two partitions of V. In addition, we introduce two "virtual" bipartite components B_0 and B_∞. We formally set that T_0 is the component sink set of B_0 and that S_∞ is the component source set of B_∞.

Now consider the dag $R = \left(\{B_0, B_1, \ldots, B_k, B_\infty\}, \widetilde{V}\right)$ defined as follows. Each vertex $v \in V$ is associated to an arc $\widetilde{v} \in \widetilde{V}$ of R ($\widetilde{V}$ is a multiset):

(7) if $v \in T_i$ and $v \in S_j$ then $\widetilde{v} = (B_i, B_j)$ is an arc from B_i to B_j.

Notice that R may have parallel arcs as there may be several vertices in $T_i \cap S_j$.

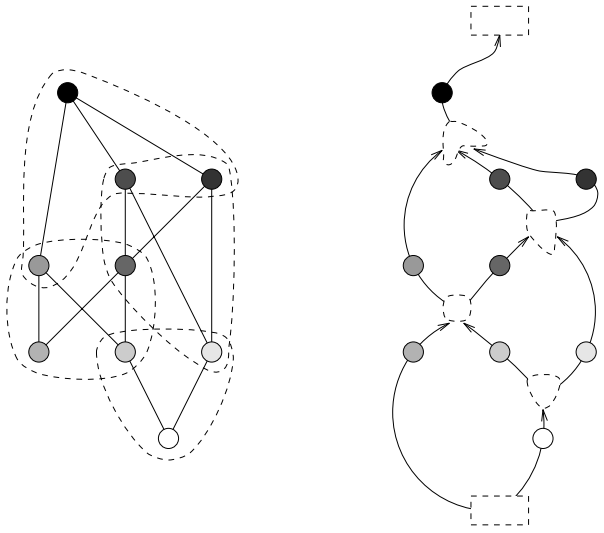


Figure 5. (a) The N-free order of Figure 1 represented by its Hasse diagram where the bipartite components are outlined. (b) Its unique edge diagram which has a single source and a single sink. Its vertices are the bipartite components represented in a (plus a source and a sink). Each arc is associated to a vertex of the N-free order.

Theorem 8 [VTL82]. *The dag R is an edge diagram of P.*

Proof: G is the line graph of R as its arcs (u, v) are those verifying $u \in S_j$ and $v \in T_j$ for some $j \in \{1, \ldots, k\}$ and we can write $u = (B_i, B_j)$ and $v = (B_j, B_k)$ for some $i, k \in \{0, 1, \ldots, k, \infty\}$. □

Notice that R is the only edge diagram of P which has a single source and a single sink.

4.2 Algorithms.

Parallel algorithms which determine an arc array representation of a corresponding edge diagram given an N-free order can easily be obtained from the recognition algorithms we have proposed.

Suppose that the input is the reduced form of an N-free order. We have already mentioned the fact that these algorithms compute the S_i and the T_i ($1 \leq i \leq k$). In addition, let us suppose that at the beginning of the algorithms, arrays $\underline{T}$ and $\underline{S}$ are respectively initialized to 0 and ∞. The sources (respectively the sinks) are the only vertices for which $\underline{T}$ (respectively $\underline{S}$) is not computed. Thus at the end of the algorithm, $\underline{T}$ (respectively $\underline{S}$) represents the partition $T_0, T_1, \ldots, T_k$ (respectively $S_1, \ldots, S_k, S_\infty$) where T_0 has number 0 (and B_∞ has number ∞). Thus by adding the following line, we obtain parallel algorithms for constructing an arc-array $\underline{\widetilde{V}}$ representing the edge diagram defined by 7 without changing the previous complexities.

(9) **For all** $1 \leq u \leq n$ **do** $\underline{\widetilde{V}}[u] := (\underline{T}[v], \underline{S}[v])$.

Theorem 10. *Algorithms 2 and 3 enhanced by (9) determine whether a transitively reduced graph is the transitively reduced form of an N-free order and construct (if the answer is yes) a corresponding edge diagram. They respectively run on EREW PRAM in $O(\log n)$ time using $n + m$ processors and on CRCW PRAM in constant time using n^2 processors.*

Proof: The proof follow from Section 4.2.

Remarks: An edge diagram R gives a sublinear representation of the corresponding N-free order P as its size is $O(n)$. This representation allows to answer in constant time to the query "Is u an immediate predecessor of v?" by testing whether the head of $\widetilde{u}$ is the tail of $\widetilde{v}$.

The transitively closed form of P can be obtained from the transitive closure of R as the query "Is u a predecessor of v?" can be answered by testing whether there exists in R a directed path from the head of $\widetilde{u}$ to the tail of $\widetilde{v}$. This is interesting since in general R is much smaller than the reduced form of P.

5. Algorithms for distributed-memory machines.

Notice that Algorithm 2 is composed of several sorting and partial sum routines. Because sorting algorithms and partial sum algorithms have been studied intensively on many distributed-memory architectures, it is a relatively small step to write distributed-memory algorithms for recognizing N-free orders and constructing edge diagrams.

Consider a hypercube architecture. The sorting procedure for hypercube with the best complexity, proposed by Cypher and Plaxton [CP90], runs in time $O(\log n(\log\log n)^2)$ on n processors. The partial sum algorithm for hypercube introduced by Nassimi and Sahni [NS81] runs optimally in time $O(\log n)$. Therefore, on a hypercube we can implement Algorithm 2 to run on m processors in time $O(\log n(\log\log n)^2)$.

On a $O(\sqrt{n})$ by $O(\sqrt{n})$ mesh of processors, sorting procedures with the best complexity—for example, the rotate sort of Gafni and Marberg [MG87] or the bitonic sort of Batcher [Bat68]—run optimally in time $O(\sqrt{n})$. A partial sum also runs optimally in time $O(\sqrt{n})$. Therefore on a $O(\sqrt{n})$ by $O(\sqrt{n})$ mesh of processors we can implement Algorithm 2 optimally in time $O(\sqrt{n})$.

6. Perspectives.

In this paper we have proposed PRAM and distributed-memory algorithms for recognizing N-free orders and constructing edge diagrams. Note that our algorithms only accept transitively reduced directed acyclic graphs as input. It is a general problem to develop parallel recognition algorithms which do not suppose that the input is transitively reduced or transitively closed, and which do not require transitive reduction nor transitive closure computations. As far as we know, this is still an open problem for many other classes of orders and even for very simple ones like the class of total orders.

7. Acknowledgments.

This paper is intentionally structured as the article of Bender, Gastaldo and Morvan [BGM93].

References.

[Bat68] K. E. Batcher. Sorting networks and their applications. In *Spring Joint Computer Conf.*, pages 307–314, 1968.

[BGM93] M. Bender, M. Gastaldo, and M. Morvan. Parallel interval order recognition and construction of interval representation. Technical Report RR 93-27, LIP ENS-Lyon, September 1993. To appear in Theoretical Computer Science.

[Col88] Richard Cole. Parallel merge sort. *SIAM J. Comput.*, 17(4), August 1988.

[CP90] R. Cypher and C. G Plaxton. Deterministic Sorting in Nearly Logarithmic Time on the Hypercube and Related Computers. In *22nd Annual Symposium on Theory of Computing*, pages 193–203, October 1990.

[Elm77] S. E. Elmaghraby. In *Activity Networks*, Wiley, New York, 1977.

[FGS85] U. Faigle, G. Gierz, and R. Schrader. Algorithmic approaches to setup minimization. *SIAM J. Comput.*, 14:954–965, 1985.

[Gab81] H. Gabow. A linear time recognition algorithm for interval dags. *Information Processing Letters*, 12(1):20–22, February 1981.

[Gol80] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

[Gri69] P. Grillet. Maximal chains and antichains. *Fund. Math.*, 65:157–167, 1969.

[HB78] R. L. Hemminger and L. W. Beineke. Line graphs and line digraphs. In L. W. Beineke and R. J. Wilson, editors, *Selected topics in graph theory*, pages 271–305, London, 1978. Academic Press.

[HJ85] M. Habib and R. Jegou. N-free posets as generalizations of series-parallel posets. *Discrete Appl. Math.*, 12(3):279–291, 1985.

[HN60] F. Harary and Z.R. Norman. Some properties of line digraphs. *Rend. Circ. Math.*, 9(161-168), 1960. Palermo.

[LM73] B. Leclerc and B. Monjardet. Orders "c.a.c.". *Fund. Math.*, 79:11–22, 1973.

[MG87] J. M. Marberg and E. Gafni. Sorting in constant number of row and column phases on a mesh. In *Proceedings of the Allerton Conference on Computing, Communication and Control*, pages 603–612, 1987.

[Moh89] Rolf H. Mohring. Computationally tractable classes of ordered sets. In I. Rival, editor, *Algorithms and Order*, pages 105–193. Kluwer Acad. Publ., Dordrecht, 1989.

[MP64] J. J. Moder and C. R. Phillips. In *Project Management with CPM and PERT*, Reinhold, New York, 1964.

[MS91] T. H. Ma and J. Spinrad. Transitive closure for restricted classes of partial orders. *Order*, 8(2):175–183, 1991.

[NS81] D. Nassimi and S. Sahni. Data broadcasting in simd computers. *IEEE Trans on Computers*, 30(2):101–107, 1981.

[PLE71] A. Pneuili, A Lempel, and W. Even. Transitive orientation of graphs and identification of permutation graphs. *Canad. J. math*, 23:160–175, 1971.

[PY79] C. H. Papadimitriou and M. Yannakakis. Scheduling interval-ordered tasks. *SIAM J. Comp.*, 8:405–409, 1979.

[Riv82] I. Rival. Optimal linear extensions by interchanging chains. *Proc. Amer. Math. Soc.*, 89:387–394, 1982.

[Spi85] J. Spinrad. On comparability and permutation graphs. *SIAM J. Comput.*, 14:658–670, 1985.

[Spi86] J. Spinrad. The minimum dummy task problem. *Networks*, 16(3):331–348, 1986.

[Sys82] M. M. Syslo. A labeling algorithm to recognize a line digraph and output its root graph. *Inform. Processing Letters*, 15:241–260, 1982.

[Sys84] M. M. Syslo. On the computational complexity of the minimum-dummy-activities problem in a pert network. *Networks*, 14:37–45, 1984.

[Sys85] M. M. Syslo. A graph theoretic approach to the jump number problem. In I. Rival, editor, *Graphs and Order*, pages 185–215, Reidel, Dordrecht, 1985.

[VTL82] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series-parallel digraphs. *SIAM J. Comput.*, 11:298–314, 1982.