Generating minimal interval extensions

Michel MORVAN & Lhouari NOURINE

Juin 1992

R.R. LIRMM n° 92-015

Generating minimal interval extensions

Michel MORVAN & Lhouari NOURINE

Juin 1992

R.R. LIRMM n° 92-015

Generating Minimal Interval Extensions

M. Morvan, L. Nourine

Département Informatique Fondamentale
(LIRMM, CNRS & Université Montpellier II),
860, rue de Saint-Priest
F 34090 Montpellier, France.
e.mail : {morvan, nourine}@crim.fr

Abstract: The aim of this paper is to present an algorithm that generates the set of minimal interval extensions of an order P in $O(n^2w|I_{min}(P)|)$, where n is the number of elements in P, and w is its width. The same algorithm can be modified to construct the lattice of maximal antichains *AM*(P) in $O(nw^2|AM(P)|)$.

Keywords: Interval Orders, Minimal interval extension, Lattice of maximal antichains.

1 Introduction

Interval extensions and related problems are of interest to computer scientists because of their relation to distributed sheduling problems. One can modelize the time concept in distributed executions in which many processors comunicate by sending and receiving messages. For example, there is a natural order associated to a distributed execution called the "causal order". This order is in general not an interval order, but in any "real execution", it is possible to associate any event to its duration interval; that is the time interval corresponding to its execution. The such obtained order on the intervals (verifying that an interval is smaller than another if it is on its left) is an interval extension of the causal order. These interval extensions of the causal order that are minimal are used to debug or analyze actual distributed programs, rather than the causal order itself (see Habib et al[8] and Diehl et al[6]).

From a theoretical point of view, these minimal interval extensions of an order can be seen as natural generalizations of linear extensions since the set of linear extensions of an order P is in bijection with the set of maximal chains of the antichains lattice of P(see Bonnet et al[3]). Also the set of minimal interval extensions of P is in bijection with the set of maximal chains of the maximal antichains lattice of P(see Habib et al[7]). On the other hand, linear extension of a partial ordered set P, extends P to a total order, when minimal interval extension, extends P to an interval order.

In Section 2 we will give formal definitions and characterization theorems concerning interval orders and related topics.

In Section 3 we will describe the strategy used to solve the problem.

In Section 4 we will present our algorithm and analyze its complexity.

2 Definitions

A poset (or partially ordered set) P is an irreflexive, asymmetric and transitive relation on a set X denoted by $P=(X,<_p)$. When necessary, we should denote P by (X,E) where $E \subseteq X^2$ and $(x,y) \in E$ iff $x<_p y$. All the orders considered in this paper are finite. Two elements x and y are said to be comparable and incomparable respectively if $x<_p y$ respectively $x \not<_p y$ and $y \not<_p x$, denoted by $x \|_p y$. We say that y covers x noted $-<_p$ iff $x<_p y$ and there is no z such that $x<_p z<_p y$; if y covers x, we say that x is an immediate predecessor of y. Consider Y a non-empty subset of P, the restriction of X to Y is called a chain if the elements of any pair of Y are comparable in P. If X=Y, we call P a linear order (also a total order). If $\tau=x_1,\ldots,x_n$ is a total ordering of the elements of P, τ is called a linear extension of P if $x<_p y$ implies $x<_\tau y$. If the elements of Y are pairwise incomparable, Y is called an antichain. The maximal length of an antichain is denoted by w and called the width of P. We say that an antichain A is maximal if another antichain B that containing A does not exist. Given the set of all maximal antichains $\mathbb{A}$ of an order P, and two maximal antichains A and B, we say that $A<_T B$ iff $\forall\ x \in A\backslash B$, $\exists\ y \in B$ such that $x<_p y$. $AM(P)=(\mathbb{A}, <_T)$ is a lattice called the maximal antichains lattice. Note that $A -<_T B$ if and only if $\forall\ x \in A\backslash B$, $\forall\ y \in B\backslash A$ $x -<_p y$ (ie. A\B and B\A form a complete bipartite order in the transitive reduction of P).

Let $Pred(x)=\{y \in X \mid y<_p x\}$ be the predecessor set of x. We define the equivalence relation $\approx$ over X by $x \approx y$ iff $Pred(x)=Pred(y)$. We denote by $P/\approx$ the quotient of P related to $\approx$. When $x \in P/\approx$, we denote by Pred(x) the predecessor set in P of the elements of the equivalence class x. Let $Pred^*=\{Pred(x) \mid x \in P/\approx\}$. The relation $\subset$ (strict inclusion) over Pred* is a partial order denoted by PRED(P). We define IMPRED(P) with the same definitions as PRED(P), but also with the immediate predecessor sets.

An order $P=(X,<_p)$ is called an interval order if there exists a function mapping each x in X to an interval I_x of the real line such that $x<_p y$ if and only if I_x lies totally to the left of I_y.

In order to introduce the notion of minimal interval extension, we give the following well known characterization theorem of interval orders.

Theorem 1: *For an order $P=(X,<_p)$, the following statements are equivalent:*

(1) P is an interval order.

(2) P does not contain any suborder isomorphic to "2+2" (see example 1(P_1)).

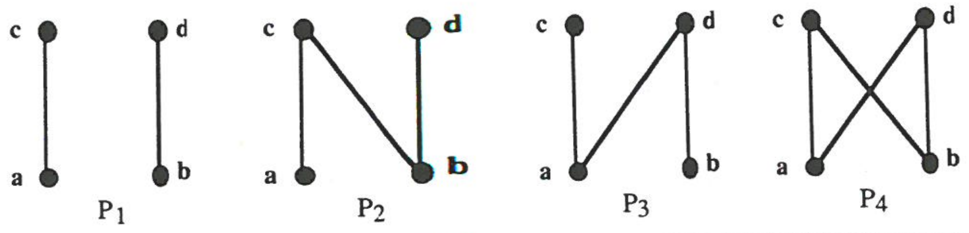
(3) $PRED(P)=(Pred^*,\subset)$ is a total order.

(4) The maximal antichains of P can be linearly ordered such that, for every element v, the maximal antichains containing v occur consecutively.

(5) $AM(P)=(\mathcal{A},<_T)$ is a total order.

Definition 1: Let P=(X,E) be an order. $I=(X,E_I)$ is an *interval extension* of P if I is an interval order and $E \subseteq E_I$. Furthermore I is called *minimal* if for every $I'=(X,E_{I'})$ with $E \subseteq E_{I'} \subset E_I$, I' is not an interval order.

Example 1: Consider the poset P_1 of figure 1. The posets P_2, P_3 and P_4 are interval extensions of P_1, but P_2 and P_3 are minimal whereas P_4 is not.



The following theorem, due to Habib et al[7] gives a characterization of the set of minimal interval extensions of an order.

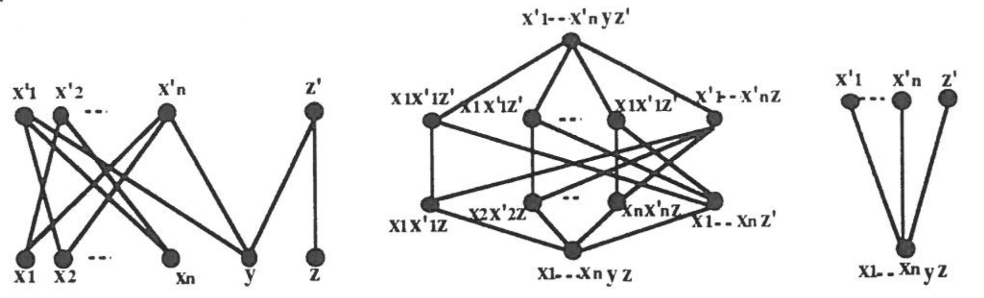
Theorem 2: *Let $P=(X,<_p)$ a poset. Then there is a one-to-one mapping between the maximal chains of AM(P) and the minimal interval extensions of P.*

3 Strategies

The purpose of this paper is to give an algorithm that generates all minimal interval extensions of a partial order. Since it is based on the previous characterization theorem, our algorithm can be easily modified to construct the maximal antichains lattice.

The generation problem has been already studied by Habib et al[9]. Their algorithm is based on another characterization theorem coming from property (3) of theorem 1. Their strategy is to generate all linear extensions of PRED(P) and test which of them correspond to a minimal interval extension. Since in the general case the number of linear extensions of PRED(P) may be larger than $|I_{min}(P)|$ (see example 2), many unnecessary computations are done. In the present paper, the use of theorem 2, allows us to generate only good minimal interval extensions, by simply generating maximal chains of $\mathcal{A}M$(P). These maximal chains will be generated by working on a graph based on IMPRED(P).

Example 2:



P AM(P) PRED(P)

The number of linear extension of PRED(P) is $(n+1)!$, but the number of maximal chains in *AM*(P) is 3n.

To compute maximal chains of *AM*(P), we must efficiently generate the set of all maximal antichains covering any maximal antichain A in *AM*(P) denoted by **Covers**(A). Note that is the same idea used by Knuth et al[11], for generating linear extensions. The linearity of the Knuth's algorithm comes from the fact that when an element becomes minimal, it remains minimal until it is chosen; this however is not our case. It is more interesting to work with the immediate predecessors because good complexity is obtained since the number of immediate predecessors of a vertex is bounded by the width of the order.

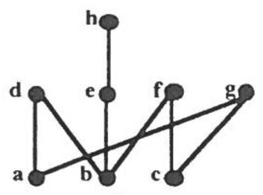
To understand the following note that :
$\forall x,y \in P$ $\mathrm{Pred}_P(x)=\mathrm{Pred}_P(y)$ iff $\mathrm{Impred}_P(x)=\mathrm{Impred}_P(y)$
If $\mathrm{Impred}_P(x)\subset\mathrm{Impred}_P(y)$ then $\mathrm{Pred}_P(x)\subset\mathrm{Pred}_P(y)$
So Pred*=Impred* and IMPRED(P) is a subgraph of PRED(P).

In order to obtain a good complexity a graph must be maintened to give a similar property, such as the one Knuth uses. This graph is associated to each maximal antichain A of *AM*(P) and allows us to produce efficiently **Covers**(A). Furthermore, the graph associated with to antichain of **Covers**(A) can be obtained by few transformations on the graph associated to A.

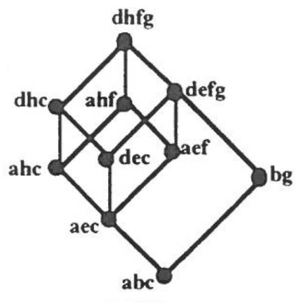
This graph denoted by $G_A=(U,V,\ell)$ is labeled and defined as follows:
For an order P=(X,E), given a maximal antichain A, we define the restriction of P to the set of elements greater or equal to an element of A by P_A (P_A is sometimes called the filter associated to A).
$U=(P_A/\approx)\backslash\{A\}$.
Let $x,y \in U$, $(x,y)\in V$ iff $\exists\, x_1\in x, y_1 \in y$
(i) $\mathrm{Impred}_P(x_1)\cap\mathrm{Impred}_P(y_1)\neq\emptyset$ or
(ii) $x_1 -\!\!< y_1$ or $y_1 -\!\!< x_1$.
(two mutually exclusive cases).
If $\mathrm{Impred}_P(x)\cap\mathrm{Impred}_P(y)\neq\emptyset$ $\ell(x,y)=\mathrm{Impred}_{P_A}(x) - \mathrm{Impred}_{P_A}(y)$.
$\ell(x,y)=\emptyset$ if $\exists\, x_1\in x, y_1\in y \mid x_1 -\!\!< y_1$
$\ell(x,y)=*$ if $\exists\, x_1\in x, y_1\in y \mid y_1 -\!\!< x_1$.
We define $\mathrm{Min}(G_A)=\{x\in U \mid \forall\, y\in U,\ (y,x)\in V \Rightarrow \ell(y,x)\neq\emptyset\}$.

Remark: If we consider the subgraph G'_A of G_A induced by $V'=\{(x,y)\in V \mid \ell(x,y)=0\}$ then $G'_A=\mathrm{PRED}(P_A)\backslash\{A\}$.
Notice that $\mathrm{Min}(G'_A)=\{x\in U \mid \nexists\, y\in U, (y,x)\in V \Rightarrow \ell(y,x)=\emptyset\}$.

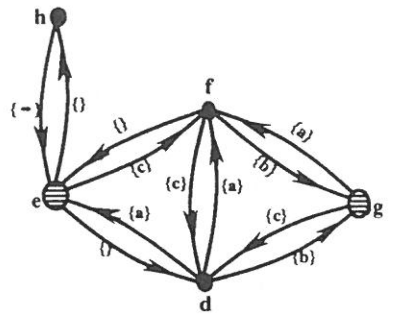
Example 3:
The following example shows the construction of the graph $G_{(abc)}$ for the order P. The labels of edges are represented by "{ }". For example the label $\ell(f,d)=\{c\}$ and $\ell(e,d)=\{\,\}$. The minimal elements of G'_A are indicated by a dotted circle.



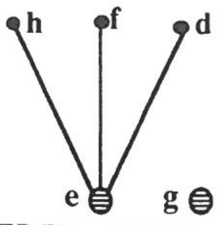
P



AM(P)



$G_{(abc)}$



$G'_{(abc)}$=PRED($P_{(abc)}$\(abc))

Theorem 3: *There is a one-to-one mapping between* ***Covers****(A) and Min(G'_A).*

Proof:

We are going to associate to each B in **Covers**(A) the unique element C in Min(G'_A) defined by C=B\A.

⇒ Let B∈ **Covers**(A), then B is a maximal antichain and by definition we have

$\forall$ x ∈ A\B, $\forall$ y ∈ B\A x$-\!<_p$y. We prove that C=B\A ∈Min(G'_A).

So we must show that C is an element of G_A and that C is minimal.

The definition of B and P_A shows immediately that :

$\forall$ x,y ∈ C Impred$_{P_A}$(x)=Impred$_{P_A}$(y) so $\exists$ C'| C⊆C'∈ ($P_A/\approx$), and since C∩A=∅ then C∈ G_A.

Now suppose that C is not minimal then $\exists$ Y∈ G_A | ℓ(Y,C)=∅. Thus $\exists$ y∈Y, x∈C such that Impred$_{P_A}$(y)⊂ Impred$_{P_A}$(x) implies A -<(A\Impred$_{P_A}$(y))∪ Y) <B. Thist is contradictory to B covers A.

⇐ Let C∈Min(G'_A), we prove that B=(A\Impred$_{P_A}$(C))∪ C) ∈ **Covers**(A). To do this we will first show that B is a maximal antichain. It is evident that B is an antichain.

Suppose that B is not maximal then $\exists$ B'∈ *AM*(P) such that B⊂B'. So, there is an x∈C'∈Min(G'_A)\B such that x is incomparable to B, then x is incomparable to B\C. But A is maximal, so $\exists$ y∈ Impred$_{P_A}$(C) such that y-<x. Then Impred$_{P_A}$(C')⊆ Impred$_{P_A}$(C) this is contradictory to x∉ C and C∈ Min(G_A).

Now let us prove that B ∈ **Covers**(A). Suppose that there is B' such that A-<B'<B.

Let x∈ B'\B, then Impred$_{P_A}$(x)∩(A\Impred$_{P_A}$(X))=∅ , and Impred$_{P_A}$(x)⊂Impred$_{P_A}$(C), which is contradictory to C∈ Min(G_A). ◊

Now we are going to show how to obtain the graph associated to each element of **Covers**(A) by a transformation of the graph G_A. To build a graph corresponding to B∈ **Covers**(A), we will start with the minimal element C of G_A associated to B. The idea is to delete all the vertices of Impred$_{P_A}$(C)∩A from every edge where they appear as labels and to join vertices of G_A that have now the same immediate predecessors sets in P_B. This can be expressed formally as follows.

Let A∈ *AM*(P) and C∈ Min(G_A) defined by C=B\A.

We define $\varphi_C(G_A)=(U_C,V_C,\ell_C\}$ by

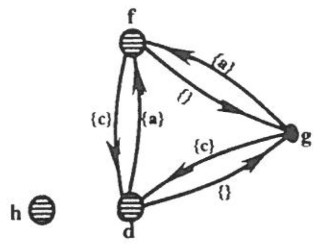
a) x∈U_C iff x=∪ x_i such that (1) $\left\{ \begin{array}{l} x_i \in U\backslash\{C\}\ \forall\ i \text{ and} \\ \forall\ i,j\ \ \ell(x_i,x_j)\backslash \text{Impred}_{P_A}(C)=\emptyset \\ \text{and x is maximal for this property.} \end{array} \right\}$

b) Let x,y ∈U_C then (x,y) ∈V_C iff $\exists$ x_1∈x, y_1∈y such that (x_1,y_1)∈V.

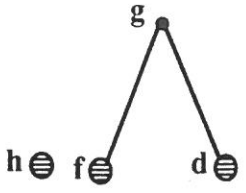
c) Notice that if (x,y)∈ V_C then ℓ(x_i ,y_j)\ Impred$_{P_A}$(C) is equal for every x_i∈x, y_j∈y, we define ℓ_C(x,y)=ℓ(x_i,y_j)\ Impred$_{P_A}$(C) for x_i ∈x, y_j∈y.

Example 4:

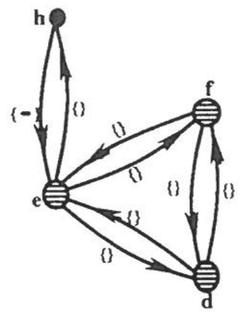
This example shows how to compute φ(G_A). The computing of φ_g(G_{abc}) shows the case of gathered elements. We will see later that it is not necessary to gather them.



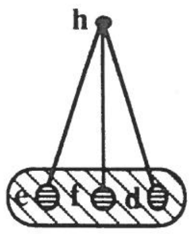
$\varphi_e(G_{abc})=G_{(aec)}$



$G'_{(aec)}=PRED(P_{(aec)}\backslash(aec))$



$\varphi_g(G_{abc})=G_{(bg)}$



$G'_{(bg)}=PRED(P_{(bg)}\backslash(bg))$

We denote by $\varphi(G_A)$ the set $\{\varphi_C(G_A) \mid C\in \mathrm{Min}(G_A)\}$. We are now going to prove that $\varphi(G_A)$ is the set of graphs associated to the elements of **Covers**(A).

Theorem 4: $G_{\mathbf{Covers}(A)}=\{G_B \mid B\in \mathbf{Covers}(A)\} = \varphi(G_A)$.

Proof:

a)Let $G\in\varphi(G_A)$. There is $C\in \mathrm{Min}(G_A)$ such that $G=\varphi_C(G_A)$. We are going to prove that $G=G_B$ where $B\in$**Covers**(A)

Let $B=\{A\text{-}\mathrm{Impred}_{P_A}(C)\}\cup\{C\}$. Let $G_B=(U_B,V_B,\ell_B)$.

$U_B=\{\ x \mid x=\cup\, x_i\ ,\ x_i\in U$ and $\forall$ i,j $(i\neq j)$ $\mathrm{Impred}_{P_A}(x_i)\backslash\mathrm{Impred}_{P_A}(x_j)\subseteq A\backslash B$ and

x is maximal for this property.}

So we have $U_B=U_C$ by definition. We also have the following equivalences:

$(x,y)\in V_C \Leftrightarrow\ \exists\ x_i\in x,y_j\in y \mid (x_i,y_j)\in V$

$\Leftrightarrow\ \exists\ x_i\in x,y_j\in y \mid (\mathrm{Impred}_{P_A}(x_i)\cap \mathrm{Impred}_{P_A}(y_j))\neq\emptyset$ or $(x_i\text{-}\!<y_j$ or $y_j\text{-}\!<x_i)$

$\Leftrightarrow\ (\mathrm{Impred}_{P_A}(x)\cap \mathrm{Impred}_{P_A}(y))\neq\emptyset$ or $(\exists\ x_i\in x,y_j\in y\ \ x_i\text{-}\!<y_j$ or $y_j\text{-}\!<x_i)$

$\Leftrightarrow\ (x,y)\in V_B$.

So $V_B=V_C$. and we conclude that $G=G_B$.

b)Let $G\in G_{\mathbf{Covers}(A)}$. There is $B\in$ **Covers**(A)such that $G=G_B$.We are going to prove that there is $C\in \mathrm{Min}(G_A)$ such that $G=\varphi_C(G_A)$. The proof ends the same as it begins, however, $C=B\backslash A$ is considered. ◊

4 The algorithm

In this section we will present the recursive algorithm which generates all minimal interval extensions of a poset P in $O(n^2w\ I_{min}(P))$.

Firstly we give an overview of the algorithm to give a general understanding. The algorithm maintains an array I that contains the current minimal interval extension represented by a sequence of maximal antichains, and the graph G_A corresponding to the last antichain in I.

The main procedure used by the algorithm, called Gen_Max_Chain, is recursive and follows a maximal chain in *AM*(P) (maintained by the array I). When the last element of I is the set of maximal elements of P, then I is a minimal interval extension of P. The depth of recursive calls used to obtain this minimal interval extension is equal to the number of maximal antichains in I. Every level of the recursion has an associated graph G_A corresponding to the last antichain A of I. Moreover, at every level we compute $\varphi(G_A)$ to apply Gen_Max_Chain to each element of **Covers**(A).

Algorithm 1: Generation of all minimal interval extensions $I_{min}(P)$;

Inputs: The transitive reduction of an order P.

```
Begin
    Preprocessing; {the preprocessing computes A=Min(P) and the graph $G_A$}
    Gen_Max_Chain(A);
End.
```

```
Procedure Gen_Max_Chain (A: Antichain);
  Begin
     If A=Max(P) Then  Process(I)
     Else
     While Min_$G_A$ $\neq\emptyset$ Do
          Begin
           Choose x from Min_$G_A$;
           Class:={x};
           Compute $\varphi_x(G_A)$;
           B:=(A\Impred(x)) $\cup$ Class;
           I:=I $\cup$ {B};
           Gen_Max_Chain(B);
           Retrieve $G_A$ and I;
         End;
End; {Gen_Max_Chain}
```

Note : The minimal interval extension is output as a set of maximal antichains; one can transform this in the representation canonique in linear time.
The following result comes immediately from Theorem 3 and Theorem 4.

Theorem 5: *Algorithm 1 computes all the minimal interval extensions of a given order P.*

Let us now describe the details of our implementation in order to evaluate the complexity.

1) Data structures :

For each vertex x in G_A, we have the list Impred(x) of immediate predecessors. This list is the input of the Algorithm.
G_A is implemented in the following way:
- To each edge (x,y) of G_A is associated the integer Label_Count(x,y) representing the number of elements of labels in the edge (x,y).
- To each vertex x of G_A is associated the list Label_List(x) containing the edges having x in their label, and a counter Count(x) that maintains the number of edges (y,x) incident to x with Label_Count equal 0 and such that y and x are not in the same class. We also associate to x the list Comp(x) that contains the vertices y such that Label_Count(x,y)=0.
- A double linked list Min_G_A is used to maintain the set Min(G_A).

2) Informal description of the operations used :

The preprocessing consists in the computation of G_A, where A represents the minimal elements of P, and initializes I with A.
We compute the transformation $\varphi_X(G_A)$ as follows:
For each element y in (Impred(x)$\cap$A), we delete y from the label of edges it belongs to, by simply decreasing the corresponding Label_Count by 1. If any Label_Count(z,t) becomes 0 then we add t to the end of the list Comp(z), and we have two cases:
a) Label_Count(t,z)=0 then z and t will be in the same class (having the same predecessors in the new minimal interval extension). Here we decrease Count(z) by 1 and we do not increase Count(t). If Count(z) becomes 0, then we add z to Min{G_A}.
b)Label_Count(t,z)$\neq$0 then we have a new comparability. If t was in Min_G_A we delete it. We increase Count(t) by 1.

Elements y of Comp(x) such that x $\in$Comp(y) (ie. Label_Count(y,x)=0) are in the same class as x; although in our implementation they are not gathered, they must be treated in the same way as x. So for each element z in Comp(y) and Label_Count(z,y)$\neq$0 we decrease Count(z). We also delete y from Min_G_A.

The operations retrieve G_A and I of the procedure Gen_Max_Chain consists only in restoring the modified values of Count, Label_Count, Comp, Min_G_A and I. This can be done with the same complexity as below with our data structures.

Remark : The algorithm does not compute the equivalent classes.

3) Detailed algorithm :

We present a recursive Pseudo-Pascal implementation of Algorithm 1. The input consists of the transitive reduction of an order P, where n is the number of elements in P.

Variables:

```
    Label_Count : array[1..n,1..n] of integer;
    Count : array[1..n] of integer;
    Comp : array[1..n] of list of integer; { Each list works as a stack}
    Label_List : array[1..n] of list of edges;
    I : a list of antichains
```

Preprocessing:

```
Min_G_A =∅;
A:=∅;
For i:=1 to n Do
   Begin
      Label_List[i]:=nil;
      Count[i]:=0;
      Comp[i]:=nil;
   End;
For i:=1 to n-1 Do
      If Impred[i]=∅ Then A:=A+{i};
      For j:=i+1 to n Do
           Begin
            If i ∈ Impred(j) Then
               Begin
                 Label_Count[i,j]:=0;
                 Label_Count[j,i]:=∞;{This value tell us that we will never
                                      have i and j in the same class}
                 Count[j]:=Count[j]+1;
                 Add {j} to Comp[i];
               End
            Else
               If j ∈ Impred(i) Then
                  Begin
                   Label_Count[j,i]:=0;
                   Label_Count[i,j]:=∞;
                   Count[i]:=Count[i]+1;
                   Add {i} to Comp[j];
                  End
               Else
                If Impred[i]∩Impred[j]≠∅ Then
                  Begin
                    X:=Impred[i]-Impred[j];
                    Y:=Impred[j]-Impred[i];
                    Label_Count[i,j] := |X|
                    Label_Count[j,i] := |Y|
                    { If |Y|=0 and |X|=0 then i and j are in the same class}
                    If |X|=0 Then
                        Begin
                            If |Y|≠0 Then Count[j]:=Count[j]+1;
                            Add {j} to Comp[i];
                        End
                    Else For z ∈ X Do Add the edge (i,j) to Label_List[z];
                    If |Y|=0 Then
                        Begin
                            If |X|≠0 Then Count[i]:=Count[i]+1;
                            Add {i} to Comp[j];
                        End
                    Else For z ∈ Y do Add the edge (j,i) to Label_List[z];
                  End;
        End;
    End;
I:={A};
End of Preprocessing.
```

Main Program:

```
Preprocessing;
Min_$G_A$ :=∅;
Reached:=∅;
For j ∈ A Do
    For i ∈ Comp[j] Do
        Begin
          Count[i]:=Count[i]-1;
          If Count[i]=0 Then  add {i} to Min_$G_A$;
        End;
Reached:={A};
Gen_Max_Chain(A);
End of Main Program.
```

The recursive procedure

```
Procedure Gen_Max_Chain (A: Antichain);
  Begin
     If A=Max(P) Then  Process(I)
     Else
      While Min_$G_A$ ≠∅ Do
          Begin
           Choose x from Min_$G_A$;
           Class:={x};
           Compute $\varphi_x(G_A)$;
           B:=(A\Impred(x)) ∪ Class;
           I:=I ∪ {B};
            Reached:=Reached+{Class};
           Gen_Max_Chain(B);
           Retrieve $G_A$ and I;
          End;
 End; {Gen_Max_Chain}
```

Computing $\varphi_x(G_A)$

```
Delete x from Min_$G_A$;
S:=(Impred(x) $\cap$ A)
For z $\in$ S Do
      {Delete z from all edge's labels}
      For (e=(a,b) $\in$ Label_Liste[z]) and (b not reached) Do
        Begin
        Label_Count[a,b]:=Label_Count[a,b]-1;
        If Label_Count(a,b)=0   Then
          Begin
            If Label_Count(b,a)=0   Then
              Begin {a and b are now in the same class}
                  Count[a]:=Count[a]-1;
                  If Count[a]=0 Then   Add a to Min_$G_A$;
                End
            Else
              Begin   { new comparability is created}
                  If Count[b]=0 Then Delete b from Min_$G_A$;
                  Count[b]:=Count[b]+1;
                End;
            Add b to Comp[a];
          End;
         End;

{Computing the class and the new minimal elements }
For  y $\in$ Comp[x] Do
     Begin
       If Label_Count[y,x]=0 Then
         Begin
           Delete y from Min_$G_A$;
           Class:=Class+{y};
         For  (z $\in$ Comp[y]) and (Label_Count[z,y]$\neq$0) Do
           Begin
             Count[z]:=Count[z]-1;
             If Count[z]=0 Then Add z to Min_$G_A$;
           End
         End
       Else
         Begin
             Count[y]:=Count[y]-1;
             If Count[y]=0 Then Add y to Min_$G_A$;
         End;
End;
```

End of computing $\varphi_x(G_A)$

Retrieving G_A and I:

```
Reached:=Reached\{Class};
For  y $\in$ Comp[x] Do
     Begin
```

```
        If Label_Count[y,x]=0 Then
          Begin
           Add y to Min_$G_A$;
           For  (z ∈ Comp[y]) and (Label_Count[z,y]≠0) Do
             Begin
                Count[z]:=Count[z]+1;
                If Count[z]=1 Then Delete z from Min_$G_A$;
             End
          End
        Else
          Begin
                Count[y]:=Count[y]+1;
                If Count[y]=1 Then Delete y from Min_$G_A$;
          End;
End;

S:=the reverse of S;
For x ∈ S Do
        {Delete x from all edge's labels}
        For (e=(a,b) ∈ Label_Liste[x]) and (b not reached)  Do
          Begin
           Label_Count[a,b]:=Label_Count[a,b]+1;
           If Label_Count(a,b)=1   Then
             Begin
              If Label_Count(b,a)=0   Then
                 Begin{a and b were in the same class}
                    Count[a]:=Count[a]+1;
                    If Count[a]=1 Then Delete a from Min_$G_A$;
                   End
              Else
                 Begin {new comparability was created}
                     If Count[b]=1 Then Add b to Min_$G_A$;
                     Count[b]:=Count[b]-1;
                  End;
              Delete b from Comp[a];
            End;
           End;
Delete the last element of I.
Add x to Min_$G_A$;
```

End of Retrieving G_A and I:

Theorem 5: *Algorithm 1 compute $I_{min}(P)$ in $O(n^2w|I_{min}(P)|)$.*
Proof:

The preprocessing takes no more than $O(n^3)$.
For the construction of minimal interval extension, each vertex is deleted at the most one time from the label of edges it belongs to. Since a vertex can belong to at most nw labels of edges, the cost of these deletion operations is in $O(n^2w)$ for the construction of one minimal interval extension. The cost of the reverse operations is the same. Since these operations are the most costly of the algorithm, the result follows. ◊

The same algorithm can be easily modified to obtain the lattice *AM*(P). We just have to do the recursive call only if the antichain B has not been obtained before. This verification can be done in $O(w^2 \text{LogN})$ where N is the number of elements of *AM*(P), by using a binary search tree (see Aho et al[1]), and computing one maximal antichain costs O(nw). So we have the following corollary.

Corollary 1: *It is possible to construct the lattice AM(P) in $O(Nw^2 Log\ N)$, where N is the number of maximal antichains in P.*

5 Conclusion

We conclude by remarking that an algorithm that generates *AM*(P) can be used to generate the Galois lattice of binary relation. Indeed any binary relation R can be seen as a bipartite order B(R) and it is shown in Behrendt[2] that *Gal*(R) =*AM*(B(R')) where R' is the complementary of R. So since w(B(R')) is in the same range as the number of elements in R, our algorithm computes *Gal*(R) in $O(n^3\ |Gal(R)|)$. This result has been shown first by Bordat[4,5] in a different way; it can be shown that Bordat's algorithm can be used to compute *AM*(P) for an order P in $O(n^3|AM(P)|)$.

A natural question is still open: is it possible to generate all minimal interval extensions of an order P in $O(|I_{min}(P)|)$, as it is possible to generate all linear extensions of P in $O(|\mathfrak{L}(P)|\ )$ amortized time.
The best known algorithm generating all linear extensions of an order are based on the exchange of elements in one linear extension to obtain the next. This method is very efficient (see Pruesse et al[12,13]), but it could be difficult to transpose this result in the case of minimal interval extensions since it seems to strongly use the distributivity of the antichains lattice of P, and that the maximal antichains lattice of P can be not distributive. This difference comes from the fact that, when selecting a minimal vertex in PRED(P) to construct a linear extension, new comparabilities may appear. So, to list only good linear extensions, we must take into acount these new comparabilities.

Acknowledgments: We would to thank Genevieve Simonet for helpful remarks.

References:

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, "Data Structures and Algorithms", Addison-Welsey , 1983.

[2] G. Behrendt,"Maximal antichains in partially ordered sets", Ars Combin., 25 C, 1988, P. 149-157.

[3] R. Bonnet et M. Pouzet,,"Extensions et stratifications d'ensembles dispersés", C. R. Acad. Sci. Paris,268,serie A, 1969, P. 1512-1515.

[4] J. P. Bordat,"Calcul pratique du treillis de Galois d'une correspondance", Math. Sci. Hum. 96 (1986) P. 31-47.

[5] J. P. Bordat, "Sur l'Algorithmique Combinatoire d'ordres fini", These de Doctorat d'état (1992) USTL Montpellier .

[6] C. Diehl and C. Jard, "Interval approximations and message causality in distributed systems", Technical report, IRISA, Rennes, France, Sept 1991.

[7] M. Habib, M. Morvan, M. Pouzet. and J.X. Rampon, "Extensions intervallaires minimales", C. R. Acad. Sci. Paris, t. 313, Série I, P. 893-898, 1991.

[8] M. Habib, M. Morvan and J.X. Rampon, "Remarks on some concurrency measures", Lectures notes in Computer Science n° 484 (R.H. Möhring Ed.), Springer Verlag, 1991, pp. 221-238.

[9] M. Habib, M. Morvan and J.X. Rampon, "About minimal interval extensions", R.R. N°84 juin 1990. CRIM, Submitted to Order.

[10] A.D. Kalvin and Y.L. Varol,"On the generation of all topological sortings",J. Algorithms, 4, pp 150-162 1983.

[11] D.E. Knuth and J.L. Szwarcfiter, "A structured program to generate all topological sorting arrangements", Information Processing Letters, 2 (1974), pp 153-157.

[12] G. Pruesse and F. Ruskey, "Generating Linear Extensions Fast", Dep. of Comp. Sc Univ of Victoria Canada.

[13] G. Pruesse and F. Ruskey, "Generating the Linear Extensions of certain posets by transpositions", SIAM J.Discrete Math., August 1991.