An On-Line Algorithm for the Mac-Neille Completion of a Poset

(Extended Abstract)

Luca Forlizzi[1] **Enrico Nardelli**[1,2]

1. Dipartimento di Matematica Pura ed Applicata, Univ. of L'Aquila, Via Vetoio, Coppito, I-67010 L'Aquila, Italia. E-mail: {forlizzi,nardelli}@univaq.it
2. Istituto di Analisi dei Sistemi ed Informatica, Consiglio Nazionale delle Ricerche, Viale Manzoni 30, I-00185 Roma, Italia.

Abstract

In this paper we introduce an efficient algorithm for the on-line computation of the MacNeille completion of a poset. Our algorithm is polynomial in the number of elements in the lattice completion and has a better worst-case complexity than previous proposals.

1 Introduction

The area of algorithms for partially ordered sets (i.e., posets) is a relatively new one and is subject of many research efforts [18, 19, 20, 22], since it has many potential applications in a lot of areas of computer science.

In particular, the computation of the MacNeille completion (or lattice completion) of a given poset is an interesting subject to investigate [2, 10, 24], due to the fact that many algorithms are more efficient when the poset is a lattice (e.g., testing dominance in a poset is easier in a lattice [25] than in a general poset [26]) and every poset can be embedded in a lattice: the MacNeille completion is then the smallest lattice allowing such an embedding. Lattices are also largely investigated in computer science as efficient representation models [1, 5, 7, 8, 11, 13, 16, 21, 27].

Also, a wide interest has recently spread about *on-line* algorithms, both in the general algorithmic community [17] and among those more specifically interested to algorithms for posets [4, 12, 15, 14]. For our purposes, on-line is intended in the sense that only insertions of elements are allowed (i.e., no deletions).

In this paper we present an efficient algorithm to compute on-line the lattice completion of a poset. Computing on-line the lattice completion means to start with a small subposet whose lattice completion is computed trivially, and then inserting one after the other the remaining elements of the poset, building, after each insertion, the lattice completion of the inserted elements.

If one just uses the MacNeille definitions of lattice completion to derive an algorithm, a time exponential in the size of the input poset may be required. Moreover, the algorithm is not on-line.

In the literature, to the best of our knowledge, the first on-line algorithm for the computation of the lattice completion that is polynomial in the number of the elements of the resulting lattice was [21]. Perry has not given in [21] a detailed computational complexity analysis, and the presentation of the algorithm is at an abstract level.

Using standard data structures we proved [9] that her algorithm requires, for each element that is added to poset P, $O(s^3)$ time to compute the (transitive closure of the) lattice completion L of P, where $n = |P|$ and $s = |L|$, starting from the lattice completion obtained after the addition of the previous element. Note that it may be $s = \Theta(2^n)$.

Caseau presented in [5] an on-line algorithm to build the new lattice completion when a new minimal element is inserted. His algorithm is not fully detailed and it has a worst case complexity of $O(n^2 s^4)$.

Habib introduced in [12] the so-called Lazy MacNeille completion for representing a poset so to allow to test if the least upper bound of two elements x and y exists (in $O(ns)$) and, if not, to add it to the representation. Hence to compute the whole MacNeille completion L of a poset P when a new element is inserted one has to repeat, in general, the above test for each pair of elements inserted in L. This gives a total complexity of $O(ns^3)$.

Our algorithm computes, for each added element, the transitive closure of the lattice completion L of poset P in time $O(sn^2 + m)$ where m is the number of edges in L. Hence our algorithm is always better than Perry's one but for the case of $s = O(n)$, when it has the same complexity. As a side effect, also the transitive reduction of L is computed.

Note that using the currently most efficient data structure for the representation of (transitive reduction of) lattices [24, 25] would not be, in general, of help in this case, since such a data structure is a static one with an $O(s^2)$ building time.

1.1 Notations

A *partially ordered set* (*poset*) is an algebraic structure $< P, \leq_P >$ where P is a set and $\leq_P$ is a partial order relation (namely a reflexive, symmetric and transitive relation) among the elements of P. To simplify notations, we usually denote a poset $< P, \leq_P >$ as P. In this paper we always assume that P is finite. Two elements $x, y \in P$ are said *comparable* if $x \leq_P y$ or $y \leq_P x$. Otherwise they are said *incomparable*. When $x \leq_P y$ we say y *dominates* x or x *is dominated by* y (in P). If $x \leq_P y$ and $x \neq y$ we write $x <_P y$. We say that y *covers* x or x *is covered by* y and write $x \prec_P y$ if $x <_P y$ and there is no $z \in P$ such that $x <_P z <_P y$. A subset X of P is called an *antichain* (resp. a *chain*) if it contains only pairwise incomparable (resp. comparable) elements. The *height* (resp. *width*) of P, denoted with $h(P)$ (resp. $\omega(P)$) is the size of a maximal chain (resp. antichain) of P.

Given a subset X of P we say that $z \in P$ is an *upper bound* (resp. *lower bound*) of X if $\forall x \in X$, it is $x \leq_P z$ (resp. $z \leq_P x$). Given $X \subseteq P$ we denote as X^{*_P} (resp. X_{*_P}) the set of all upper bounds (resp. lower bounds) of X. Given $x \in P$ we simplify the notation writing x^{*_P} instead of $\{x\}^{*_P}$ and x_{*_P} instead of $\{x\}_{*_P}$. The element z is called the *least upper bound* or *join* of X and is denoted as $lub_P(X)$, if $z \in X^{*_P}$ and $z \leq_P t$ for all $t \in X^{*_P}$. The *greatest lower bound* or *meet* of X (denoted as $glb_P(X)$) is defined dually. A non-empty poset P is called a *lattice* if for each $x, y \in P$ both $lub_P(\{x, y\})$ and $glb_P(\{x, y\})$ exist. When P is a lattice then for each $X \subseteq P$ both $lub_P(X)$ and $glb_P(X)$ exist. Given $X \subseteq P$ and $x \in X$ then x is said to be a *maximal element* (resp. *minimal*) of X if there is no $z \in X$ such that $x <_P z$ (resp. $z <_P x$). We denote the set of all maximal (resp. minimal) elements of X as $maximal_P(X)$ (resp. $minimal_P(X)$). Given $X \subseteq P$ and $t \in X$ we say that t is the *top* (resp. *bottom*) of X if t is the unique maximal (resp. minimal) element of X, denoted by $Top(X)$ (resp. $Bottom(X)$). If L is a lattice both $Top(L)$ and $Bottom(L)$ exist. Given a poset P, we define its *MacNeille completion* (or *lattice completion*) as the poset $< M(P), \subseteq >$ where $M(P) = \{Q \mid Q \subseteq P \wedge Q = (Q^{*_P})_{*_P}\}$. It is well known [3, 6] that $M(P)$ is a lattice and has the additional property of being the smallest lattice into which P can be embedded.

When we speak of the representation of a poset we usually mean a representation of the whole order relation. When we want to refer to the transitive reduction we will specify it explicitly.

2 The Perry's algorithm for MacNeille completion

2.1 An informal description

The algorithm of Perry builds the MacNeille completion of a given poset P by starting from an initial lattice L_0 containing just the top and the bottom of P^1.

The i-th on-line pass takes as input lattice L_{i-1} which is the MacNeille completion of a subposet P_{i-1} of P such that $|P_i| = i + 2$, together with a new element $q \in P$ such that $q \notin P_{i-1}$. The purpose of the i-th on-line pass is to compute the MacNeille completion L_i of subposet $P_i = P_{i-1} \cup \{q\}$. This will require the insertion in L_{i-1} of new order relations and possibly new elements to obtain lattice L_i. When all the elements of P have been processed the algorithm terminates.

In a first step, all order relations of the new element q with L_{i-1} are determined and inserted into the representation of L_{i-1}. For each element l of L_{i-1}, the set of elements of P_i dominated by both q and l is determined and then the set of the lower bounds of the upper bounds of the above set is compared against the set of elements of P_i dominated by q. If the former set is contained in the latter, then l is dominated by q, while if the former contains the latter, then l dominates q. If the two sets coincide then $l = q$. If neither of the two set is contained in the other one l and q are not comparable.

In a second step, new elements and order relations are possibly inserted in $L_{i-1} \cup \{q\}$ to enforce the lattice property and to obtain L_i. In fact it may happen that for some element l in L_{i-1}, the pair (l, q) lacks a join (or meet) in $L_{i-1} \cup \{q\}$: then a new element must be added to be the required join (or meet).

The search for elements l in L_{i-1} such that the pair (l, q) lacks a join can be restricted to a subset $R_U(q)$. Then for each element l in $R_U(q)$ a suitable function $\Phi(.)$ is applied to determine the join $\Phi(l)$ in L_{i-1} of the elements that dominate both l and q.

If such a join is l itself and l does not dominate q then l is not the join of elements in $L_{i-1} \cup \{q\}$, hence a new element needs to be added. Moreover, order relations are possibly added among the new element and elements in L_{i-1} or elements previously added during this step.

Otherwise two things may happen:

1. l dominates q, then l is the required join; or
2. $\Phi(l) \neq l$, then we can defer the enforcement of the lattice property of $L_{i-1} \cup \{q\}$ to the visit of $\Phi(l)$. In fact in this case the join in $L_{i-1} \cup \{q\}$ of pair $(\Phi(l), q)$ is also the join of pair (l, q).

The search for elements l in L_{i-1} such that the pair (l, q) lacks a unique meet is done in a dual way.

2.2 A formal description

In this section we give a more formal description of the generic on-line pass of the Perry's algorithm. For more details see [21]. In this and the following subsection, input (resp. output) lattice L (resp. L') to the algorithm corresponds to the lattice indicated

^{1}If P does not contain a top or a bottom, these can be added as L_{i-1} (resp L_i) in the informal description. Similarly, P corresponds to P_{i-1} and P' corresponds to P_i.

If x and y are elements of poset T we use the following notations: $x :=_T y$ meaning that x is made coincident with y in T; $x :\leq_T y$ meaning that the pair (x, y) is added to the current partial order relation of T with the semantics $x \leq_T y$. Comments are enclosed between '/*' and '*/'.

If an element is inserted while visiting element $l \in L$ (lines 19 and 28), then we let l refer to this new element as $New(l)$.

INPUT: P'**:Poset;** q**:new element;** L**:lattice;**
OUTPUT: L'**:lattice;**

```
1   begin
2     /* Step 1: Form a temporary poset $T$ relating $q$ with elements of $L$. */
3     for $l \in L$ do
4         if $l \in P$
5             then relate $l$ and $q$ in $T$ as they are related in $P'$
6              else if $((l_{*_L} \cap P)^{*_{P'}})_{*_{P'}} \subseteq q_{*_{P'}}$ then $l :\leq_T q$ fi
7                   if $q_{*_{P'}} \subseteq ((l_{*_L} \cap P)^{*_{P'}})_{*_{P'}}$ then $q :\leq_T l$ fi
8                   if $q_{*_{P'}} = ((l_{*_L} \cap P)^{*_{P'}})_{*_{P'}}$ then $q :=_T l$ fi fi od
9     /* Step 2: Form lattice $L'$ adding new elements and order relations to $T$. */
10    Set $L' := T$
11    /* Let $cover_T(q) = \{x_1, \dots, x_k\}$ and $cocover_T(q) = \{y_1, \dots, y_m\}$. */
12    /* We use the following notation:
13    $\Phi(l) = glb_L(\{lub_L((x_1, l)), \dots, lub_L((x_k, l))\})$
14    $\Psi(l) = lub_L(\{glb_L((y_1, l)), \dots, glb_L((y_m, l))\})$ */
15    /* We use the following notation:
16    $R_U = (glb_L(cover_T(q)))^{*_L} \setminus (\bigcup_i \{x_i\}^{*_L})$
17    $R_D = (lub_L(cocover_T(q)))_{*_L} \setminus (\bigcup_i \{y_i\}_{*_L})$ */
18    for $l \in R_U$ such that $l$ is incomparable with $q$ in $T$ and $\Phi(l) = l$ do
19        insert in $L'$ a new element $New(l)$
20        $l :\leq_{L'} New(l)$
21        $q :\leq_{L'} New(l)$
22        for $l_1 \in L$ do
23            if $(l_1 \leq_L l) \wedge (New(l_1)$ exists$)$ then $New(l_1) :\leq_{L'} New(l)$ fi
24            if $(l \leq_L l_1) \wedge (New(l_1)$ exists$)$ then $New(l) :\leq_{L'} New(l_1)$ fi
25            if $l_1 \in l_{*_L} \cup \{q\}$ then $l_1 :\leq_{L'} New(l)$ fi
26            if $l_1 \in \{l, q\}^{*_T}$ then $New(l) :\leq_{L'} l_1$ fi od od
27    for $l \in R_D$ such that $l$ is incomparable with $q$ in $T$ and $\Psi(l) = l$ do
28        insert in $L'$ a new element $New(l)$
29        $New(l) :\leq_{L'} l$
30        $New(l) :\leq_{L'} q$
31        for $l_1 \in L$ do
32            if $(l_1 \leq_L l) \wedge (New(l_1)$ exists$)$ then $New(l_1) :\leq_{L'} New(l)$ fi
33            if $(l \leq_L l_1) \wedge (New(l_1)$ exists$)$ then $New(l) :\leq_{L'} New(l_1)$ fi
34            if $l_1 \in l^{*_L} \cup \{q\}$ then $New(l) :\leq_{L'} l_1$ fi
35            if $l_1 \in \{l, q\}_{*_T}$ then $l_1 :\leq_{L'} New(l)$ fi od od
36  end
```

2.3 Complexity of Perry's algorithm

During Step 1 the cost for each $l \in L$ is dominated by the cost of finding sets $q_{*_{P'}}$ and $((l_{*_L} \cap P)^{*_{P'}})_{*_{P'}}$ and to check if one of them is contained in the other. To find $q_{*_{P'}}$ requires $O(n)$. To find $(l_{*_L} \cap P)$ also requires $O(n)$. Given a subset Q of P, finding the set of its upper bounds (or lower bounds) requires $O(n^2)$ because we have to compare each element of P with each element of Q. Then finding $((l_{*_L} \cap P)^{*_{P'}})_{*_{P'}}$ requires $O(n^2)$. Given $Q_1, Q_2 \subseteq P$, to check if $Q_1 \subseteq Q_2$ requires $O(n^2)$ (we have to search each element of Q_1 in Q_2). Hence the total cost of Step 1 is $O(sn^2)$.

During Step 2, finding $cover_T(q)$ and $cocover_T(q)$ requires $O(s^2)$. In fact since a transitive reduction of L is not available, one has to find maximal elements of a subset of L. Finding $glb_L(cover_T(q))$ and $lub_L(cocover_T(q))$ requires $O(s|cover_T(q)|)$ because finding the least upper bound of a pair of elements requires $O(s)$. It is $|cover_T(q)| = O(\omega(L))$ and we proved [9] that for some classes of posets it is $|cover_T(q)| = \Omega(\omega(P)^2)$. To determine R_U, R_D requires $O(s\omega(L))$ because we have to compare each element of L against each element of $cover_T(q)$ (or $cocover_T(q)$).

Then a loop through R_U begins. In the loop, function $\Phi(.)$ is evaluated. To evaluate $\Phi(.)$ takes $O(s\omega(L))$ because the evaluation consists in $|cover_T(q)| + 1$ operations of least upper bound or greatest lower bound of a pair of elements. Moreover in the loop an element is compared against each element of L. Hence the whole cost of the loop is $O(s^2\omega(L) + s^2)$. The dual loop through R_D has the same cost.

The total cost for the on-line pass of Perry's algorithm is therefore $O(s^3)$.

3 A Better Algorithm for the Transitive Closure

3.1 An informal description

The basic approach is the same as in Perry's algorithm. A first important difference is that we substitute the computation of $\Phi(.)$ with the (more efficient) computation of a different structure.

The computation of $\Phi(l)$, for an $l \in L_{i-1}$, in Perry's algorithm has the purpose of checking if a new element needs to be added to enforce the lattice property. This happens if $l = \Phi(l)$.

We substitute this computation with a different operation. Namely, we search for an element x in $cover_{L_{i-1}}(l)$ that is dominated by every element in L_{i-1} which dominates in $L_{i-1} \cup \{q\}$ both l and q. If x exists then $l <_{L_{i-1}} x$ and $x \leq_{L_{i-1}} \Phi(l)$, hence $l \neq \Phi(l)$.

To efficiently execute the above test, we build and maintain at each on-line pass also the transitive reduction of L_{i-1}, using suitable data structures to be presented in Sect. 3.5. Ths computation of $\Psi(.)$ is substituted with a similar operation.

An additional important difference from the point of view of the overall time complexity is in Step 2. We substitute the check against each element currently in the lattice (lines 22-26 and 31-35 of Perry's algorithms) with a check guided by the order relations currently existing in the lattice.

Finally, a minor difference with Perry's algorithm is in Step 1. We use a different method to check, given an element l of L_{i-1} whether l has to dominate new element q, or to be identified with it, or to be dominated by it (lines 6-8 of the Perry's algorithm). This also provides more efficiency, even if does not affect the overall time complexity.

3.2 A formal description

In this section we give a more formal description of the generic on-line pass of our algorithm. We first give the general schema and then details the internal procedures. We omit the operations dealing with the data structure maintaining the transitive reduction, that will be described in Sect. 3.5. Discussion about correctness is in Sect. 3.3.

INPUT: P'**:Poset;** q**:new element;** L**:lattice;**
OUTPUT: L'**:lattice;**

```
1   begin
2     /* Step 1: Form a temporary poset $T$ relating $q$ with elements of $L$ */.
3     for $l \in L$ do
4       if $l \in P$
5         then relate $l$ and $q$ in $T$ as they are related in $P'$
```

```
6                 else if ($l$ has to be dominated by $q$)$\wedge\neg$($l$ has to dominate $q$)
7                     then $l :\leq_{L'} q$ fi
8                 if $\neg$($l$ has to be dominated by $q$)$\wedge$($l$ has to dominate $q$)
9                     then $q :\leq_{L'} l$ fi
10                if ($l$ has to be dominated by $q$)$\wedge$($l$ has to dominate $q$)
11                    then $q :=_T l$
12                         $L' := T$
13                         stop fi fi od
14        /* Step 2: Form lattice $L'$ adding new elements to $T$. */
15        Set $L' := T$
16        for $l \in (glb_L(cover_P(q)))^{*L}$ and such that $l$ is incomparable with $q$ in $T$ do
17            if a new element needs to be inserted for $l$
18              then
19                  insert in $L'$ a new element $New(l)$
20                  $l :\leq_{L'} New(l)$
21                  $q :\leq_{L'} New(l)$
22                  for $l_1 \in l^{*L}$ do
23                      if $q \leq_T l_1$ then $New(l) :\leq_{L'} l_1$ fi
24                      if ($New(l_1)$ exists) then $New(l) :\leq_{L'} New(l_1)$ fi od
25                  for $l_1 \in l_{*L}$ do
26                      $l_1 :\leq_{L'} New(l)$
27                      if ($New(l_1)$ exists) then $New(l_1) :\leq_{L'} New(l)$ fi od fi
28            /* Insert here operations on the data structure
29            representing the transitive reduction. */
30        od
31        /* Update the data structure representing transitive reduction of $L'$
32        considering new elements inserted in $L'$ during the above for cycle. */
33        for $l \in (lub_L(cocover_P(q)))_{*L}$ and such that $l$ is incomparable with $q$ in $T$ do
34            if a new element needs to be inserted for $l$
35              then
36                  insert in $L'$ a new element $New(l)$
37                  $New(l) :\leq_{L'} l$
38                  $New(l) :\leq_{L'} q$
39                  for $l_1 \in l^{*L}$ do
40                      $New(l) :\leq_{L'} l_1$
41                      if ($New(l_1)$ exists) then $New(l) :\leq_{L'} New(l_1)$ fi od
42                  for $l_1 \in l_{*L}$ do
43                      if $l_1 \leq_T q$ then $l_1 :\leq_{L'} New(l)$ fi
44                      if ($New(l_1)$ exists) then $New(l_1) :\leq_{L'} New(l)$ fi od fi
45            /* Insert here operations on the data structure
46            representing the transitive reduction. */
47        od
48        /* Update the data structure representing transitive reduction of $L'$
49        considering new elements inserted in $L'$ during the above for cycle. */
50      end
```

To test whether l has to be dominated by q (lines 6, 8,10) we use a boolean function (*LessThan*) presented below. If *LessThan* returns true then l has to be dominated by q.

To test whether l has to dominate q (lines 6, 8,10) we use a dual boolean function (*GreaterThan*).

BOOLEAN FUNCTION LessThan
INPUT: l:element;
OUTPUT: result:boolean;

```
1       begin
```

```
    $result := true$
    for $x \in P$ do
        if $x \leq_L l \wedge x \not\leq_{P'} q$ then $result := false$ fi od
    return $result$
end
```

To test whether it is necessary to create a new element in the visit of $(lub_L(cocover_P(q)))_{*_L}$ we use a boolean function (*NewDown*) presented below. If *NewDown* returns true then a new element has to be inserted.

To test whether it is necessary to create a new element in the visit of $(glb_L(cover_P(q)))^{*_L}$ we use a dual boolean function (*NewUp*).

BOOLEAN FUNCTION NewDown
INPUT: l:element
OUTPUT: result:boolean

```
begin
    Compute $l_{*_L} \cap q_{*_{P'}}$
    $result := true$
    for $y \in cocover_L(l)$ do
        if $\forall x \in (l_{*_L} \cap q_{*_{P'}}),\ x \leq_L y$
          then $result := false$
                 store $y$ fi od
    return $result$
end
```

In the above function we need $cocover_L(l)$, which is not efficiently provided by the representation of L. Hence we use a supplementary data structure (see Sect. 3.5) representing the transitive reduction of L.

Note that to be able to answer test $x \leq_L y$ in $O(1)$ we represent partial order relations in L with a boolean adjacency matrix that is suitably enlarged during on-line passes. Note also we want to search in l^{*_L} and l_{*_L} without necessarily searching through L. This can be achieved by threading non zero entries of the matrix (which correspond to l^{*_L} and l_{*_L}).

3.3 Correctness

We here give correctness proofs for our algorithm. We assume correctness of Perry's algorithm (see [21]) and we show our algorithm produces the same results.

In Step 1 we substitute comparisons of lines 6-8 in Perry's algorithm with functions *LessThan* and *GreaterThan*. These functions implement an equivalent (but computationally less expensive) test, as shown by the following theorem.

Theorem 1 *It is:*

$$((l_{*_L} \cap P)^{*_{P'}})_{*_{P'}} \subseteq q_{*_{P'}} \iff \forall x \in (l_{*_L} \cap P), x \leq_{P'} q$$

and

$$q_{*_{P'}} \subseteq ((l_{*_L} \cap P)^{*_{P'}})_{*_{P'}} \iff \forall x \in (l^{*_L} \cap P), q \leq_{P'} x$$

□

Note also that if during Step 1 for an $l \in L$ we identify l and q then obviously $L = L'$, hence we can exit from the algorithm (line 13 of our algorithm).

In Step 2, Perry's algorithm performs a loop on each element of R_U and a similar one on each element of R_D. In our algorithm we have instead a loop on $(glb_L(cover_P(q)))^{*_L}$

and a similar one on $(lub_L(cocover_P(q)))_{*_L}$. Our loops perform the same operations as Perry's ones.

In fact $glb_L(cover_T(q)) = glb_L(cover_P(q))$ hence $R_U \subseteq (glb_L(cover_P(q)))^{*_L}$. Moreover $\forall x \in ((glb_L(cover_P(q)))^{*_L} \setminus R_U)$ we have $q \leq_T x$ hence our algorithm takes no actions when it examines elements of this kind because of the condition at line 16. The same happens with respect to R_D and $(lub_L(cocover_P(q)))_{*_L}$.

Then our algorithm performs the tests implemented by procedure $NewUp(l)$ and $NewDown(l)$ instead of evaluating functions $\Phi(l)$ and $\Psi(l)$. The following theorem proves that this is correct:

Theorem 2 *We have:*

$$\Phi(l) = l \iff \nexists y \in cover_L(l) \mid \forall x \in (l^{*_L} \cap q^{*_{P'}}), y \leq_L x$$

and

$$\Psi(l) = l \iff \nexists y \in cocover_L(l) \mid \forall x \in (l_{*_L} \cap q_{*_{P'}}), x \leq_L y$$

□

Finally note that we have substituted instructions in lines 22-26 (resp., lines 31-35) of Perry's algorithm, executing a loop through whole L, with instructions in lines 22-27 (resp., lines 39-44), in our algorithm, executing a loop through l^{*_L} (resp., through l_{*_L}).

The correctness of these substitutions above can be easily seen checking that conditions in lines 22-26 of Perry's algorithm imply that, for each l_1 that is incomparable with l, neither l_1 nor the possibly existing $New(l_1)$ have to be related with $New(l)$. Dually for conditions in lines 31-35.

3.4 Complexity of Our Algorithm

In order to analyze computational complexity of our algorithm we need the following results:

Theorem 3 *Let L be the MacNeille completion of P. Then $\forall l \in L$, we have $|cover_L(l)| = O(\omega(P))$ and $|cocover_L(l)| = O(\omega(P))$.* □

Corollary 4 *Let L be the MacNeille completion of P. Then the number of order relations in the transitive reduction representation of L is $O(s\omega(P))$.* □

Note that Theorem 3 can not be used to show that the computational complexity of the evaluation of functions $\Phi(l)$ and $\Psi(l)$ is $O(s\omega(P))$. In fact it does not apply to $|cover_T(q)|$ because the intermediate working poset T is not the normal completion of P. It can be shown [9] that for some classes of posets $|cover_T(q)| = \Omega(\omega(P)^2)$, but it is not known if posets exist such that $|cover_T(q)| = \Omega(\omega(L))$.

We are now ready to discuss time complexity.

In Step 1, for each $l \in L$ the cost of the step is dominated by the cost of functions *LessThan* and *GreaterThan* which is $O(n)$. Hence the total cost of Step 1 is $O(sn)$.

In Step 2, finding $glb_L(cover_P(q))$ and $lub_L(cocover_P(q))$ requires $O(s|cover_P(q)|)$ because finding the least upper bound of a pair of elements requires $O(s)$, hence the cost is $O(s\omega(P))$. In the subsequent loops (each iterated $O(s)$ times) function *NewUp* (or *NewDown*) is evaluated. The cost of *NewDown* is $O(n\omega(P))$, because $|l_{*_L} \cap q_{*_{P'}}| = O(n)$ and $|cocover_L(l)| = O(\omega(P))$. The cost of *NewUp* is the same. Moreover in each loop any new element $New(l)$ is compared against each element of l^{*_L} and l_{*_L}. Note that $\sum_{l \in L} |l^{*_L}| + |l_{*_L}| = O(m)$, where m is the number of order relations of the transitive closure. Obviously in the worst case $m = O(s^2)$. Hence the whole cost of Step 2 is $O(sn\omega(P) + m)$.

Finally note that each update to the structure described at the end of Sect. 3.2 allowing to test $x \leq_L y$ in constant time can be executed in constant time during Step 1 and Step 2.

The total cost for an on-line pass is $O(sn^2 + m)$.

3.5 Maintaining the transitive reduction

We now describe a data structure that for each element $l \in L$, where L is the input lattice of our algorithm, stores $cover_L(l)$ and $cocover_L(l)$. Suppose we have such a structure for L before executing a generic pass of our algorithm. Then during Step 2 of our algorithm we need to update the data structure according to the new lattice L'. We here below discuss explicitly only how to perform changes due to those new elements inserted by our algorithm during the visit of $(lub_L(cocover_P(q)))_{*_L}$ in lines 33-47. Changes caused by the visit in lines 16-30 can performed in a dual way.

3.5.1 Informal Description

We use a queue M, cleared at the beginning of the generic pass, where we store elements of L whose visit during the current pass has determined the insertion of a new element. To implement function $New(\cdot)$ we use also an array A that for each element of L stores a boolean flag and an element of L'.

Moreover we have to do some extra work (to be inserted at lines 45-46) of our algorithm, after each call to function *NewDown*.

First of all, we set $A(l).flag := NewDown(l)$. If $NewDown(l)$ returns true a new element $New(l)$ has been created. Then we enqueue l in M, and store $New(l)$ in $A(l).element$. Note that M maintains elements in their topological order.

Otherwise there exists $y \in cocover_L(l)$ such that $\forall x \in (l_{*_L} \cap q_{*_{P'}}),\ x \leq_L y$. Such y has been found and stored by function $NewDown(l)$. Then we store y in $A(l).element$.

All this extra work can be done in $O(1)$ hence the complexity of our algorithm is not affected.

We now describe the procedure to be inserted at lines 48-49 to update the representation of the transitive reduction of L'. This procedure visits elements of M and for each visited element l first computes $cocover_{L'}(A(l).element)$ and then updates $cocover_{L'}(y)$ and $cover_{L'}(y)$ of other elements $y \in L'$, by deleting transitive relations, to provide a correct representation of the transitive reduction of L'. Note that $cover_{L'}(A(l).element)$ has been computed as a consequence of the updates performed during the visit of elements x preceding l in M.

3.5.2 A formal description and correctness

The procedure updating the transitive reduction of L' is the following:

PROCEDURE BuildAll
INPUT: M**:list of elements;**

```
1      begin
2        $\bar{l} := lub_L(cocover_P(q))$
3        if $M = \emptyset$
4          then $cocover_{L'}(q) := \{\bar{l}\}$
5                 $cover_{L'}(\bar{l}) := cover_{L'}(\bar{l}) \cup \{q\}$
6                 stop
7            else /* Note that $\bar{l}$ is the first element in $M$. */
8                 $cocover_{L'}(q) := \{A(\bar{l}).element\}$
9                 $cover_{L'}(A(\bar{l}).element) := \{q\}$
10                repeat dequeue $l$ from $M$
11                  $cover_{L'}(A(l).element) := cover_{L'}(A(l).element) \cup \{l\}$
12                  $cocover_{L'}(l) := cocover_{L'}(l) \cup \{A(l).element\}$
13                  compute $cocover_{L'}(A(l).element)$
14                  for $y \in cocover_{L'}(A(l).element)$ do
15                    for $z \in cover_{L'}(y)$ do
16                      if $A(l).element <_{L'} z$
17                        then $cover_{L'}(y) := cover_{L'}(y) \setminus \{z\}$
```

```
                                    $cocover_{L'}(z) := cocover_{L'}(z) \setminus \{y\}$ fi od
                            $cover_{L'}(y) := cover_{L'}(y) \cup \{A(l).element\}$ od
                    until $M = \emptyset$
        fi
    end
```

Note that $cocover(\cdot)$ and $cover(\cdot)$ are implemented as lists where deletions can be done in constant time, since the operation is executed while visiting the element to be deleted.

We now describe the procedure that given an element l computes $cocover_{L'}(A(l).element)$. This set of elements is built by generating for each element of $cocover_L(l)$ a candidate and then checking if such a candidate needs to be inserted into $cocover_{L'}(A(l).element)$ or not.

We first define a function (*FindCandidate*) that provides for an input element y a candidate for the insertion in $cocover_{L'}(A(l).element)$. Let x be the least upper bound of the sets of elements of L dominated by both y and q. If x is dominated by both y and q (remember that it may be $x = y$) then *FindCandidate* returns x. Otherwise it returns $A(x).element$. The formal description of such a function is:

FUNCTION FindCandidate
INPUT: y**:element**
OUTPUT: element

```
    begin
      if $y \leq_{L'} q$
        then return $y$
         else if $A(y).flag$
                then return $A(y).element$
                 else /* Remember that when $A(y).flag$ =false then
                      $A(y).element$ is an upper bound of $(y_{*_L} \cap q_{*_{P'}})$. */
                      $FindCandidate(A(y).element)$ fi fi
    end
```

We are now ready to give a procedure (*Build*) checking if candidates have to be included in $cocover_{L'}(A(l).element)$ and inserting them in the positive case. If a candidate dominates a previously found candidate z then z has not to be included. If a candidate y is dominated by a previously found candidate then y has not to be included. The formal description of the procedure is:

PROCEDURE Build
INPUT: l**:element;**

```
    begin
      $S := \emptyset$
      for $y_1 \in cocover_L(l)$ do
          $y_2 := FindCandidate(y_1)$
          for $z \in S$ do if $z <_{L'} y_2$ then $S := S \setminus \{z\}$ fi od
          $insert := true$
          for $z \in S$ do
              if $y_2 <_{L'} z$ then $insert := false$ fi od
          if $insert$ then $S := S \cup \{y_2\}$ fi od
      $cocover_{L'}(A(l).element) := S$
    end
```

Correctness of procedure *Build* is given by following theorems.

Theorem 5 *Given $l \in L$, procedure Build(l) returns $cocover_{L'}(New(l))$.* □

The proof of previous theorem is based on the following Lemma.

Lemma 6 *Let $l \in L$ and $\hat{l} = New(l)$. For each $x \in cocover_{L'}(\hat{l})$, it exists $x_1 \in cocover_L(l)$ such that FindCandidate(x_1) returns x.* □

Finally we prove correctness of the main procedure.

Theorem 7 *Procedure BuildAll computes $cocover_{L'}(l)$ and $cover_{L'}(l)$ for any $l \in (lub_L(cocover_P(q)))_{*L}$.* □

3.5.3 Complexity

Function *FindCandidate* requires $O(n)$. In fact each operation but for the recursive call requires $O(1)$. In the function body a recursive call is invoked with an argument strictly less than the input argument to the function. Hence arguments of successive recursive calls form an $O(n)$ chain in L as shown by the following theorem:

Theorem 8 *Let L be the MacNeille completion of P. Then $h(L) = O(n)$.* □

Procedure *Build* requires $O(n\omega(P))$ because it performs $|cocover_L(l)|$ calls to *FindCandidate* and compares each of the returned element with at most $|cocover_L(l)|$ elements.

Procedure *BuildAll* requires $O(sn\omega(P))$. In fact for each $l \in M$ procedure *Build*(l) requires $O(n\omega(P))$ and other operations require $O(\omega(P)^2)$.

References

[1] H. Aït-Kaci, "A lattice-theoretic approach to computation based on a calculus of partially ordered types", Ph. D. Dissertation, University of Pennsylvania, 1984.

[2] H. Aït-Kaci, R. Boyer, P. Lincoln and R. Nasr, "Efficient Implementation of Lattice Operations", ACM TOPLAS, 11(1):115-146, Jan 89.

[3] G. Birkhoff, "Lattice Theory", American Mathematical Society Colloquium Publications Vol. 25, (Providence, RI: American Mathematical Society), 1967.

[4] V. Bouchitté, J. Rampon "On-line algorithms for orders", Theoretical Computer Science 175 (1997) 225-238.

[5] Y. Caseau "Efficient Handling of Multiple Inheritance Hierarchies", in OOPSLA '93, pp.271-287, 1993.

[6] B.A. Davey, H.A. Priestley, "Introduction to Lattices and Order", Cambridge University Press, 1991.

[7] L.Forlizzi, E.Nardelli, "Some Results on the Modelling of Spatial Data", 25th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'98), Vol. 1521 of LNCS, Springer Verlag 1998.

[8] L.Forlizzi, E.Nardelli, "On the use of posets as a formal model for spatial data", Technical Report 1/98, Dip. di Matematica, Univ. di L'Aquila, JAN 1998, submitted for publication.

[9] L.Forlizzi, E.Nardelli, "Algorithms for the Mac-Neille completions of posets", Technical Report 10/99, Dip. di Matematica, Univ. di L'Aquila, Apr. 1999.

[10] D.D.Ganguly, C.K.Mohan, S.Ranka, "A Space-and-Time Efficient Coding Algorithm for Lattice Computations", IEEE TKDE, 6(5):819-829, Oct. 1994.

[11] R. Godin, H. Mili, "Building and maintaining analysis-level class hierarchies using Galois lattices", in OOPSLA '93, pp.394-410, 1993.

[12] M.Habib, L.Nourine, “Bit-Vector Encoding for Partially Ordered Sets”, Int. Workshop on Orders, Algorithms and Applications (ORDAL'94), Lyon, France, Jul.94, LNCS 831, V.Bouchitté and M.Morvan (Eds.).

[13] S.C.Hirtle, “Representational Structures for Cognitive Space: Trees, Ordered Trees and Semi-Lattices”, in Spatial Information Theory: A Theoretical Basis for GIS, Vol. 988 of LNCS, Springer Verlag 1995.

[14] G.V. Jourdan, J.X. Rampon, C. Jard, “Computing On-Line the Lattice of Maximal Antichains of Posets”, Order 11:197-210, 1994.

[15] C. Jard, G.V. Jourdan, J.X. Rampon, “Some online computation of the ideal lattice of posets”, IRISA Research Report n.773, 1993.

[16] W. Kaintz, M. Egenhofer, I. Greasley, “Modelling spatial relations and operations with partially ordered sets”, Int. J. of GIS, vol. 7, no. 3, 215-229., 1993.

[17] A. Fiat, G.J. Woeginger, (Eds.), “Online Algorithms”, Vol. 1442 of LNCS, Springer Verlag 1998.

[18] Enrico Nardelli, Vincenzo Mastrobuoni, and Alesiano Santomo. On building the transitive reduction of a two-dimensional poset. *Information Processing Letters*, 63:9–12, 1997.

[19] Enrico Nardelli, Vincenzo Mastrobuoni, and Alesiano Santomo. Computing a poset from its realizer. *Information Processing Letters*, 64:149–154, 1997.

[20] V.Bouchitté, M.Morvan, (Eds.), Proc. of the International Workshop on Orders, Algorithms, and Applications (ORDAL'94), Lyon, France, July 1994, Vol. 831 of LNCS, Springer Verlag, 1994.

[21] L.M. Perry, “Extending (Finite) Partially Ordered Sets to Lattices: An Incremental Approach”, Master's Thesis, Univ. of Maine, Dep. of Surv. Eng., Orono, ME, 1990.

[22] I.Rival, (Ed.), “Algorithms and Orders”, Kluwer Academic Publishers, Dordrecht, 1989.

[23] G. Steiner “An algorithm to generate the ideals of a partial order”, Operation Research Letters volume 5 number 6, 1986.

[24] M. Talamo, P. Vocca, “A Data Structure for Lattice Representation”, TCS, 175(2):373-392, 97.

[25] M.Talamo, P.Vocca, “An Optimal Time*Space Data Structure for Lattices Representation”, to be published on SIAM Journal on Computing.

[26] M.Talamo, P.Vocca, “Optimal digraph search on a sparse representation”, Technical Report 11-98, Dipartimento di Matematica, Univ. of Rome, “Tor Vergata”, 1998. Submitted to Journal of Graph Algorithms and Applications.

[27] M. F. Worboys, “A generic model for planar geographical objects”, INT. J. Geographical Information Systems, Vol 6, NO 5, 353-372, 1992.