ELSEVIER

Theoretical Computer Science 165 (1996) 391–405

Theoretical Computer Science

Tree structure for distributive lattices and its applications

Michel Habib *, Lhouari Nourine

LIRMM (CNRS UMR C09928 & Université Montpellier II), 161, rue Ada, 34392 Montpellier, Cedex 5, France

Received May 1994; revised November 1995
Communicated by M. Nivat

Abstract

From a well-known decomposition theorem, we propose a tree representation for distributive and simplicial lattices. We show how this representation (called ideal tree) can be efficiently computed (linear time in the size of the lattice given by any graph whose transitive closure is the lattice) and compared with respect to time and space complexity. As far as time complexity is concerned, we simply consider the time needed for computations of basic lattice operations such as meet or join and reachability ($x \leqslant y$). Therefore an ideal tree can be considered as a good data structure for a distributive lattice, since for a lattice $L = (X,E)$ it uses $O(|X|)$ space and allows computations of reachability, meet and join operations in $O(|\mathcal{M}(L)|)$, where $\mathcal{M}(L)$ denotes the suborder of the meet irreducible elements in L. Furthermore, optimal bit-vector encoding for distributive lattices can be easily derived from this data structure. Relationships with encoding proposed by Aït-Kaci et al. [3], Caseau [5] are also discussed. Intensive use of this ideal tree allow us to achieve best running time algorithms for most of the applications in which distributive lattices are involved; as for example, constructing the lattice of ideals or generating ideals for a given partial order. Therefore this data structure can be used in many areas such as scheduling theory, in which several algorithms are based on a dynamic programming approach of the lattice of ideals of the precedence ordering; or distributed programming, in which some of the debugging tools rely on the calculation of the lattice of ideals of the causality ordering of the events.

1. Introduction and motivations

The recent growing interest and remarkable progress for algorithms on lattices are due mainly to the importance lattice theory plays in computer science. Distributive lattices are important because they can easily be associated with set families and therefore to data types, see [24]. More recently Aït-Kaci et al. [3] used lattices in the design of programming languages and similarly Caseau [5] to deal with multiple

* Corresponding author. E-mail: (habib,nourine)@lirmm.fr.

PII S0304-3975(95)00232-4

inheritance hierarchies. In both cases a taxonomy of objects or classes is considered. These class/subclass hierarchies or taxonomies play a crucial role in object-oriented languages in which a hierarchical structure is often associated to a type structure. Therefore for the implementation of these languages it is worthwhile to compute efficiently the usual meet and join operations. In such applications sometimes bitwise operations are considered. Similarly, acyclic hierarchies, not necessarily lattices but just directed acyclic graphs, are studied in artificial intelligence (see [8]) for the IS-A hierarchies (or taxonomies) used for knowledge representation, but they are also needed in the databases area in which the hierarchy is simply the structure of the database itself.

In the following we show how to build a tree (named ideal tree) that represents a distributive lattice. Furthermore we show that such a tree, not unique, allows efficient computations for meet and join.

Recent work on ideal trees showed that this tree representation can be used instead of ideal lattice, for example see [18] where it is shown that incremental computation of the tree is enough for debugging of distributed computations. Moreover, this tree gives a natural algorithm for generating ideals of posets. This kind of algorithms can be used as procedure for dynamic programming techniques on the lattice of ideals, see [13]. Finally we hope that ideal trees can be very useful for algorithmic lattice theory.

A partially ordered set $P = (X, \leqslant_P)$ is a reflexive, asymmetric and transitive binary relation on a set X. When necessary, we may consider P as a directed graph (X,E) where $E \subseteq X^2$ and $(x, y) \in E$ iff $x \leqslant_P y$. An order P is called a lattice if the join and the meet exists for every pair of vertices of P.

Let L be a lattice. We denote the set of join-irreducible elements of L by $\mathscr{J}(L)$ and the set of meet-irreducible elements by $\mathscr{M}(L)$. Join-irreducible elements are exactly those elements which cover only one element, and meet-irreducible elements are elements which are covered by only one element in the covering graph of L. For usual definitions on lattices see [7]. Hereafter for our algorithmic purpose, all lattices are supposed to be finite and defined by their directed covering graph. For such a connected graph $G = (X,E)$, $|X| \leqslant |E| + 1$.

2. Tree structure for distributive lattices

Decomposition or factorization properties are well known for distributive lattices, and let us begin with such a decomposition induced by meet and join irreducible elements. This decomposition has produced a linear time algorithm for recognizing an acyclic directed graph whose transitive closure is a distributive lattice, see [14].

Let L be a distributive lattice and $m \in \mathscr{M}(L)$ then there exists a unique join irreducible element j corresponding to the bottom of the sublattice $L\backslash[\perp, m]$. Let us denote by m^* (resp. j^*) the unique element of L that covers m (resp. is covered by j). One can verify easily that $j \vee m = m^*$ and $j \wedge m = j^*$ (this a well known Birkhoff's result [4] which states order-isomorphism between $\mathscr{M}(L)$ and $\mathscr{J}(L)$).

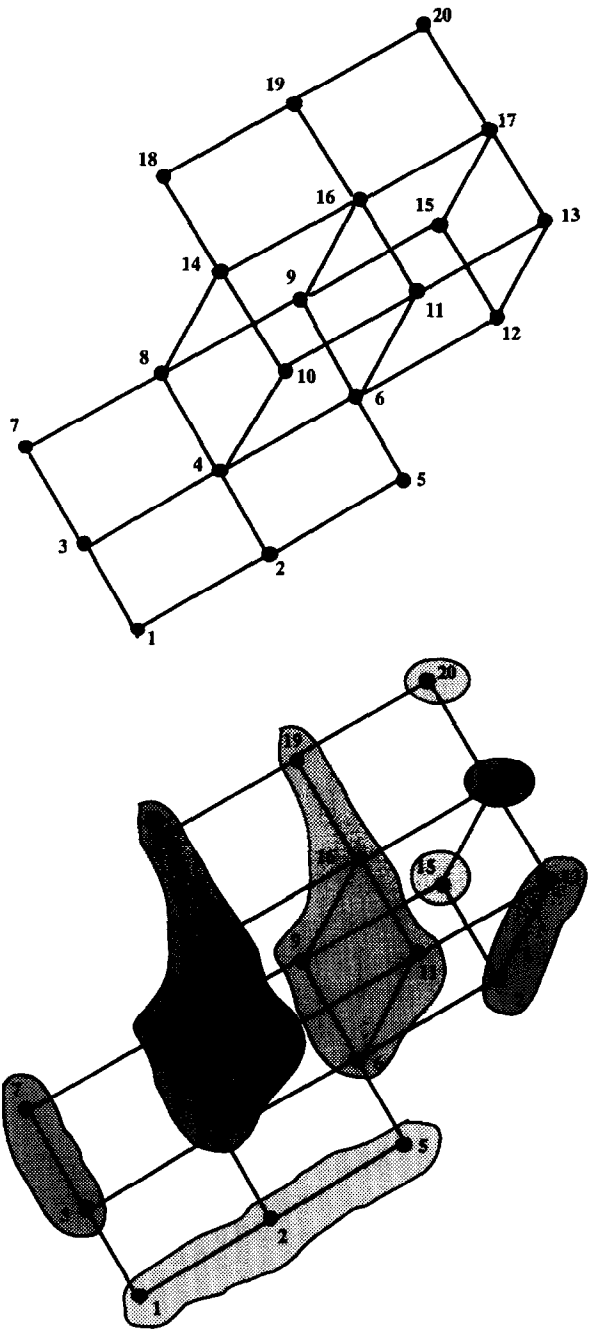

Fig. 1. Example of distributive lattice decomposition with $\tau = 5, 7, 18, 19, 13, 15, 17$.

Theorem 1. *Let L be a distributive lattice and $\tau = m_1, \ldots, m_n$ a linear extension of $\mathcal{M}(L)$. Then*

(i) $[j_1, \mathbf{T}] = L \backslash [j_1^*, m_1]$ *is a distributive lattice, and it exists an order-isomorphism Ψ_{m_1} from $[j_1^*, m_1]$ onto $[j_1, m_1^*]$.*

(ii) $\mathcal{M}([j_1, \mathbf{T}]) = \mathcal{M}(L) \backslash \{m_1\}$.

(iii) $\tau_2 = m_2, \ldots, m_n$ *is a linear extension of* $\mathcal{M}([j_1, T])$.

We denote by L_i the interval $[j_i^*, m_i]$ corresponding to ith element in τ. The isomorphism Ψ_{m_i} is noted Ψ_i related to the interval $[j_i^*, m_i]$. By induction, a decomposition of L into disjoint sublattices $\{L_1, \ldots, L_n, L_{n+1}\}$ can be obtained, where L_{n+1} is reduced to $\mathbf{T}$ and L_i denotes the sublattice $[j_i^*, m_i]$ corresponding to ith element in τ, called a *block*.

Now let us show that the isomorphisms $\{\Psi_i,\ i \in [1..n]\}$ leads to a tree representation of a distributive lattice. Consider a linear extension $\tau = m_1, \ldots, m_n$ of $\mathcal{M}(L)$. Using Theorem 1, we obtain a decomposition of L into sublattices (blocks) $\{L_1, \ldots, L_n, L_{n+1}\}$ where each element of $x \in L_i$ is covered by only one element $y \in L_{i+1} \cup \cdots \cup L_{n+1}$ such that $y = \Psi_i(x)$.

Theorem 2. *The set of edges $A = \{(x, y)$ such that $x \in L_i$ and $y = \Psi_i(x)\ i \in [1..n]\}$ defines a spanning tree of L of height $|\mathcal{M}(L)|$.*

Let us denote by $A(L)$ this tree and refer to as an *ideal tree*.

Proof. Let us show by induction that $A_j = \{(x, y)$ such that $x \in L_i$ and $y = \Psi_i(x)$ for some $i \in [j..n]\}$ is a spanning tree of $L_j \cup \cdots \cup L_{n+1}$.

Clearly A_n is reduced to one edge and then A_n is a spanning tree. Suppose that A_{j+1} is a spanning tree of $L_{j+1} \cup \cdots \cup L_{n+1}$ and let us show that A_j is also a spanning tree of $L_j \cup \cdots \cup L_{n+1}$. From Theorem 1 the sublattice L_j is isomorphic to a sublattice of $L_{j+1} \cup \cdots \cup L_{n+1}$, and each element of L_j is covered by only one element from $L_{j+1} \cup \cdots \cup L_{n+1}$ under the isomorphism Ψ_j. Thus $A_{j+1} \cup \{(x, y)$ such that $x \in L_j$ and $y = \Psi_j(x)\}$ is a tree. Since all edges belong to the covering graph of L, we conclude that A_j is a spanning tree of $L_j \cup \cdots \cup L_{n+1}$. Each step of the linear extension τ increases exactly by one the height of $A(L)$. Therefore height$(A(L)) = |\mathcal{M}(L)| =$ height(L). □

It should be noticed that the ideal tree $A(L)$ contains only covering edges which connect different blocks. This means that the covering edges inside blocks do not belong to $A(L)$. Based on this idea, les us give a simple and linear time algorithm to compute such an ideal tree of a distributive lattice L.

Algorithm 1. Computing the ideal tree of a distributive lattice

Let $G = (X, E)$ be the covering graph of a distributive lattice L.

(1) Compute a linear extension $\tau = \{m_1, \ldots, m_n\}$ of $\mathcal{M}(L)$.

(2) For $i = 1$ to n do

Initially no vertex is marked;
Delete all edges between descendants of m_i not already marked from L;
Mark all descendants of m_i not already marked;

Proposition 1. *Let $G = (X,E)$ be the covering graph of a distributive lattice L. Then Algorithm 1 computes an ideal tree of L in $O(|E|)$ time complexity.*

Proof.

Step 1: Clearly one can compute from G, a linear extension τ of $\mathcal{M}(L)$ in $O(|E|)$.

Step 2: Takes elements of $\mathcal{M}(L)$ according to this linear extension τ. The descendants of the first vertices are visited and marked by deleting all scanned edges. For the next elements, the algorithm only visits vertices which are marked. Thus each vertex is visited at most twice, and therefore Algorithm 1 realizes the required complexity. □

By this process we obtain an ideal tree $A(L)$ depending on the chosen linear extension. Descendants of m_i not yet marked form a block or a sublattice which will be numbered by i. Moreover all these elements have a unique upper cover outside this block.

The remaining of the paper is devoted to give evidence for the following claim: "ideal tree is an efficient data structure for dealing with distributive lattices". To do so, let us describe more precisely this data structure. To each element x of $A(L)$ is associated the following:

BLOCK(x) : the number of the block that x belongs to (i.e. BLOCK(x) = i iff x belongs to L_i).

FATHER(x) : is the unique upper cover of x out of its block.

SON(x) : All elements that have x as unique upper cover out of their block.

Clearly each arc of $A(L)$ can be weighted as follows: $v(x, \text{FATHER}(x)) = \text{BLOCK}(x)$ (see Fig. 2). Therefore to each vertex x one can associate a code, denoted by CODE(x) corresponding to the sequence of valuations obtained by the unique path joining x to the root of $A(L)$. Clearly element codes are sorted increasingly.

Such a tree only requires $O(|X|)$ space memory for a distributive lattice $L = (X,E)$. Later, it will shown from this tree that it is possible to answer queries about reachability, meet and join operations with better costs than those previously obtained using transitive closure or transitive reduction representation.

Observations.

(1) Note that such an ideal tree encoding uses less space memory than the covering graph, since it needs only a space proportional to the number of elements in L. It should also be pointed out that the number of arcs in the transitive reduction is between $|X|$ (L is a chain) and $|X| * \log |X|$ (L is a boolean lattice).

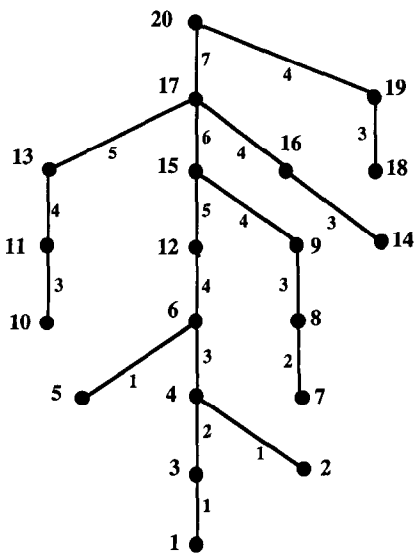

Fig. 2. The ideal tree corresponding to the decomposition shown in Fig. 1.

(2) It is well known, see [4], that any distributive lattice L can be seen as the ideal lattice of the suborder $\mathcal{M}(L)$ (resp. $\mathcal{J}(L)$). Using this remark and the fact that each element of $A(L)$ corresponds to unique element of L, each element of $A(L)$ corresponds to an ideal of $\mathcal{M}(L)$ (resp. $\mathcal{J}(L)$).

(3) The decomposition yielded by this process is not unique, since it depends on the linear extension taken on $\mathcal{M}(L)$. One can notice that if such a decomposition was unique, a linear time-complexity algorithm for isomorphism testing between two distributive lattices could easily be derived. In fact the total number of decompositions is exactly the number of linear extensions of $\mathcal{M}(L)$.

(4) Let x be a vertex with block number k. Then all its descendants have a block number less than k.

(5) Sons of a vertex have different block number.

(6) The code of an element is sorted in increasing order, it corresponds to a filter in $\mathcal{M}(L)$ and its size is at most $|\mathcal{M}(L)|$.

Let us now consider classical operations on lattices such as reachability testing, join and meet.

2.1. Reachability

Property 1. *Let x, y be in L. Three cases have to be distinguished:*

(i) *If* BLOCK(x) > BLOCK(y) *then* $x <_L y$.

(ii) *If* BLOCK(x) = BLOCK(y) *then* ($x \leqslant y \Leftrightarrow$ FATHER(x) $\leqslant$ FATHER(y)).

(iii) *If* BLOCK(x) < BLOCK(y) *then* ($x \leqslant y \Leftrightarrow$ FATHER(x) $\leqslant y$).

Proof. (i) Since the decomposition process follows a linear extension of $\mathcal{M}(L)$, an element with block number i cannot be a descendant of elements with block number $i-1$.

(ii) If BLOCK(x) = BLOCK(y) then x and y belong to the same block. This block is isomorphic to a sublattice containing FATHER(x) and FATHER(y).

(iii) Since there is an isomorphism between the block of x and a sublattice containing FATHER(x), x behaves exactly as its father (i.e. same comparabilities) with regard to higher numbered blocks (i.e. greater than BLOCK(x)). □

2.2. Join computations

Property 2. *Let x, y be in L. Two cases have to be distinguished:*

(i) *If* BLOCK(x) = BLOCK(y) *then* $x \vee y$ *is the son (in the same block as x and y) of* FATHER(x) $\vee$ FATHER(y).

(ii) *If* BLOCK(x) $<$ BLOCK(y) *then* $x \vee y =$ FATHER(x) $\vee y$.

Proof. (i) Let L_k be the block containing x and y. Since L_k is a sublattice of L, therefore $x \vee y$ belongs to L_k, see Fig. 3.

Moreover L_k is lattice-isomorphic to a sublattice containing FATHER(x) and FATHER(y). Therefore FATHER(x) $\vee$ FATHER(y) covers $x \vee y$, and moreover it is the unique son of FATHER(x) $\vee$ FATHER(y) belonging to L_k.

(ii) Let $z =$ FATHER(x) $\vee y$ then we have $x <_L$ FATHER(x) $<_L z$. Clearly $x \vee y$ does not belong to the same block as x, and since every element greater than x out of the block of x is also greater than FATHER(x) then $x \vee y =$ FATHER(x) $\vee y$. □

2.3. Meet computations

Property 3. *Let x, y be in L. Two cases have to be distinguished:*

(i) *If* BLOCK(x) = BLOCK(y) *then* $x \wedge y$ *is the son (in the same block as x and y) of* FATHER(x) $\wedge$ FATHER(y).

(ii) *If* BLOCK(x) $<$ BLOCK(y) *then* $x \wedge y$ *is the son of block* BLOCK(x) *of* FATHER(x) $\wedge y$.

Proof. (i) Similar proof to that of (i) in Property 2.

(ii) Suppose that $x \wedge y = t$. Let $z =$ FATHER(x) $\wedge y$ then $z <_L$ FATHER(x). This implies that z belongs to the sublattice containing FATHER(x) and this sublattice is lattice isomorphic to BLOCK(x). Thus there exists z' in BLOCK(x) and FATHER(z') $= z$. Clearly if $z' <_L t <_L x$ and by the isomorphism FATHER, we have $z <_L$ FATHER(t) $<_L$ FATHER(x) which contradicts $z =$ FATHER(x) $\wedge y$. □

The following corollary (left to the reader) illustrates the classical operations on lattices which can be computed using the previous properties.

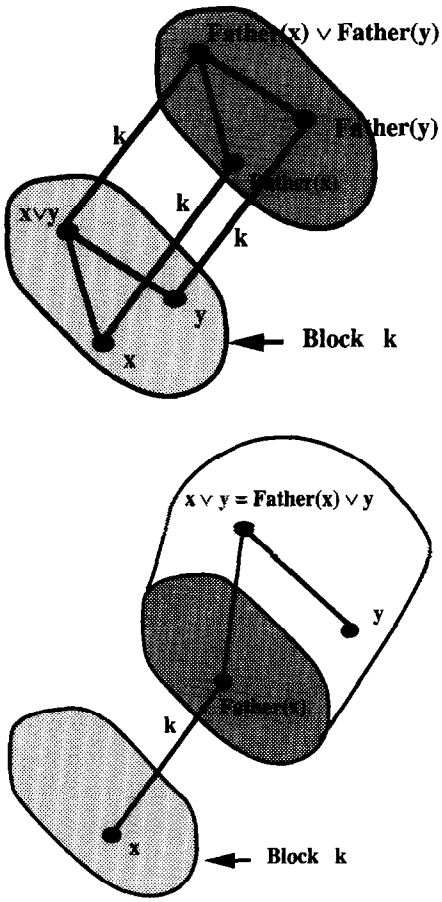

Fig. 3. Computing $x \vee y$.

Corollary 1. *For $x, y \in L$, we have the following equivalences:*

(a) $x \leqslant y$ *iff* $\mathrm{CODE}(x) \supseteq \mathrm{CODE}(y)$,

(b) $x \wedge y$ *is the unique vertex* z *such that* $\mathrm{CODE}(z) = \mathrm{CODE}(x) \cup \mathrm{CODE}(y)$,

(c) $x \vee y$ *is the unique vertex* z *such that* $\mathrm{CODE}(z) = \mathrm{CODE}(x) \cap \mathrm{CODE}(y)$.

From these properties one can easily derive recursive procedures to answer these queries.

Example 1. Let us consider the ideal tree of Fig. 2. For example, let us compute $5 \wedge 7$. Since $\mathrm{BLOCK}(5) < \mathrm{BLOCK}(7)$ then $5 \wedge 7$ can be obtained as the unique son of $6 \wedge 7$ that belongs to block 1. Let us denote by $\mathrm{son}(x, i)$ the unique son, when it exists, of x in block i. Similarly since $\mathrm{BLOCK}(7) < \mathrm{BLOCK}(6)$ then $6 \wedge 7 = \mathrm{son}(6 \wedge 8, 2)$, and $\mathrm{BLOCK}(8) < \mathrm{BLOCK}(6)$ $6 \wedge 8 = \mathrm{son}(6 \wedge 9, 3)$ and $\mathrm{BLOCK}(6) = \mathrm{BLOCK}(9)$ then $6 \wedge 9 = \mathrm{son}(12 \wedge 15, 4)$ and $\mathrm{BLOCK}(12) < \mathrm{BLOCK}(15)$ then $12 \wedge 15 = \mathrm{son}(15 \wedge 15, 5)$,

finally this process terminates, since $15 \wedge 15 = 15$. Backtracking yields: $\mathrm{son}(15,5) = 12$, $\mathrm{son}(12,4) = 6$, $\mathrm{son}(6,3) = 4$, $\mathrm{son}(3,1) = 1$, and we end up with $5 \wedge 7 = 1$.

Proposition 2. *Let $L = (X,E)$ be a distributive lattice. An ideal tree uses $O(|X|)$ space and allows the computation of join and meet in $O(\mathrm{width}(\mathcal{M}(L)) * |\mathcal{M}(L)|)$ time complexity, and reachability in $O(|\mathcal{M}(L)|)$.*

Proof. Clearly the answer for reachability between x and y costs the maximum size of $\mathrm{CODE}(x)$ and $\mathrm{CODE}(y)$ which is at most the number of meet-irreducible elements in L. The computation of the join for x and y can be done in two times. First we compute the code of join using intersection of $\mathrm{CODE}(x)$ and $\mathrm{CODE}(y)$ which can be done in $O(|\mathcal{M}(L)|)$, since the code is sorted. But for a given code, how to find the corresponding element in the ideal tree? Since element codes are sorted, a top-down search can be recursively applied from the root for a son of a given block. So if the list of sons is represented by a linked sorted list then this operation costs $O(|\mathrm{SON}(x)|)$, which is at most the width of $\mathcal{M}(L)$. Since $\mathrm{height}(A(L)) = |\mathcal{M}(L)|$, then join calculations can be achieved in $O(\mathrm{width}(\mathcal{M}(L)) * |\mathcal{M}(L)|)$ time complexity. For meet calculations the result is similar. □

Andrew Fall [9] improves the complexity analysis for join and meet operations.

Proposition 3. *Join and meet operations can be achieved in $O(|\mathcal{M}(L)|)$ time complexity.*

Proof. As in Proposition 2, $\mathrm{CODE}(x \vee y)$ is computed using intersection of $\mathrm{CODE}(x)$ and $\mathrm{CODE}(y)$. Consider a vertex z in the tree with block number k. The block number of any of its descendant must be less than k. Also, if we use the fact that the sons of any node are sorted decreasingly with respect to their block number. Therefore, the block number of any vertex "to the right" of z must also be less than k. Thus any block number is checked at most once during a top-down search to find the unique vertex that matches this $\mathrm{CODE}(x \vee y)$. Since there are exactly $|\mathcal{M}(L)|$ blocks, join and meet can be computed in $O(|\mathcal{M}(L)|)$ time complexity. □

3. Bit-vector encodings

Initially the notion bit-vector encoding comes from boolean lattices. Indeed, we can associate each element of a boolean lattice with a subset such that two elements are comparable if and only if their associated subsets are ordered by inclusion. Bit-vector encodings yielded a very intensive research last few years, for very practical uses of these codes, see, for example, [3, 5, 9, 15].

Ideally, a bit-vector encoding of an order P is a function which assigns to each element x of P a vector of bits such that meet, join and reachability can be done using logical operations on these vectors. This can also be seen as subset of integers (i.e. this subset made up from the vector with those entries equal to 1) denoted by

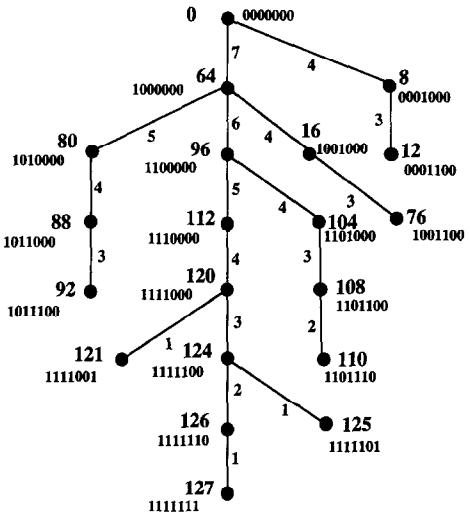

Fig. 4. A numbering of a tree and its binary representation.

SET(x) such that

For all $x, y \in P$ $x <_P y$ if and only if SET(x) $\subset$ SET(y).

In this section we present an encoding based on a labelling of the vertices that allows us to answer reachability in O(1) and efficient algorithms for meet and join operations. This complexity O(1) only depends on the ability to perform the bitwise logical operations as OR, AND and XOR and the representation of integers. In this section we assume that a memory cell of our target machine has more than $|\mathcal{M}(L)|$ bits. To be entirely precise we should have taken as a time unit an elementary operation on one bit. Hereafter we stick to this O(1) convention, which seems to be the most widely used (see [17]).

An obvious remark shows that the size of a bit-vector for a partial order P is at least the size of a maximal chain in P. Let us recall that the minimum size of a boolean lattice in which P can be embedded into, is an order theoretical invariant known as 2-dimension(P) and was already studied by Griggs et al. [11]. Such an embedding produces a bit-vector which allows O(1) reachability computations. Relationships between 2-dimension and partial order encodings have been studied by Habib et al. [12, 15].

Given an ideal tree $A(L)$ of a distributive lattice L, let us start with a labelling algorithm which assigns a number $\ell(x)$, for each vertex x in $A(L)$. The root of $A(L)$ is labelled 0, and a vertex x is labelled $\ell(x) = \ell(\text{FATHER}(x)) + 2^{\text{BLOCK}(x)}$. The computation of this labelling can be done using a top-down breadth-first search on the ideal tree. The size in bits of this labelling is equal to the number of meet-irreducibles in L, that is the height of L or $|\mathcal{M}(L)|$ (Fig. 4).

Let us now consider the following calculations $x \vee y$, $x \wedge y$ and reachability queries, using this bit-vector. Since $\ell(x) = \sum_{i \in \text{CODE}(x)} 2^i$, we have $\ell(x \vee y) = \ell(x)$ AND $\ell(y)$,

Table 1

	Space	Reachability	Meet	Join
Ideal tree	$O(\|X\|)$	$O(\|\mathcal{M}(L)\|)$	$O(\|\mathcal{M}(L)\|)$	$O(\|\mathcal{M}(L)\|)$
Transitive reduction	$O(\|X\| + \|E_{tr}\|)$	$O(\|X\|)$	$O(\|X\| + \|E_{tr}\|)$	$O(\|X\| + \|E_{tr}\|)$
Transitive closure	$O(\|X\| + \|E_{tc}\|)$	$O(\|X\|)$	$O(\|X\|)$	$O(\|X\|)$
Bit-vectors	$O(\|X\|)$	$O(1)$	$O(\log \|X\|)$	$O(\log \|X\|)$

$\ell(x \wedge y) = \ell(x)$ OR $\ell(y)$. To test whether $x \leqslant y$, it suffices to check if $\ell(x \vee y) = \ell(y)$. This can be done, as follows: first, compute $\ell(x \vee y)$ then test if $\ell(x \vee y)$ XOR $\ell(y) = 0$. Therefore reachability queries can be done in $O(1)$ using bit-vectors.

This encoding is optimal for distributive lattices (with respect to reachability) and better than the encoding proposed by Ait-Kaci et al. [3] since the tree representation allows also efficient retrieval for a given bit-vector of the corresponding element in the lattice.

Proposition 3. *For any distributive lattice $L = (X, E)$ an optimal bit-vector encoding of size $O(|\mathcal{M}(L)|)$, can be computed in $O(|X|)$, using an ideal tree representing L. This bit-vector allows reachability in $O(1)$ and $O(|\mathcal{M}(L)|)$ for the join and meet computations.*

Proof. Since $|\mathcal{M}(L)|$ is the size of a maximal chain in L, using the above remark, the bit-vector encoding described above is optimal. Clearly the bit-vector of $x \vee y$ and $x \wedge y$ can be done in constant time using logical operations. But it remains to find the right element associated with this bit-vector. This can be done in $O(|\mathcal{M}(L)|)$ searching along the ideal tree as in Proposition 2. □

Corollary 2. *There exists a data structure using $O(|X|)$ space, which allows to achieve join and meet operations in $O(\log |X|)$ and reachability in $O(1)$.*

Proof. Together with the idea of bit-vectors we add in $O(|X|)$ an ordered array made up with the codes corresponding to the vertices of the ideal tree. In this array we also maintain a pointer from the code to its corresponding vertex in the ideal tree. Therefore to find the join of x and y, we first compute the code of $x \vee y$ in $O(1)$ and then find the corresponding node using a binary search in $O(\log |X|)$. □

Since for any distributive lattice L, $|L| \leqslant 2^{|\mathcal{M}(L)|}$. It should be noticed that $\log |L| \leqslant |\mathcal{M}(L)|$. Table 1 summarizes the performances of known representations for a distributive lattice $L = (X, E)$.

$$|\mathcal{M}(L)| \ll |X| \ll |E_{tr}| \ll |E_{tc}|,$$

$w(\mathcal{M}(L)) =$ maximal degree of a vertex in the transitive reduction of the L.

$=$ width $(\mathcal{M}(L))$.

For the bit-vectors option, they must be of size at least $|\mathcal{M}(L)|$. It should be noticed that in many cases $|X|$ can be exponential in $|\mathcal{M}(L)|$.

Let us now state our main result, which summarizes most of this paper:

Main Theorem. *Let $L = (X,E)$ be the covering graph of a distributive lattice. A tree $T = (X,F)$ can be computed from L in $O(|E|)$ using $O(|X|)$ space. Using T, meet, join and reachability operations can be implemented in $O(|\mathcal{M}(L)|)$. Furthermore if a bit-vector encoding of size $\mathcal{M}(L)$ is chosen then reachability can be checked in $O(1)$, and meet and join only require $O(\log|X|)$.*

Conversely L can be computed from T in $O(|E|)$.

Proof. The first part of this theorem has been shown throughout this paper. It only remains to show the converse, i.e. how to retrieve L from T. To achieve this, we have just to follow the decomposition, gathering the comparabilities in a bottom-up search. This is just a slight modification of the linear transitive closure algorithm produced for distributive lattices (see Habib and Nourine [14]); for more details see Habib et al. [13]. □

4. Ideal tree for simplicial lattices

Similar decomposition can also be found for a larger class of lattices, namely the simplicial lattices. A lattice L is called *meet-simplicial* (resp. *join-simplicial*) (also called join-extremal or meet-extremal by Markowsky [20]) if its length is equal to $|\mathcal{M}(L)|$ (resp. $|\mathcal{J}(L)|$). A lattice L is called *simplicial* if L is meet-simplicial or join-simplicial. This name simplicial comes from a characterization of these lattices by means of a perfect simplicial elimination scheme. Loosely speaking, a lattice is simplicial if and only if its strict incidence bipartite graph admits a perfect simplicial elimination scheme, where a vertex is said to be simplicial if it has at most one neighbour (for details the reader is referred to [22]).

Theorem 3 (Markowsky [20]). *Let L be a meet-simplicial lattice, then it exists a minimal meet-irreducible m of L such that $L\backslash[\bot,m]$ is a meet-simplicial lattice of length $|\mathcal{M}(L)|$-1 and there exists an order-embedding from $[\bot,m]$ to $L\backslash[\bot,m]$.*

Using Theorem 3, there exists a linear extension of $\mathcal{M}(L)$ which guarantees a decomposition of the lattice. This linear extension can be obtained in a greedy fashion, since it corresponds to a perfect elimination scheme. Clearly the main difference with Theorem 1 is that for distributive lattices every linear extension of $\mathcal{M}(L)$ leads to a complete decomposition.

For simplicial lattices the ideal tree does not allow easy computations for the meet and join operations, but it remains possible to compute reachability in $O(|\mathcal{M}(L)|)$, using the collection of order embeddings defined during the decomposition process.

Example 2. Consider the ideal tree of Fig. 5. Let us now try to compute $b \vee c$ and $d \wedge e$ with the previous algorithms. We have CODE($b \vee c$) = $\{1,4\}$ and CODE($d \wedge e$) = $\{1,2,3\}$. Unfortunately there is no vertex having either code $\{1,4\}$ or code $\{1,2,3\}$. The right answers are $b \vee c = \{f\}$ and $d \wedge e = \{a\}$ which have CODE(f) = $\{1\}$ and CODE(a) = $\{1,2,3,4\}$.

The failure for join and meet computations with the previous algorithm in the case of simplicial lattices which are generally not distributive can be understood by the fact that the decomposition is not a lattice homorphism, i.e. is not a lattice embedding.

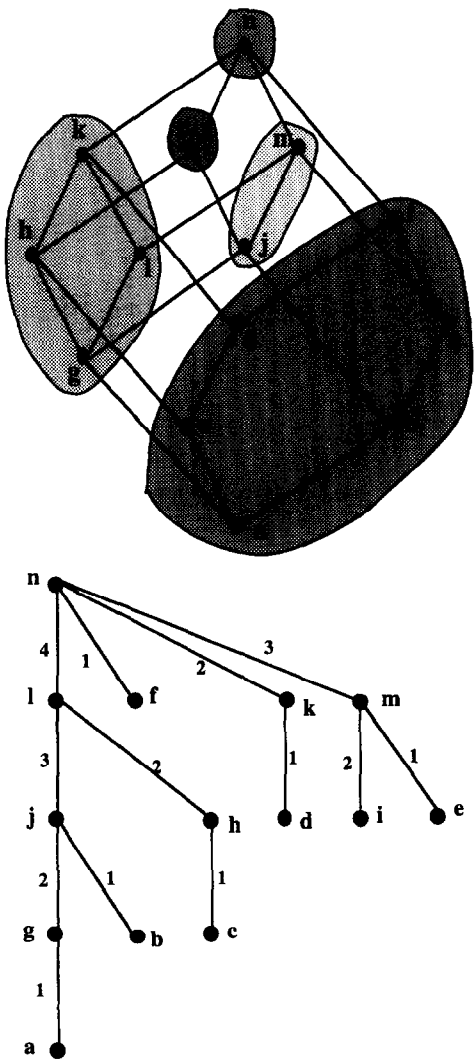

Fig. 5. A meet-simplicial lattice and its associated tree given by the linear extension (f,k,m,i).

Anyway the algorithm for reachability remains correct since the Property 1 is still valid for simplicial lattices. In fact Property 1 only relies on the existence of order embedding associated with the decomposition.

5. Conclusion

Using the correspondence between distributive lattices and ideal lattices of posets, we were able to produce an optimal increamental algorithm to build this ideal lattice of a poset P using the ideal tree. It is also possible to use the ideal tree structure to generate the set $I(P)$ of all ideals of P in $O(\Delta * I(P))$ time complexity (where Δ denotes the maximum indegree in P), see [13]. But the construction of an enumeration algorithm that uses only O(1) per ideal, is still an open problem. We were only able to achieve this result for particular classes of posets, such as interval orders, see [16].

Finally this ideal tree can be used instead of the ideal lattice for analysing distributed computations see Jegou et al. [18].

Acknowledgements

The authors wish to thank Andrew Fall and George Steiner for many stimulating discussions on this paper.

References

[1] R. Agrawal, A. Bordiga and H.V. Jagadish, Efficient management of transitive relationship in large data bases, including IS-A hierarchies, presented at the 1989 ACM Sigmod Conf.

[2] A.V. Aho, M.R. Garey and J.D. Ullman, The transitive reduction of a directed graph, *SIAM J. Comput.* **1** (2) (1972) 131–137.

[3] H. Aït-Kaci, R. Boyer, P. Lincoln and R. Nasr, Efficient implementation of lattice operations, *ACM Trans. Programming Languages Systems* **11** (1989) 115–145.

[4] G. Birkhoff, *Lattice Theory*, Coll. Publ., Vol. XXV (American Mathematical Society, Providence, RI, 3rd ed., 1967).

[5] Y. Caseau, Efficient handling of multiple inheritance hierarchies, preprint 1993, in: *Proc. OOPSLA'93* (ACM Press, New York, 1993) 271–287.

[6] B. Charron-Bost, Mesures de la concurrence et du parallélisme des calculs répartis, Thèse de Doctorat, Université Paris VII, 1989.

[7] B.A. Davey and H.A. Priestley, *Introduction to Lattices and Order* (Cambridge University Press, Cambridge, 1991).

[8] R. Ducournau and M. Habib, Masking and conflicts or to inherit is not to own!, in: M. Lenzerini, D. Nardi and M. Simi, eds., *Inheritance Hierarchies in Knowledge Representation and Programming Languages*, Viareggio, février 1989 (Wiley, New York, 1991) 223–244.

[9] A. Fall, private communication. Simon Fraser University, Canada.

[10] A. Goralcikova and V. Koubek, A reduct-and-closure algorithm for graphs, in: *Proc. MFCS 79* (Springer, Berlin, 1979) 301–307.

[11] J.R. Griggs, J. Stahl and W.T. Trotter, A Sperner theorem on unrelated chains of subsets, *J. Combin. Theory Ser. A* **36** (1984) 124–127.

[12] M. Habib, M. Huchard and L. Nourine, Embedding partially ordered sets into chain-products, in: *Proc. KRUSE'95* (University of Santa-Cruz, 1995) 147–161.

[13] M. Habib, R. Medina, L. Nourine and G. Steiner, Efficient algorithms on distributive lattices, preprint LIRMM 95-033.
[14] M. Habib and L. Nourine, Linear time recognition algorithm for distributive lattices, R.R.LIRMM N°93-013, Mars 1993; submitted.
[15] M. Habib and L. Nourine, Bit-vector encoding for partially ordered sets, in: *Proc. ORDAL'94*, Lecture notes in Computer Science, Vol. 831 (Springer, Berlin, 1994) 1–12.
[16] M. Habib, L. Nourine and G. Steiner, Gray codes for the ideals of interval orders, preprint LIRMM 94-017.
[17] D. Harel and R.E. Tarjan, Fast algorithms for finding nearest common ancestors, *SIAM J. Comput.* **13** (1984) 338–355.
[18] R. Jegou, R. Medina and L. Nourine, Linear space algorithm for on-line detection of global predicates, in: *Proc. Workshop Structures in Concurrency Theory* (Juin 1995).
[19] L. Lamport, Time, clocks, and the ordering of events in a distributed system, *Comm. ACM* **21** (1978) 558–565.
[20] G. Markowsky, Primes, irreducibles and extremal lattices, *Order* **9** (1992) 265–290.
[21] F. Mattern, Virtual time and global states of distributed systems, in: M. Cosnard et al., eds., *Parallel and Distributed Algorithms* (Elsevier, Amsterdam, 1989) 215–226.
[22] M. Morvan and L. Nourine, Simplicial elimination schemes, extremal lattices and maximal antichains lattice, preprint LIRMM 93; *Order*, to appear.
[23] L. Nourine, Quelques Propriétés algorithmiques des Treillis, Thèse de Doctorat, Université de Montpellier II, Juin 1993.
[24] D. Scott, Data types as lattices, *SIAM J. Comput.* (1976) 522–587.
[25] G. Steiner, An Algorithm to generate the ideals of a partial order, *Oper. Res. Lett.* **5** (1986) 317–320.