Concepts can't afford to stammer

Anne Berry* Jean-Paul Bordat† Alain Sigayret*

Abstract: *Generating concepts defined by a binary relation between a set $\mathcal{P}$ of properties and a set $\mathcal{O}$ of objects is one of the important current problems encountered in Data Mining.*

We present a new algorithmic process which generates each concept exactly once, using graph-theoretic results. We present two associated algorithms, both with a good worst-time complexity analysis, which make them competitive with the best existing algorithms.

Our algorithms can be used to compute the edges of the lattice as well as to generate only frequent sets.

1 Introduction

In the context of Data Base Management and Data Mining problems, data bases are often represented by a binary relation between a set $\mathcal{P}$ of properties and a set $\mathcal{O}$ of objects. One of the tools for analyzing the data contained in the base is to compute all possible combinations of elements of the relation into maximal rectangles. These rectangles, called **concepts**, are organized into a hierarchical structure called a Galois lattice or a concept lattice. This theory, though studied by mathematicians as far back as the Nineteenth Century (see [1], [7]), was made popular and developed by Wille and his team ([7]), and remains one of the important current trends of research in Data Mining and Artificial Intelligence: concept lattices are used in fields as varied as the discovery of association rules in Data Bases, the generation of frequent item sets ([16]), machine learning ([10], [12]) and the reorganization of object hierarchies ([9], [3]).

The main drawback to this approach is that a concept lattice is, in general, of exponential size. As a consequence, it is of primary importance to be able to generate concepts efficiently.

Concept generation has given rise to a steady flow of research for the past thirty years. One of the first algorithms to be published in this field is due to Malgrange ([11]); this algorithm generates successive layers of the lattice, by defining possible candidates by combinations of concepts of the previous layer; it has an exponential worst-time complexity per generated concept.

Bordat's algorithm ([5]) was an improvement, as it runs in $O(n^3)$ per concept, where $n = |\mathcal{P}|$, in a Breadth-First fashion; one of the interesting features of this algorithm is that it also computes all the edges of the lattice. This time complexity was recently improved by Nourine and Raynaud ([13]) to $O(n^2)$ per concept.

All these algorithms require exponential space and store the computed concepts.

When the concepts do not need to be stored, but only encountered at least once, the space problem becomes easier, though the existing worst-time complexities per concept are higher: the best such algorithm, due to Ganter ([6]), runs in $O(n^3)$ time per concept, using the interesting notion of *lectic order*, which avoids scanning all the possible subsets of properties, without, however, avoiding re-computing the same concept $O(n)$ times.

In this paper, we address the issue of efficiently computing all the concepts, encountering each exactly once, using only polynomial space.

Our contribution is a Depth-First type algorithm which runs in $O(m)$ time per concept, $O(nm)$ time per maximal chain which is traversed – where m denotes the size of the complement of the relation, with $O(m) \in O(n^2)$, – and requires only small polynomial space ($O(n^2)$). Though the complexities are difficult to compare, this algorithm improves [13] regarding space requirements, as it does not need exponential space; it also significantly improves Ganter's $O(n^3)$ time per concept. Furthermore, the number of concepts tends to become exponential when the relation is dense; in this case, m is of order n, and our complexity becomes $O(n^2)$ per traversed maximal chain.

*LIMOS, bat. ISIMA, 63173 Aubière cedex, France. Mail: berry@isima.fr, sigayret@isima.fr

†LIRMM, 161 Rue Ada, 34392 Montpellier, France. Mail: bordat@lirmm.fr.

The algorithm we introduce here presents similarities with [5]. It uses each concept $A \times B$ to generate the concepts which are just above $A \times B$ in the lattice (called the **cover** of $A \times B$), working at each step on a subrelation.

One of the new features which we add here is that each concept inherits information on the previously processed concepts, in order to avoid generating the same concept more than once, a breakthrough in concept generation.

Another difference is that we use a recursive Depth-First Search of the lattice, which enables to store only a polynomial number of bytes: a concept lattice, though it may be exponentially large, is of small height ($O(n)$). Moreover, the fashion in which we compute the cover of each concept is different.

Our approach is based on our experience on graphs. In [5], Bordat used a directed bipartite graph to handle the relation. In [2], Berry and Sigayret proposed a different encoding into a undirected co-bipartite graph denoted G_R for which they established a one-to-one correspondence between the concepts of the lattice and the minimal separators of the graph. This is algorithmically interesting because, in the past decade, much research has been done on minimal separators.

[2] pointed out that, using the underlying co-bipartite graph and these recent results on the emerging theory of minimal separation, the current best algorithms for generating concepts could easily be matched both in terms of time and space. In this paper, we use graph properties to improve these. Though we will not explicitly use results on minimal separation, they underly our approach (the reader is referred to [2] and [15] for a full explanation on this relationship).

In order to compute the atoms of a concept lattice, we use the graph notion of domination between vertices: a vertex x is said to **dominate** another vertex y if the neighborhood of x includes the neighborhood of y. We use a property from [2]: if all the vertices of a set A share the same neighborhood, and do not together dominated another vertex of the encoding graph G_R, then A defines an atom $A \times B$ of the concept lattice.

Our main complexity improvement follows from the remark that much of the information necessary to determine the domination relationships between vertices can be inherited as one moves up into the lattice, along a path from the bottom to the top (called a **maximal chain**). This enables us to avoid recomputing all the domination information each time a new concept is encountered by the Depth-First process.

We will in the next section give some formal definitions and previous results. In Section 3, we will explain how to compute the cover of a concept, and how to avoid computing a concept several times, and give a first algorithm. In Section 4, we present our data structure, the domination table, give a second algorithm, and illustrate the processes involved with a small example.

2 Preliminaries

2.1 Lattices

Given a finite set $\mathcal{P}$ of properties and a finite set $\mathcal{O}$ of objects, a binary relation R is defined as a subset of the Cartesian product $\mathcal{P} \times \mathcal{O}$. For $X \subseteq \mathcal{P}$, $Y \subseteq \mathcal{O}$, we will denote by $R(X, Y)$ the subrelation $R \cap (X \times Y)$. We will use n to denote $|\mathcal{P}|$, and m to denote the size of the complement of R.

A **concept**, also called a maximal rectangle or closed set of R, is a sub-product $A \times B \subseteq R$ such that $\forall x \in \mathcal{O} - B, \exists y \in A \,|\, (y, x) \notin R$, and $\forall x \in \mathcal{P} - A, \exists y \in B \,|\, (x, y) \notin R$. A is called the **intent** of the concept, B is called the **extent**. The triple $(\mathcal{P}, \mathcal{O}, R)$ is called a **context**.

The concepts, ordered by inclusion on the intents, define a lattice, called a **concept lattice** or Galois lattice. In the rest of this work, we will sometimes denote a concept only by its intent. A lattice is represented by its Hasse diagram: transitivity and reflexivity arcs are omitted. In the rest of this work, when we refer to a path in the lattice, we mean a path in the Hasse diagram of the lattice. Concepts are often referred to as **elements** of this lattice. Such a lattice, has a smallest element, called **bottom**, and a greatest element, called **element**. A path from bottom to top is called a **maximal chain** of the lattice.

We will say that a concept $A' \times B'$ is a **successor** of concept $A \times B$ if $A \subset A'$ and there is no intermediate concept $A'' \times B''$ such that $A \subset A'' \subset A'$. The set of successors of an element is called the **cover** of this element. The successors of the bottom element are called **atoms**. A concept $A' \times B'$ is an **descendant** of concept $A \times B$ if $A \subset A'$. The notions of **predecessor** and **ancestor** are defined dually.

It is important to note that a concept $A' \times B'$ is a descendant of concept $A \times B$ iff $A \subset A'$, and that all the concepts whose intent contains A form a sublattice, the bottom element of which is $A \times B$; this sublattice is isomorphic to the concept lattice of subrelation $R(\mathcal{P} - A, B)$.

Example 2.1 *Binary relation R; the associated concept lattice $\mathcal{L}(R)$ is shown in Figure 1.*

Set of properties:
$\mathcal{P} = \{a, b, c, d, e, f, g, h\}$,

Set of objects:
$\mathcal{O} = \{1, 2, 3, 4, 5, 6\}$.

R	a	b	c	d	e	f	g	h
1		×	×	×	×			
2	×	×	×				×	×
3	×	×				×	×	×
4				×	×			
5			×	×				
6	×							×

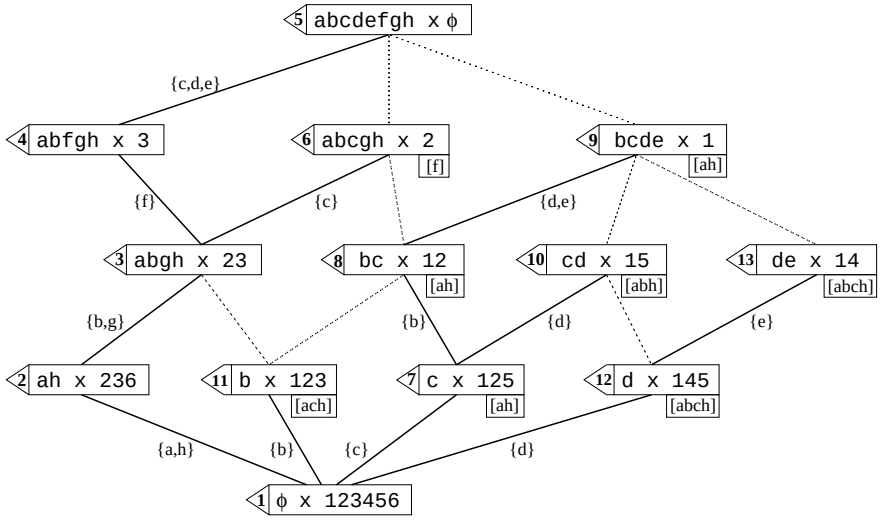

Figure 1: *Concept lattice $\mathcal{L}(R)$ of relation R; the concepts are numbered in prefix order following our sample recursive execution described in Section 4; the inherited sets of already processed vertices appear between brackets. The edges of the Depth-First tree are labeled by the non-dominating maxmod used to compute each new concept.*

2.2 Graphs

Given a relation R, [2] defined an underlying encoding graph G_R as the graph with vertex set $\mathcal{P} \cup \mathcal{O}$; $\mathcal{P}$ and $\mathcal{O}$ are cliques, and for a vertex x of $\mathcal{P}$ and a vertex y of $\mathcal{O}$, there is an xy edge in G_R iff (x, y) is **not** in R. For $X \subset \mathcal{P} \cup \mathcal{O}$, $G_R(X)$ denotes the subgraph of G_R induced by vertex set X. For a concept $A \times B$, we will denote by G_R^A the subgraph $G_R((\mathcal{P}-A) \cup B)$ of G_R associated with subrelation $R((\mathcal{P}-A), B)$.

We will use only external neighborhoods, which we denote by N^+: if $x \in \mathcal{P}, N^+(x) = \{y \in \mathcal{O} | (x, y) \in G\}$, and if $x \in \mathcal{O}, N^+(x) = \{y \in \mathcal{P} | (y, x) \in G\}$, where G is the current subgraph of G_R.

In our example, $N^+(a) = \{1, 4, 5\}$, $N^+(b) = \{4, 5, 6\}$, $N^+(c) = \{3, 4, 6\}$, $N^+(d) = \{2, 3, 6\}$, $N^+(e) = \{2, 3, 5, 6\}$, $N^+(f) = \{1, 2, 4, 5, 6\}$, $N^+(g) = \{1, 4, 5, 6\}$, $N^+(h) = N^+(a)$.

In order to compute the cover of a concept, we need several graph notions.

Definition 2.2 *A vertex x is said to **dominate** a vertex y iff $N^+(y) \subseteq N^+(x)$; we will say that this domination is **strict** when x dominates y and $N^+(y) \neq N^+(x)$.*

Another important notion for this paper is that of maximal clique modules.

Definition 2.3 *A set X of properties is said to form a maximal clique module of G_R if every vertex x of X dominates every other vertex of X, and X is a maximal such set; we will call X a **maxmod**.*

The maxmods of G_R define a partition of $\mathcal{P}$; essentially, a maxmod behaves as a single vertex, as all the vertices of a maxmod X share the same external neighborhood, denoted by $N^+(X)$. We thus extend Definition 2.2 to maxmods: we will say that a maxmod X dominates another maxmod Y if $N^+(Y) \subset N^+(X)$, or, equivalently, if $N^+(y) \subset N^+(x)$ for $x \in X$ and $y \in Y$.

Theorem 2.4 *([2]) A concept $(A + X) \times B'$ covers a concept $A \times B$ iff, in G_R^A, X is a non-dominating maxmod.*

In our example, in G_R, the partition of $\mathcal{P}$ into maxmods is: $\{a,h\}$, $\{b\}$, $\{c\}$, $\{d\}$, $\{e\}$, $\{f\}$, $\{g\}$, $\{h\}$. $\{e\}$ dominates $\{d\}$, $\{f\}$ dominates $\{g\}$, $\{g\}$ dominates $\{a,h\}$ and $\{b\}$; $\{c\}$ is neither dominated nor dominating. The non-dominating maxmods of G_R are: $\{a,h\}$, $\{b\}$, $\{c\}$ and $\{d\}$.

Theorem 2.4 can be used to recursively calculate the cover of an element in a depth-first fashion, starting with the bottom element, by computing, for each concept as it is being processed, the corresponding set of non-dominating maxmods. This process, implemented in a straightforward fashion, will generate each concept exactly as many times as the number of predecessors it has. In this section and in Section 4, we will present two algorithms which use this depth-first recursive approach, but with various improvements.

3 A General Algorithmic Process

We will see in Subsection 3.1 that, for a given concept A, we can generate each non-dominating maxmod of the corresponding subgraph G_R^A in $O(m')$ time, where m' is the number of edges of G_R^A.

We will then go on to explain in Subsection 3.2 how we can store useful information on the already processed concepts in order to generate each concept only once, and give both a corresponding general algorithmic process and a specific algorithm which runs in $O(km')$ per generated concept using the results from Subsection 3.1., where k is at most equal to the number of predecessors of the concept in the lattice.

3.1 Using partition refinement to compute non-dominating maxmods

In this subsection, we explain how to generate the non-dominating maxmods of a graph G with m edges using a partition refinement technique, based on the famous algorithm by Rose, Tarjan and Lueker, known as Lex BFS [14], and on the related work by Hsu and Ma ([8]), which allows us to compute the partition of G into maxmods in $O(m)$ time.

[8] uses partition refinement to compute the partition into maxmods of any chordal graph. The graphs we use are co-bipartite, and in general non-chordal. This refinement process, as well as the associated properties we describe in this subsection, can be extended to an arbitrary graph, while retaining the same $O(m)$ worst time; however, we will describe the process on the co-bipartite graphs we need in our algorithms.

Algorithm OPM
Input: A graph G_R associated with a context $(\mathcal{P}, \mathcal{O}, R)$.
Output: An ordered partition $(K_1, ..., K_p)$ of $\mathcal{P}$ into maxmods of G_R.
PART $\leftarrow$ $(\mathcal{P})$;
for $y \in \mathcal{O}$ **do**
  for each class K of PART such that $|K| > 1$ **do**
    $K' \leftarrow K - N^+(y)$;
    $K'' \leftarrow K \cap N^+(y)$;
    if $K' \neq \emptyset$ and $K'' \neq \emptyset$ **then**
      In PART, replace K by K' followed by K'';
return PART.

This process has several properties.

Property 3.1 *Algorithm OPM computes the partition into maxmods of G_R.*

For an arbitrary graph, the partition refinement process corresponding to OPM also computes the partition into maxmods of the input graph; this is well-known but to our knowledge unpublished.

Property 3.2 *Let S be the set of vertices of $\mathcal{P}$ which are simplicial in G_R, $S \neq \emptyset$; then in the output of Algorithm OPM, $K_1 = S$.*

Note that, given a concept $A \times B$ different from bottom, the set of vertices of $\mathcal{P}$ which are simplicial in G_R^A is empty.

Theorem 3.3 *Algorithm OPM outputs a partition into maxmods as an ordered sequence, which is a linear extension of the partial order on the maxmods: if a maxmod Y dominates a maxmod X, then Y is* after *X in this sequence, which we will call an* ordered partition into maxmods.

According to Theorem 3.3, the first element of the ordered partition is thus a non-dominating maxmod. For a given maxmod X, the set of maxmods which dominate X can be computed in $O(m)$ time; this is done by searching for universal vertices in the subgraph defined by the neighborhood $N^+(X)$ of X; this set of universal vertices is the union of all maxmods which dominate X.

The set of non-dominating maxmods can thus be obtained by repeating the following step:

- Choose the first maxmod X of the ordered partition and remove it;
- Compute the maxmods which dominate X and remove them from the ordered partition.

This leads us to propose a second algorithm, **NDMM**, which extracts from this partition the non-dominating maxmods in $O(m)$ time each.

Algorithm NDMM
Input: A graph G_R, associated with context $(\mathcal{P}, \mathcal{O}, R)$, and an ordered partition PART=$(K_1, ..., K_p)$ of $\mathcal{P}$ into maxmods of G_R.
Output: Partition PART only contains the non dominating maxmods of G_R.
MARKED $\leftarrow \emptyset$;
for i **from** 1 **to** p **do:**
  Choose x in K_i;
  if $x \in$ MARKED
    then Delete K_i from PART
    else $D \leftarrow$ Set of properties from maxmods which dominate K_i in G_R;
      MARKED $\leftarrow$ MARKED $\cup D$;
return PART.

Algorithm OPM will be used both in Subsection 3.2 and in Section 4; NDMM will be the basis of the algorithm proposed in Section 3.2.

3.2 Generating each concept only once

Our stated goal is to generate each concept exactly once. Because the ordering on the concepts is defined by inclusion, any concept $A' \times B'$ which is a descendant of $A \times B$ verifies $A \subset A'$. Since our algorithms work up in a depth-first fashion, when a maxmod X is used to generate a concept $(A + X) \times (B - N^+(X))$, then **all** concepts containing X in their intent will be generated by the recursive call upon $(A + X) \times (B - N^+(X))$. If a concept which is processed later uses a maxmod containing some vertex x of X, we know that this concept has been generated previously. If we are careful to store information on the maxmods which have already been used by the recursive process, we can avoid computing the same concept more than once, using the following step:

General algorithmic process on concept $A \times B$:
  Compute set ND **of non-dominating maxmods of** G_R^A**;**
  // *Each maxmod* X *of* ND *defines a concept covering* $A \times B$.
  Compute set NEW **of maxmods of** ND **containing no already processed vertex;**
  // *Each maxmod* X *of* NEW *defines a new concept covering* $A \times B$.
  For each maxmod X **in** NEW**:**
    Recursively apply process to new concept with intent $A + X$**;**
    ADD all vertices of X **to set of already processed vertices.**

Note that when adding all vertices of X to set of already processed vertices, one could, at no extra cost, also add the set of vertices contained in the maxmods which dominate X, which would improve the running time.

The bottleneck complexity for the above process is computing the set of non-dominating maxmods, and the resulting worst-time analysis will depend on how this is achieved.

Using the process described in Subsection 3.1, which computes each non-dominating maxmod of the current subgraph G_R^A in $O(m')$ time, where m' is the number of edges of this graph, we obtain a complexity of $O(\Delta m')$, where Δ is the number of direct predecessors of A in the lattice.

The corresponding algorithm, presented below, is initially called on the bottom element $(U \times \mathcal{O})$ by CONCEPTS-1$((U \times \mathcal{O}), U)$ on a MARKED set initialized with U, where U is set K_1 from the partition output by an execution of Algorithm OPM on G_R.

Algorithm CONCEPTS-1
Input: A concept $A \times B$, a set MARKED of vertices of $\mathcal{P}$.
Output: The not yet encountered descendants of $A \times B$.
$G \leftarrow G_R^A$;
//1a. *Compute the partition into maxmods.*
PART $\leftarrow$ OPM(G);
//1b. *Select the non dominating maxmods.*
ND $\leftarrow$ NDMM(G_R^A,PART)
// *If desirable, generate the cover of* $A \times B$. //2. *Remove the already processed maxmods.*
NEW $\leftarrow$ ND;
for X **in** NEW **do**
    If $X \cap$ MARKED$\neq \emptyset$ **then**
        Remove the elements of X from NEW;
//3. *Generate the unprocessed descendants of* $A \times B$.
for X **in** *New* **do**
    //*When generating frequent sets, test size of* $B - N^+(X)$*; if too small, take next* X *in New.*
    $G \leftarrow G_R^{A+X}$;
    CONCEPTS-1(G, $(A + X) \times (B - N^+(X))$, MARKED);
    MARKED $\leftarrow$ MARKED $\cup X$;

Experimentally, this algorithm, implemented as explained above, runs rapidly, because of the extra information on the already processed vertices, especially if at each step where a maxmod X is added to the set of already processed vertices, the maxmods which dominate X are also added.

As we will see in the next section, however, the worst-time analysis can still be improved on, using inheritance mechanisms between concepts.

4 Using inherited domination information

In this section, we present a new data structure which enables us to efficiently store information on the domination relation, and avoids recomputing all such information as the depth-first concept generating process moves up along a chain of the lattice. We then give the corresponding algorithm.

4.1 Data structure: the domination table

In order to improve the worse-time behavior of the algorithmic process described above, we propose a more sophisticated approach to computing the non-dominated maxmods, which uses the fact that domination is inherited when moving up in the lattice:

Property 4.1 *Let* $A \times B$ *and* $A' \times B'$ *be concepts, with* $A \subset A'$ *(i.e.* $A' \times B'$ *is a descendant of* $A \times B$*). Then if* x *dominates* y *in* G_R^A *and* $x, y \in G_R^{A'}$*, then* x *dominates* y *in* $G_R^{A'}$.

To efficiently answer requests on the set of non-dominating maxmods, we use a domination table containing information on the current graph. As this information can be inherited along a maximal chain, maintaining this table in the course of the Depth-First traversal along a maximal chain avoids recomputing the entire domination information at each step of the algorithm.

The inheritance mechanism involved is the following: when moving up into the lattice, say from a concept $A \times B$ represented by the underlying graph G_R^A to a second concept $(A + X) \times (B - N^+(X))$, covering the first, and represented by the underlying graph G_R^{A+X}, two things happen:
1. Set X of properties disappear from the graph.
2. Set $N^+(X)$ of objects disappear from the graph.

In our example, when moving up from the bottom element $\emptyset \times \mathcal{O}$ *to element* $ah \times 236$*, the new subgraph will be defined on* $(\mathcal{P} - \{a, h\}) \cup \{2, 3, 6\}$*, so that properties* a *and* h *will disappear, as well as objects* 1, 4 *and* 5.

A vertex x is defined as dominating another vertex y in graph G_R if when there is an yi edge, there also is an xi edge; equivalently, if $(y, i) \notin R$, then $(x, i) \notin R$. Our idea, used to maintain Galois subhierarchies in [3], is to list into a table L, for each pair of properties (x, y), the objects which **prevent** x from dominating y:

Property 4.2 *An object i will appear in the list $L[x,y]$ iff $(x,i) \in R$ and $(y,i) \notin R$.*

The corresponding lists in our example are given in table L below:

L	a	b	**c**	d	e	f	g	h
a	$\emptyset$	$\{1\}$	$\{1,5\}$	$\{1,4,5\}$	$\{1,4\}$	$\emptyset$	$\emptyset$	$\emptyset$
b	$\{6\}$	$\emptyset$	$\{5\}$	$\{4,5\}$	$\{4\}$	$\emptyset$	$\emptyset$	$\{6\}$
c	$\{3,6\}$	$\{3\}$	$\emptyset$	$\{4\}$	$\{4\}$	$\{3\}$	$\{3\}$	$\{3,6\}$
d	$\{2,3,6\}$	$\{2,3\}$	$\{2\}$	$\emptyset$	$\emptyset$	$\{3\}$	$\{2,3\}$	$\{2,3,6\}$
e	$\{2,3,6\}$	$\{2,3\}$	$\{2,5\}$	$\{5\}$	$\emptyset$	$\{3\}$	$\{2,3\}$	$\{2,3,6\}$
f	$\{2,6\}$	$\{1,2\}$	$\{1,2,5\}$	$\{1,4,5\}$	$\{1,4\}$	$\emptyset$	$\{2\}$	$\{2,6\}$
g	$\{6\}$	$\{1\}$	$\{1,5\}$	$\{1,4,5\}$	$\{1,4\}$	$\emptyset$	$\emptyset$	$\{6\}$
h	$\emptyset$	$\{1\}$	$\{1,5\}$	$\{1,4,5\}$	$\{1,4\}$	$\emptyset$	$\emptyset$	$\emptyset$

From this table L, we can see that c will dominate a when objects 1 *and* 5 *have disappeared as* $L[c,a] = \{1,5\}$ *(read column* **c** *and row* **a***). $a1$ and $a5$ are edges of G_R, $c1$ and $c5$ are not.*

Table L contains at most nm bits, since for each slot $L[x,y]$, the elements of the list it contains correspond to distinct neighbors of y in G_R, and each row y contains n such lists.

When moving up from one concept to one of its successors in our recursive algorithmic process, updating table L means for each (x,y)-pair in $\mathcal{P}^2$, removing from list $L[x,y]$ the objects which disappear from the graph.

Actually, we are only concerned with the **number** of vertices which a vertex x dominates in a given graph, so that cardinalities are sufficient for our data structure: a maxmod X will be non-dominating when, for any $x \in X$, the number of vertices which x dominates is exactly $|X|$. We will thus use L as an underlying abstract data type, and implement it with a cardinality table T, which contains numbers between 0 and $|\mathcal{O}|$, $T[x,y]$ representing the size of list $L[x,y]$. Vertex x dominates vertex y iff $T[x,y] = 0$.

In order to have rapid access to this information, we also keep a table D, scanning $\mathcal{P}$, where $D[x]$ gives the number of vertices y such that $T[x,y] = 0$, i.e. the number of vertices which x dominates. A maxmod X will thus be non-dominating if and only if for an arbitrary $x \in X$, $D[x] = |X|$, and the query: *'Which are the non-dominating maxmods?'* can be answered in very efficient $O(n)$ time using table D.

In our example, tables T and D would be:

T	a	b	c	d	e	f	g	h
a	0	1	2	3	2	0	0	0
b	1	0	1	2	1	0	0	1
c	2	1	0	1	1	1	1	2
d	3	2	1	0	0	1	2	3
e	3	2	2	1	0	1	2	3
f	2	2	3	3	2	0	1	2
g	1	1	2	3	2	0	0	1
h	0	1	2	3	2	0	0	0

D:

a	b	c	d	e	f	g	h
2	1	1	1	2	5	4	2

We have: $T[a,e] = 3$ (read column a and row e) and $T[e,a] = 2$,

$D[a] = 2$ as $T[a,a] = T[a,h] = 0$.

The process for constructing the initial domination table T from a table T initialized to containing zero values is the following:

For each x **in** $\mathcal{P}$ **do**
  $D[x] \leftarrow n$;
  For each y **in** $\mathcal{P}$ **do**
    For each z **in** $\mathcal{O}$ **do**
      If $(x,z) \in R$ and $(y,z) \notin R$ **then**
        If $T[x,y] = 0$ **then** $D[x] \leftarrow D[x] - 1$;
        $T[x,y] \leftarrow T[x,y] + 1$;

4.2 Algorithmic use of the domination table

We will use the data structures described in the previous subsection to implement the UPDATE primitive in the algorithm we now present: CONCEPTS-2.

The algorithm is initially called on the bottom element $(U \times \mathcal{O})$ by CONCEPTS-2$((U \times \mathcal{O}), U)$ on a MARKED set initialized with U, where U is set K_1 from the partition output by an execution of Algorithm

OPM on G_R. Tables T and D are initialized from G_R as described in the previous subsection.

Algorithm CONCEPTS-2
Input: A concept $A \times B$, a set MARKED of vertices of $\mathcal{P}$.
Output: The not yet encountered descendants of $A \times B$.
Initialization:
    $G \leftarrow G_R^A$;
    Compute, using **OPM**, the partition of $\mathcal{P} - \mathcal{A}$ in G into maxmods;
    // *The maxmod which a vertex $x \in \mathcal{P}$ belongs to is denoted by $M(x)$.*
    For x **in** MARKED **do** MARKED $\leftarrow$ MARKED $\cup M(x)$;
//1. *Compute the set* ND *of non-dominating maxmods of G.*
ND $\leftarrow \emptyset$;
for x **in** $\mathcal{P} - A$ **do**
    if $D[x] = |M(x)|$ **then** ND $\leftarrow$ ND $\cup \{M(x)\}$;
//2. *If desirable, generate the cover of $A \times B$.*
//3. *Generate the unprocessed descendants of $A \times B$.*
for X **in** ND such that $X \cap$ MARKED $= \emptyset$ **do**
    $A' \leftarrow A + X$; $B' \leftarrow \mathcal{O} - N^+(X)$;
    PRINT($A' \times B'$);
    //*When generating frequent sets, test size of B'; if too small, take next X in* ND–MARKED.
    UPDATE(A , X , G_R^A , *pre*);
    CONCEPTS-2($A' \times B'$, MARKED);
    UPDATE(A , X , G_R^A , *post*);
    MARKED $\leftarrow$ MARKED $\cup X$;

Algorithm UPDATE
Input: Concept A, a non dominating maxmod X of G_R^A,
and a variable V set to *pre* for pre-updating or to *post* for post-updating.
// *Tables T and D are global variables.*
Output: Tables T and D are modified using X and A.
Choose a representative x **in** X;
//1. *Update table D.*
for y **in** $(\mathcal{P} - A) - X$ **do**
    if $T[y, x] = 0$ **then**
        if $V = pre$ **then** $D[y] \leftarrow D[y] - |X|$;
        else $D[y] \leftarrow D[y] + |X|$;
//2. *Update tables T and D.*
for j **in** $N^+(x)$ **do**
    $Z \leftarrow N^+(j) - X$;
    $U \leftarrow (\mathcal{P} - A) - Z - X$;
    for (u, z) **in** $U \times Z$ **do**
        if $V = pre$ **then**
            $T[u, z] \leftarrow T[u, z] - 1$;
            if $T[u, z] = 0$ **then** $D[u] \leftarrow D[u] + 1$;
        else // V=*post*
            $T[u, z] \leftarrow T[u, z] + 1$;
            if $T[u, z] = 1$ **then** $D[u] \leftarrow D[u] - 1$;

4.3 Complexity Analysis

We will first evaluate the worst-time complexity required by the main algorithm, and then examine the time required by the updating process.

- Each step of Algorithm CONCEPTS-2 requires computing the subgraph $G = G_R^A$ and its maxmods, which can be done with Algorithm OPM in $O(m')$, where m' is the number of edges of G, as seen in Subsection 3.1. Using table D, finding the set of non-dominating maxmods requires $O(n')$ time, where $n' = |\mathcal{P} - A|$. Comparing these with MARKED costs $O(n')$ time, thus a concept is processed in global $O(m')$ time.

- Tables T and D are pre-updated at each step to describe the domination relationships in the new graph before a recursive call, and then post-updated back to their original form. Clearly, the costs of the pre-updating and post-updating processes are exactly the same.

  We will now discuss the cost of the pre-updating process when moving from concept $A \times B$ to its successor $A' \times B' = (A + X) \times (B - N^+(X))$, obtained from non-dominating maxmod X of G_R^A.

  We need to evaluate the number of unit decrementations on T at each step. This corresponds to the number of object removals from lists in L. Pre-updating means removing from L all objects i such that i fails to be in the successor, i.e. $i \in (B - N^+(X))$.

  By Property 4.2, an object i will appear in the list $L[x, y]$ iff $(x, i) \in R$ and $(y, i) \notin R$, which can be translated as: $(x, i) \notin G_R^A$ and $(y, i) \in G_R^A$.

  Since we are generating subgraph $G_R^{A'}$, we do not need the elements of A'; thus, the effort required is:

  $|\mathcal{P} - A)|.\Sigma_{i \in (B - N^+(X)}|N^+(i)|$,

  where $N^+(X)$ is the neighborhood in subgraph G_R^A.

  Note that $(B - N^+(X)) < |\mathcal{O}|$, $|\mathcal{P} - A|$ is of order n, and $|N^+(i)| < n$.

  Let τ be the spanning tree of the lattice induced by the recursive calls of CONCEPTS-2; the global time required for pre-updating T along a root-to-leaf traversal of τ is bounded by $O(nm)$.

Since a concept $A \times B$ obviously has only one father in τ, computing $A \times B$ requires generating just one edge of the lattice as described above. The time required for processing concept $A \times B$ is thus in $O(m' + |\mathcal{P} - A)|.\Sigma_{i \in (B - N^+(X)}|N^+(i)|$, which is of order m' plus n' times the number of objects which have disappeared when moving up in τ from $A \times B$'s father.

Since, along a path from root to leaf in τ, no object can disappear twice, the global time complexity of the algorithm is bounded by $O(m)$ per concept plus $O(nm)$ per traversed maximal chain of the lattice, though this is very rough compared to the complexity analysis detailed above.

We will end with the space complexity: the recursive queue contains at most $O(n)$ concepts of size $O(n)$ each, MARKED is of size $O(n)$; T contains $O(nm)$ bits; the global space complexity is thus in $O(nm)$.

Example 4.3 *Let us execute Algorithm CONCEPTS-2 on relation R of Example 2.1, associated with concept lattice of Figure 1 (page 3).*
Step 1: The execution starts with the bottom element $\emptyset \times 123456$. In $G = G_R$, the non-dominating maxmods are $\{a, h\}$, $\{b\}$, $\{c\}$ and $\{d\}$. The cover of $\emptyset \times 123456$ is: $ah \times 236$, $b \times 123$, $c \times 125$, $d \times 145$. The set MARKED of already processed vertices is empty. $ah \times 236$ is chosen to be processed next.
Step 2: Concept $ah \times 236$ is chosen to be processed next; the table is accordingly pre-updated: since objects 1, 4 and 5 disappear, pairs from the Cartesian products $\{b, c, d, e\} \times \{f, g\}$, $\{d, e\} \times \{b, c, f, g\}$ and $\{c, d\} \times \{b, e, f, g\}$ should cause the corresponding numbers from T do be decremented by 1, since in the current subgraph, $(\mathcal{P} - \{a, h\}) = \{b, c, d, e, f, g\}$, $N^+(1) = \{f, g\}$, $N^+(4) = \{b, c, f, g\}$ and $N^+(5) = \{b, e, f, g\}$.
New tables T and D obtained:

T	b	c	d	e	f	g
b	0	0	0	0	0	0
c	1	0	0	0	1	1
d	2	1	0	0	1	2
e	2	1	0	0	1	2
f	1	1	0	0	0	1
g	0	0	0	0	0	0

D:

b	c	d	e	f	g
2	3	6	6	3	2

(for $T[x, y]$, read column x and row y)

Graph G becomes $G_R(\{b, c, d, e, f, g, 2, 3, 6\})$. Maxmods of G: $\{b, g\}$, $\{c\}$, $\{d, e\}$, $\{f\}$; non-dominating maxmod: $\{b, g\}$. Concept $abgh \times 23$ is generated.
Step 3: $abgh \times 23$ is processed. Non-dominating maxmods: $\{c\}$ and $\{f\}$. Concepts $abcgh \times 2$ and $abfgh \times 3$ are generated; $abfgh \times 3$ is chosen to be processed next.
Step 4: $abfgh \times 3$ is processed. Non-dominating maxmod: $\{c, d, e\}$; top element $abcdefgh \times \emptyset$ is generated.
Step 5: $abcdefgh \times \emptyset$ is processed; the graph G obtained is empty; no new concept can be generated.
Step 6: step 3 recursively calls $abcgh \times 2$, with MARKED=$\{f\}$. Non-dominating maxmod: $\{d, e, f\}$; since f is in MARKED, no new concept is generated.
Step 7: step 1 recursively calls $c \times 125$ with MARKED=$\{a, h\}$. Non-dominating maxmods: $\{b\}$ and $\{d\}$. Concepts $bc \times 12$ and $cd \times 15$ are generated. $bc \times 12$ is chosen to be processed next.

Step 8: $bc \times 12$ is processed, with MARKED=$\{a, h\}$. Non-dominating maxmods: $\{d, e\}$ and $\{a, g, h\}$; since a and h are in MARKED, only $\{d, e\}$ will be used to generate a new concept: $bcde \times 1$.
Step 9: $bcde \times 1$ is processed, with MARKED=$\{a, h\}$. Non-dominating maxmod: $\{a, f, g, h\}$; since a and h are in MARKED, no new concept is generated.
Step 10: step 7 recursively calls $cd \times 15$, with MARKED=$\{a, b, h\}$; $\{a, h\}$ is inherited from concept $ah \times 236$, a brother of father $c \times 125$, and $\{b\}$ is inherited from brother concept $bc \times 12$. Non-dominating maxmod: $\{b, e\}$. Since b is in MARKED, no new concept is generated.
Step 11: step 1 recursively calls $b \times 123$ with MARKED=$\{a, c, h\}$. Non-dominating maxmods: $\{a, g, h\}$ and $\{c\}$. Since a, c and h are in MARKED, no new concept is generated.
Step 12: step 1 recursively calls $d \times 145$ with MARKED=$\{a, b, c, h\}$. Non-dominating maxmods: $\{c\}$ and $\{e\}$. Since c is in MARKED, only concept $de \times 14$ is generated.
Step 13: $de \times 14$ is processed, with MARKED=$\{a, b, c, h\}$. Non-dominating maxmod: $\{b, c\}$. Since b and c are in MARKED, no new concept is generated. The recursive queue is empty and the algorithm terminates.

5 Conclusion

In this paper, we use a graph-theoretic approach which translates the concepts on a maximal chain of a lattice into a sandwich family of graphs, where domination is inherited. This enables us to propose new algorithmic processes, which are capable generating each concept exactly once. Our complexity analysis involves traversed maximal chains of the lattice, so that our complexity is difficult to compare with that of the other existing algorithms, which are evaluated only regarding concepts. However, even with a rough analysis, our complexity is competitive. As a final concluding remark, we believe that our complexity analysis could be simplified and streamlined or even amortized.

Acknowledgement

We heartily thank Eric SanJuan for encouraging us in this work, and for many interesting and fruitful discussions on concept generation and its mathematical aspects.

References

[1] M. Barbut and B. Monjardet. *Ordre et classification.* Classiques Hachette, 1970.

[2] A. Berry and A. Sigayret. *Representing a concept lattice by a graph.* Proceedings of Discrete Maths and Data Mining Workshop, 2nd SIAM Conference on Data Mining (SDM'02), Arlington (VA), April 2002, submitted to Discrete Applied Mathematics.

[3] A. Berry and A. Sigayret. *Maintaining class membership information.* To appear in the proceedings of OOIS'02, Lecture Notes in Computer Science, Sept. 2002.

[4] G. Birkhoff. *Lattice Theory.* American Mathematical Society, 3rd Edition, 1967.

[5] J.-P. Bordat. *Calcul pratique du treillis de Galois d'une correspondance.* Mathématiques, Informatique et Sciences Humaines, 96:31–47, 1986.

[6] B. Ganter. *Two basic algorithms in concept analysis.* Preprint 831, Technische Hochschule Darmstadt, 1984.

[7] B. Ganter and R. Wille. *Formal Concept Analysis.* Springer, 1999.

[8] W.-L. Hsu and T.-H. Ma. *Substitution decomposition on chordal graphs and its applications.* SIAM Journal on Computing, 28:1004-1020, 1999.

[9] M. Huchard, H. Dicky and H. Leblanc. *Galois lattice as a framework to specify building class hierarchies algorithms.* Theoretical Informatics and Applications, 34:521–548, 2000.

[10] M.Liquiere and J.Sallantin. *Structural machine learning with Galois lattices and Graphs.* Proc. of the 1998 Int. Conf. on Machine Learning (ICML'98) Morgan Kaufmann Ed, 305-313.

[11] Y. Malgrange. *Recherche des sous-matrices premières d'une matrice à coefficients binaires.* Deuxième congrès de l'AFCALTI, Octobre 1961, Gauthier-Villars, 1961.

[12] E. Mephu Nguifo and P. Njiwoua. *Using Lattice-Based Framework as a Tool for Feature Extraction.* ECML, 304–309, 1998.

[13] L. Nourine and O. Raynaud. *A Fast Algorithm for building Lattices.* IPL, 71, 199–204, 1999.

[14] D. Rose, R. E. Tarjan and G. Lueker. *Algorithmic aspects of vertex elimination on graphs.* SIAM J. Comput., 5, 146–160, 1976.

[15] A. Sigayret. *Data mining: une approche par les graphes.* PhD Thesis, DU 1405 - EDSPIC 269, Université Clermont-Ferrand II (France), 2002.

[16] P. Valtchef, R. Missaoui, and R. Godin. *A Framework for Incremental Generation of Frequent Closed Item Sets.* Proceedings of Discrete Maths and Data Mining Workshop, 2nd SIAM Conference on Data Mining (SDM'02), Arlington (VA), April 2002.