# A local approach to concept generation

Anne Berry* Jean-Paul Bordat† Alain Sigayret*

16th January 2006

**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 and Knowledge Discovery in Databases.**

**We present a new algorithmic process which computes all the concepts, without requiring an exponential-size data structure, and with a good worst-time complexity analysis, which makes it competitive with the best existing algorithms for this problem. Our algorithm can be used to compute the edges of the lattice as well at no extra cost.**

**Keywords:** Concept Lattice, Galois lattice, maximal rectangles, concept generation.

## 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 ways of analyzing the data contained in the base is to examine 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 **concept lattice**. This theory, though studied by mathematicians as far back as the nineteenth century (see [1]), was made popular and developed by Wille and his team ([12]), 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 ([25]), the generation of frequent item sets ([30]), machine learning ([20], [22]), software engineering ([16]) and the reorganization of object hierarchies ([15], [4]).

A concept lattice is, in general, of exponential size. As a consequence, it is of primary importance to be able to generate each concept efficiently. One of

*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 problems is to avoid multiple generation of the same concept. This requires either storing all the previously generated concepts and running through this set at each step of the algorithm, using an efficient data structure; this approach requires exponential time. Another solution is to not store all the concepts, but nevertheless find a technique to avoid redundancy.

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 ([21]); 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; a similar process was later presented by Chein ([7]). These algorithms require exponential time per generated concept, as at each step all previously generated concepts need to be compared with the current candidate. Norris ([23]) improved these with an incremental approach. All these algorithms require exponential space.

Ganter's algorithm ([11]) was an important improvement, as it runs fast ($O(|\mathcal{P}|^2|\mathcal{O}|)$ per concept) without requiring exponential space; it introduces the interesting notion of *lectic order*, which avoids scanning all the possible subsets of properties, without, however, avoiding re-computing the same concept $O(|\mathcal{P}|)$ times. One of its drawbacks is that it does not compute the edges of the Hasse diagram of the lattice.

Bordat's algorithm ([6]) can run slightly faster ($O(|\mathcal{P}+\mathcal{O}|^{2.376})$ per concept), and can be implemented to run in the same time as Ganter's algorithm. Bordat's algorithm is very different from the previous ones, as it works with a local approach: given any concept and the relation, it can generate all the successors of this concept (called the **cover** of the concept) in either $O(|\mathcal{P}+\mathcal{O}|^{2.376})$ or $O(|\mathcal{P}|^2|\mathcal{O}|)$. This enables it to generate, at no extra cost, the edges of the lattice, which is often important to compute in applications. It introduces a data structure to store the already computed concepts, which requires only $O(|\mathcal{P}|)$ time to check whether a given concept has already been generated.

This time complexity was recently improved by Nourine and Raynaud ([24]) to $O(|\mathcal{P}|^2)$ per concept, using an approach based on Norris' algorithm, with an exponential data structure different from Bordat's to store the concepts; however, it has been shown to perform poorly in practise, as illustrated by Kuznetsov and Obiedkov's survey on concept generation algorithms ([18]).

Many other algorithmic processes now exist, such as [19], as well as many specialized algorithms to deal with practical data, using for example incremental approaches or a decomposition of the data (see e.g. [31], [9]).

The modern concept generation algorithms can be put into one of two classes: either they do not require exponential space, and the best such algorithms is Ganter's in $O(|\mathcal{P}|^2|\mathcal{O}|)$ per concept. Or they use exponential space, and they run faster, at least in the worst case: $O(|\mathcal{P}+\mathcal{O}|^{2.376})$ for Bordat's and $O((|\mathcal{O}|+(|\mathcal{P}|)(|\mathcal{O}|)$ for Nourine and Raynaud's.

In this paper, we address the issue of examining Bordat's approach more closely. Various implementations have been made for Bordat's algorithm; the original paper presents a Breadth-First Search, but the algorithm can also be used in a Depth-First fashion. In this paper, we formally show that Bordat's

algorithm can run without its exponential data structure, and in fact requires only polynomial space.

Our approach is based on our experience on graphs. In [6], Bordat used a directed tripartite graph to handle the relation. In [3], Berry and Sigayret proposed a different encoding into an undirected co-bipartite graph for which they established a one-to-one correspondence between the concepts of the lattice and the minimal separators of the graph (the reader is referred to [3] and [29] for a full explanation on this relationship.) This is algorithmically promising because, in the past decade, much research has been done on minimal separator generation (see [17], [28], [2]) as well as on the structural properties of minimal separators. [3] 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 particular, they showed that generating the concepts defined by a binary relation is equivalent to generating the minimal ab-separators of a corresponding graph: they thus matched the best complexity due to Nourine and Raynaud by using the minimal ab-separator algorithm due to Shen and al. ([27]), which claims the same $O((|\mathcal{P}| + |\mathcal{O}|)^2)$ time per generated object. Though we will not explicitly use results on minimal separation, they underly our approach; in our definitions, we will retain graph terms such as 'domination' and 'maxmod'.

The algorithmic process we use here presents similarities with [6]. It uses each concept $A \times B$ to generate its cover in the lattice, working at each step on a subrelation. However, the fashion in which we compute the cover of each concept is different and more efficient.

Our contribution in this paper is threefold:

- First, we present an efficient linear-time process, based on graph partition-refinement techniques, to group together the properties which correspond to similar columns in the relation. This is important, because it is a mandatory first step in computing the cover of an element, as similar properties will always appear together.

- Second, we explain how to let each concept inherit information on the previously processed concepts, in order to avoid generating the same concept more than once, a breakthrough in concept generation. This is important, because as discussed above, the process presented in Bordat's original paper (as well as most of the other concept generation algorithms) requires a data structure of exponential size to check whether a concept has already been generated. In the present paper, we are able to preserve Bordat's good complexity and strong structural approach without requiring any data structure.

- Third, we note that much useful information can be inherited by a concept from its predecessors, which enables us to avoid recomputing all the information at each step in the course of a Depth-First traversal, using

a polynomial-sized data structure which is transmitted from 'father concept' to 'son', thus improving the running time to $O(|\mathcal{P}||\mathcal{O}|)$ time per generated concept, plus a cost of $O(|\mathcal{P}|^2|\mathcal{O}|)$ time per maximal branch of the recursive tree induced by the Depth-First search.

The paper is organized as follows: in Section 2, we give some preliminary definitions. In Section 3, we explain our approach in more detail, illustrating it with a simple example, which we use thoughout the paper. We then show how to partition the properties into classes which share the same columns, and then go on to show how we avoid processing a concept several times. We propose a general process which simplifies and improves Bordat's algorithm. In Section 4, we further improve this by using our new data structure, the *domination table*, to transmit inherited information when moving up along a chain of the lattice.

# 2 Preliminaries

In all this work, $\subset$ denotes strict inclusion, as opposed to $\subseteq$. If the intersection of two sets is empty, we will often use $+$ to denote their union. For notions on ordered sets not defined here, the Reader is referrred to [5] and to [1].

Given a finite set $\mathcal{P}$ of properties (which we will denote by lowercase letters in our examples) and a finite set $\mathcal{O}$ of objects (which we will denote by numbers), a *binary relation* $R$ is defined as a subset of the Cartesian product $\mathcal{P} \times \mathcal{O}$. $\overline{R}$ is the complement of relation $R$ (i.e. $\overline{R} = (\mathcal{P} \times \mathcal{O}) - R$). The triple $(\mathcal{P}, \mathcal{O}, R)$ is called a **context**. Such a relation is often represented by a table, where elements of $R$ are denoted by crosses or by ones. Given an element of $\mathcal{P} \times \mathcal{O}$, we will often refer to it as a *one* if it is in $R$, and as a *zero* if it is not. $|R|$ will denote the number of ones, and $|\overline{R}|$ the number of zeroes. We will use $n$ to denote $|\mathcal{P} + \mathcal{O}|$. For a *property* $x$, we will denote $R[x]$ the set of objets which $R$ puts in relation with $x$: $R[x] = \{y \in \mathcal{O} | (x, y) \in R\}$; for an *object* x, $R[x]$ will be defined in a similar way: $R[x] = \{y \in \mathcal{P} | (y, x) \in R\}$. If $X$ is a set of properties or a set of objects, we will denote $R[X]$ the set $\bigcap_{x \in X} R[x]$. For $X \subseteq \mathcal{P}$, $Y \subseteq \mathcal{O}$, we will denote by $R(X, Y)$ the subrelation $R \cap (X \times Y)$.

A **concept**, also called a **maximal rectangle** or closed set of $R$, is a subproduct $A \times B \subseteq R$ 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**. Brackets are often omitted when denoting intents and extents. Given the intent $A$ of a concept, it is easy to compute its extent: $B = R[A]$. Thus in the rest of this work, we will often denote a concept only by its intent.

The concepts, ordered by inclusion on the intents, or, dually, by inclusion in the extents, define a lattice, called a **concept lattice** or **Galois lattice**. In such a lattice, given two concepts $A_1 \times B_1$ and $A_2 \times B_2$, then $A_1 \subset A_2$ iff $B_2 \subset A_2$. $\mathcal{P}$ and $\mathcal{O}$ thus play a symmetric role, and can be interchanged in the complexity discussions throughout this work. Concepts are often referred to as **elements** of this lattice. A lattice is represented by its Hasse diagram: transitivity and reflexivity edges 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.

Such a lattice has a smallest element, called **bottom**, and a greatest element, called **top**. 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 a **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 the concept lattice of subrelation $R(\mathcal{P}, B)$.

**Example 2.1** *We now introduce the example which we will use throughout this paper, using binary relation R, given by the table below; 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* | × | | | | | | | × |

*The bottom element is* $\emptyset \times 123456$*; the atoms are* $ah \times 236, b \times 123, c \times 125$ *and* $d \times 145$*; bc (or, equivalently,* $bc \times 12$*) is a successor of b,* $(\emptyset, b, bc, abcgh, abcdefgh)$ *is a maximal chain of the lattice, the cover of b is* $\{abgh, bc\}$.

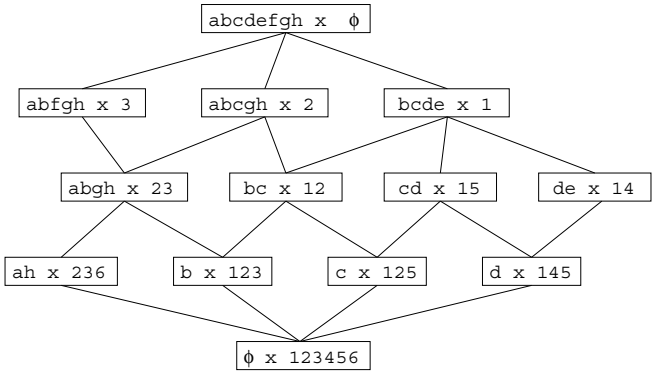


Figure 1: *Concept lattice* $\mathcal{L}(R)$ *of relation R.*

# 3 A general algorithmic process

## 3.1 How the algorithm works

We will begin by giving the general idea of the process we use.

Let us first make the trivial preliminary observation that if the relation has columns containing only ones, the corresponding properties can be discarded as non-informative, as clearly they will be included in every intent of the lattice; in particular, they will constitute the intent of the bottom element.

Once the relation has been purged of its columns of ones, knowing which properties will appear in the atoms is simply a question of inclusion between columns: any property $x$ such that there is some property $y$ with $R[x] \subset R[y]$ will NOT appear in any atom, and conversely any $x$ whose column is not properly included in any other will appear in an atom. Furthermore, if $R[x] = R[y]$, then $x$ and $y$ will ALWAYS appear together in every intent, whether atomic or not. In this case, $x$ and $y$ can be considered as a single property, and will be grouped together inside an equivalence class we call a *maxmod*.

*In our example,* $R[g] \subset R[a], R[g] \subset R[b], R[e] \subset R[d], R[f] \subset R[a], R[f] \subset R[b]$, *and* $R[f] \subset R[g]$. *If we accordingly discard* $e, f$ *and* $g$, *what is left, i.e.* $a, b, c, d$ *and* $h$ *will constitute the atoms; since* $R[a] = R[h]$, *we know that* $a$ *and* $h$ *will always appear together; since there are no other equal columns, the atoms will be* $ah, b, c$ *and* $d$.

The sublattice of which a given concept $A \times B$ is the bottom element is exactly the lattice of the subrelation $R(\mathcal{P}, B)$; since in this subrelation, the elements of $A$ correspond exactly to the columns of ones, the atoms of this lattice are defined by the properties of $R(\mathcal{P} - A, B)$ whose columns are not properly included in any other.

The algorithm from [6] works according to these principles. For each concept which is processed, it either checks all pairs of properties for row inclusion (hence the $O(|\mathcal{P}|^2|\mathcal{O}|)$ complexity), or it pre-processes an intermediate tripartite graph in $O(n^\alpha)$, which is the time required to achieve transitive closure or, equivalently, matrix multiplication; currently, $\alpha$ is approximately 2.376 ([8]). After this, a pair of properties can be checked for row inclusion in $O(1)$ time, and thus the atoms can be found in $O(n^2)$ time by comparing all possible pairs, which is less than the $O(n^\alpha)$ pre-processing cost. The algorithm is presented in a Breadth-First implementation: it repeatedly dequeues the next concept, computes its cover, and enqueues each element of the cover which has not already been generated. Note that this process could just as well be implemented in a Depth-First fashion, by using a stack instead of a queue. The main drawback to Bordat's approach is that it requires a potentially exponential-size data structure to store all the already computed concepts.

We will show later in this section how to dispense with this data structure. In Section 4, we will use the property that row inclusion is inherited by successors:

if in the subrelation corresponding to a concept $A \times B$ we have $R[x] \subseteq R[y]$, then this will remain true in the smaller subrelation defined by any descendant of $A \times B$.

## 3.2 Processing each concept only once

We will need some additional formal definitions of the notions discussed above.

**Definition 3.1** *Let $(\mathcal{P}, \mathcal{O}, R)$ be a context, let $x, y \in \mathcal{P}$. We will say that $x$ **dominates** $y$ if $R[x] \subseteq R[y]$.*

**Definition 3.2** *Given a context $(\mathcal{P}, \mathcal{O}, R)$, a **maxmod** of $R$ is a set $X$ of properties such that $\forall x, y \in X, R[x] = R[y]$ and $\forall z \in \mathcal{P} - X$, $R[z] \neq R[x]$. In this case, $R[X]$ is simply $R[x]$, where $x$ is an arbitrary element of $X$. The **degree** of a maxmod $X$ is $|R[X]|$.*

**Definition 3.3** *Given two maxmods $X$ and $Y$, we will say that $X$ **dominates** $Y$ if $R[X] \subset R[Y]$. A maxmod $X$ is said to be **dominating** if there is some maxmod $Y$ which it dominates, and **non-dominating** if there is no other maxmod which it dominates.*

**Theorem 3.4** *([3]) Given a concept $A \times B$, $A + X$ is the intent of a concept belonging to the cover of $A \times B$ iff $X$ is a non-dominating maxmod of $R(\mathcal{P} - A, B)$. In this case, the extent of $A + X$ is $B \cap R[X]$.*

Our stated goal is to process each concept exactly once, without using a data structure to store all the already-computed concepts. Using a Depth-First approach, if a non-dominating maxmod $X$ is called first, then all the concepts whose intent contain $X$ will be generated; hence if we do not want these concepts to be generated another time, we can give all the other non-dominating maxmods of $R$ the information that no concept containing $X$ should be stacked again. We can use this principle recursively on each sublattice which is processed. The same principle can be used for a Breadth-First approach, essentially by forcing the corresponding spanning tree to be the same as the one described by the recursive approach. Note that the process could be parallelized easily.

We give below the general algorithm for a Breadth-First approach, using a set Marked which prevents from processing a given concept more than once. Note that if a maxmod $Y$ dominates a maxmod $X$, and if all the concepts containing $X$ have been generated, then all the concepts containing $Y$ have also also been generated; with no extra cost, we can add to our Marked set all these $Y$.

**BF-CONCEPTS**

**input** : A context $\mathcal{C} = (\mathcal{P}, \mathcal{O}, R)$.
**output** : The concepts defined by $\mathcal{C}$ are printed.
Initialize QUEUE with bottom element and MARKED as empty;
**repeat**
  0. $\{ A \times B$, MARKED $\}$ ← DEQUEUE;
  1. Compute the partition PART of $\mathcal{P} - \mathcal{A}$ into maxmods;
  2.
  Find the set ND of non-dominating maxmods of subrelation $R(\mathcal{P} - A, B)$;
  Compute the cover of $A \times B$ (*if desirable*);
  3.
  NEW ← ND from which any maxmod containing an element of MARKED has been removed;
  **for** $X$ ***in*** NEW **do**
    PRINT $(A + X) \times (B \cap R[X])$;
    ENQUEUE($\{ (A + X) \times (B \cap R[X])$, MARKED $\}$);
    $Y \leftarrow$ union of all maxmods which dominate $X$;
    MARKED ← MARKED $\cup X \cup Y$;
**until** *Queue is empty*;

The corresponding iterative Depth-First algorithm can be deduced from BF-CONCEPTS by using a stack instead of a queue, and replacing ENQUEUE with PUSH, and DEQUEUE with POP. The Depth-First approach is easily implemented by a recursive algorithm, which we give below.

**DF-CONCEPTS**

**input** : A concept $A \times B$, a subset MARKED of $\mathcal{P}$.
**output** : The not yet encountered descendants of $A \times B$ are printed.
1. Compute the partition PART of $\mathcal{P} - \mathcal{A}$ into maxmods;
2.
Find the set ND of non-dominating maxmods of subrelation $R(\mathcal{P} - A, B)$;
Compute the cover of $A \times B$ (*if desirable*);
3.
NEW ← ND from which any maxmod containing an element of MARKED has been removed;
**for** $X$ ***in*** NEW **do**
  PRINT $(A + X) \times (B \cap R[X])$;
  DF-CONCEPTS( $(A + X) \times (B \cap R[X])$ , MARKED );
  $Y \leftarrow$ union of all maxmods which dominate $X$;
  MARKED ← MARKED $\cup X \cup Y$;

The bottom element which is used to initialize both processes is $(U \times \mathcal{O})$, where $U$ is the set of properties which correspond to a column of ones (i.e. $U = \{x \in \mathcal{P} | R[x] = \mathcal{O}\}$).

Algorithm DF-CONCEPTS is initially called on the bottom element on a MARKED set initialized with $U$ by DF-CONCEPTS( $(U \times \mathcal{O})$ , $U$ ).

Both of the above algorithms generate all the concepts as well as the edges of the Hasse diagram of the lattice, without requiring a data structure to store the already generated concepts. This is an important improvement, as it makes the algorithmic step purely local; for instance, it could be implemented with a parallel approach.

We will remark that the recursive approach requires only polynomial space, while the Breadth-First approach may require exponential space; this is because a concept lattice is of small height (less than $|\mathcal{P}|$), but may be of exponential width (the lattice defined by the power set is an example of this); using a Breadth-First approach requires storing the entire width into the queue at some time. By contrast, the recursive stack will only contain at a given time at most a set of elements which belong to a common maximal chain, as well as the cover of each of these elements, which adds up to at most $|\mathcal{P}|^2$ entries, each requiring $O(|\mathcal{P}|)$ space.

Both algorithms can easily be implemented in $O(|\mathcal{P}|^2|\mathcal{O}|)$ per concept by pairwise comparisons of the properties for row-inclusion in order to find the partition into maxmods and the set ND of non-dominating maxmods (Steps 1 and 2).

We will now examine how we can achieve these Steps 1 and 2 more efficiently.

### 3.3 Using partition refinement to compute the partition into maxmods

The first problem we address is that of efficiently computing the partition into maxmods. In order to avoid spending $O(|\mathcal{P}|^3)$ time on this, we use a technique of partition refinement which repeatedly uses the ones of a line to split the current partition of the property set. This process was originally implicitly used in the famous graph algorithm known as LexBFS, introduced by Rose, Tarjan and Lueker ([26]) to recognize chordal graphs efficiently; Hsu and Ma in [14] later explicitly used partition refinement to find the partition into maximal clique modules of a chordal graph.

**Algorithm MAXMOD-PARTITION**

**input** : A context $(\mathcal{P}, \mathcal{O}, R)$.

**output** : An ordered partition PART=$(K_1, ..., K_p)$ of $\mathcal{P}$ into the maxmods of $R$.

PART$\leftarrow$ $(\mathcal{P})$;
**for** $y \in \mathcal{O}$ **do**
  **foreach** *class* $K$ *of* PART *such that* $|K| > 1$ **do**
    $K' \leftarrow K \cap R[y]$;
    $K'' \leftarrow K - R[y]$;
    **if** $K' \neq \emptyset$ *and* $K'' \neq \emptyset$ **then**
      In PART, replace $K$ by $K'$ followed by $K''$;
**Return** PART;

**Example 3.5** *Let us run algoritm MAXMOD-PARTITION on relation of Example 2.1.*
**Initially:** PART=$(\{a, b, c, d, e, f, g, h\})$.
**Step 1:** *choose (for example) object* 1 *first;* $R[1] = \{b, c, d, e\}$.
PART *contains only one class* $K = \{a, b, c, d, e, f, g, h\}$ *which is split into* $K' = \{b, c, d, e\}$ *and* $K'' = \{a, f, g, h\}$. PART *becomes* $(\{b, c, d, e\}, \{a, f, g, h\})$ *denoted* $bcde \mid afgh$.
**Step 2:** *with object* 2; $R[2] = \{a, b, c, g, h\}$. *Class* $\{b, c, d, e\}$ *is split into* $\{b, c\}$ *and* $\{d, e\}$; *Class* $\{a, f, g, h\}$ *is split into* $\{a, g, h\}$ *and* $\{f\}$. PART *becomes:* $bc \mid de \mid agh \mid f$.

*Below is the entire succession of partition refinement steps:*

$abcdefgh$

$\downarrow \quad R[1] = \{b, c, d, e\}$

$bcde \mid afgh$

$\downarrow \quad R[2] = \{a, b, c, g, h\}$

$bc \mid de \mid agh \mid f$

$\downarrow \quad R[3] = \{a, b, f, g, h\}$

$b \mid c \mid de \mid agh \mid f$

$\downarrow \quad R[4] = \{d, e\}$

$b \mid c \mid de \mid agh \mid f$

$\downarrow \quad R[5] = \{c, d\}$

$b \mid c \mid d \mid e \mid agh \mid f$

$\downarrow \quad R[6] = \{a, h\}$

$b \mid c \mid d \mid e \mid ah \mid g \mid f$

**Property 3.6** *Let $(\mathcal{P}, \mathcal{O}, R)$ be a context. Algorithm MAXMOD-PARTITION computes the partition of $\mathcal{P}$ into maxmods.*

**Proof:** Let $(\mathcal{P}, \mathcal{O}, R)$ be a context; let $x_1$ and $x_2$ be two elements of $\mathcal{P}$. We will show that $x_1$ and $x_2$ are in the same maxmod iff they stay in the same class at the end of Algorithm MAXMOD-PARTITION.
At the beginning of the algorithm, $x_1$ and $_2$ are in the same class.

- $\Rightarrow$ If $x_1$ and $x_2$ are in the same maxmod, then $R[x_1] = R[x_2]$. When the algorithm is processing an object $y$, $x_1$ and $x_2$ will remain in the same class, which is $K'$ if $x_1 \in R[y]$, and $K$" if $x_1 \notin R[y]$.Thus no object can separate $x_1$ and $x_2$ and they stay in the same class at the end of the algorithm.
- $\Leftarrow$ If $x_1$ and $x_2$ are *not* in the same maxmod, then $R[x_1] \neq R[x_2]$. W.l.o.g. let $y$ be an object such that $y \in R[x_1] - R[x_2]$. When $y$ is processed, if $x_1$ and $x_2$ are still in the same class $K$, they will be separated as $x_1$ will go into $K'$ and $x_2$ will go into $K$".

□

It is easy to see that Algorithm MAXMOD-PARTITION can be implemented in $O(min(|R|, |\overline{R}|))$.

## 3.4 Computing the non-dominating maxmods

Once we have computed the partition PART into maxmods, we need to compute the set ND of *non-dominating* maxmods. We will use an ordering on the maxmods: we either order the maxmods by decreasing degree, or simply use the ordered partition output by Algorithm MAXMOD-PARTITION.

In both cases, a given maxmod can dominate only maxmods which lie to its left, with the effect that the leftmost maxmod is guaranteed to be non-dominating. This property is trivial for the ordering by decreasing degree, and too complicated to go into in this paper for ordering induced by Algorithm MAXMOD-PARTITION.

We thus present the following process for computing the set of non-dominating maxmods:

**Algorithm NON-DOMINATING-MAXMODS**

**input** : A context $(\mathcal{P}, \mathcal{O}, R)$, and a partition PART=$(K_1, ..., K_p)$ of $\mathcal{P}$ into maxmods of $R$, ordered either by Algorithm MAXMOD-PARTITION or by decreasing degree of the maxmods.

**output** : The set ND of non-dominating maxmods.

ND $\leftarrow \emptyset$;
**repeat**
1. $X \leftarrow$ leftmost maxmod in PART; Remove $X$ from PART;
2. ND $\leftarrow$ ND $+ \{X\}$ ;
3. Compute the maxmods which dominate $X$ and remove them from PART;

**until** PART *is empty*;

There are several ways of doing Step 3 of Algorithm NON-DOMINATING-MAXMODS (which computes the maxmods which dominate $X$).

1. We can use Bordat's original approach: Step 3 will begin with an $O(|(\mathcal{P} - A)+B|^\alpha)$ pre-processing step in the subrelation $R(\mathcal{P}-A, B)$ corresponding to the concept $A \times B$ which is being processed. After this, finding the maxmods which dominate maxmod $X$ can be done in $O(|\mathcal{P}|)$ time. Thus each non-dominating maxmod can be found in $O(|\mathcal{P}|)$ time, and the entire set in $O(|\mathcal{P}|^2)$ time, which is less than the cost of the pre-processing.

2. We can dispense with the pre-processing: each maxmod is selected in $O(1)$ as leftmost in the remainder of the partition, and computing the maxmods which dominate current maxmod $X$ requires scanning the relation in $O(|R|)$. This adds up to $O(|R|)$ time per non-dominating maxmod which is generated.

To process a concept $A \times B$, Algorithm NON-DOMINATING-MAXMODS either costs $O(|(\mathcal{P} - A) + B|^\alpha)$, if Bordat's pre-processing is used, as discussed above, or $O(|R(\mathcal{P} - A, B)|\Delta)$, where $\Delta$ is the number of elements of the cover of the concept in the lattice ($\Delta$ is in $O(|\mathcal{P}|)$); in this case, each concept is generated as many times as it has predecessors, costing $O(|R|)$ each time. The second version may be better if the number of predecessors of the concepts is small.

**Example 3.7** *Let us run Algorithm NON-DOMINATING-MAXMODS on partition* PART=$(b, c, d, e, ah, g, f)$ *obtained in Example 2.1.*
ND=$\emptyset$.
**Step 1***: $X \leftarrow \{b\}$ is a non-dominating maxmod.* ND $\leftarrow \{\{b\}\}$. $\{b\}$ *is dominated by* $\{f\}$ *and* $\{g\}$*, as* $R[b] = \{1, 2, 3\}$*,* $R[f] = \{3\} \subset R[b]$ *and* $R[g] = \{2, 3\} \subset R[b]$. PART $\leftarrow (c, d, e, ah)$.
**Step 2***: $X \leftarrow \{c\}$ is a non-dominating maxmod.* ND $\leftarrow \{\{b\}, \{c\}\}$. $\{c\}$ *is neither dominating nor dominated.* PART $\leftarrow (d, e, ah)$.

**Step 3**: $X \leftarrow \{d\}$ *is a non-dominating maxmod.* ND $\leftarrow \{\{b\},\{c\},\{d\}\}$. $\{d\}$ *is dominated by* $\{e\}$ *as* $R[e] = \{1,4\} \subset R[d] = \{1,4,5\}$. PART $\leftarrow (ah)$.
**Step 4**: $X \leftarrow \{a,h\}$ *is a non-dominating maxmod.* ND $\leftarrow \{\{b\},\{c\},\{d\},\{a,h\}\}$. PART $\leftarrow \emptyset$ *and the algorithm stops.* ND=$\{\{b\},\{c\},\{d\},\{a,h\}\}$ *will be outputted as the set of non-dominating maxmods.*

Our preliminary experimentations suggest that the second version runs more rapidly in most cases; it runs even faster if, after computing the partition into maxmods, instead of computing the set of non-dominating maxmods, one first removes all maxmods which contain a property belonging to the set Marked, and then tests the leftmost *remaining* maxmod to check whether it is non-dominating (it may fail to be so because of the removed maxmods); in practise, it is non-dominating almost every time, so we run in linear time per generated concept.

We will see in the next section how we can improve the worst-time complexity.

According to Theorem 3.4, we can generate, given the set ND of non-dominating maxmods, the cover of an element, as well as the corresponding edges of the lattice. This process can obviously be removed from Step 2 of algorithms BF-CONCEPTS and DF-CONCEPTS, if the purpose is only to reach each concept once.

**COVER**

**input** : A concept $A \times B$, set ND of non-dominating maxmods of $R(\mathcal{P} - A, B)$.
**output** : The cover of $A \times B$ is printed.
**for** $X$ ***in*** ND **do**
  PRINT $(A + X) \times (B \cap R[X])$;
    // *which belongs to the cover of* $A \times B$
  PRINT $(A \times B, (A + X) \times (B \cap R[X]))$;
    // *which is an edge of the Hasse diagram*

# 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.

## 4.1 Data structure: the domination table

In order to improve the worst-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 trivially, domination is inherited when moving up in the lattice:

**Property 4.1** *Let $A_1 \times B_1$ and $A_2 \times B_2$ be concepts, with $A_1 \subset A_2$ (i.e. $A_2 \times B_2$ is a descendant of $A_1 \times B_1$). Then if $x$ dominates $y$ in $R(\mathcal{P} - A_1, B_1)$ and $x, y \in \mathcal{P} - A_2$, then $x$ dominates $y$ in $R(\mathcal{P} - A_2, B_2)$.*

To efficiently answer requests on the set of non-dominating maxmods, we use a domination table containing information on the current subrelation. 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 $A \times B$ corresponding to the subrelation $R(\mathcal{P} - A, B)$ to a second concept $(A + X) \times (B \cap R[X])$, covering the first, and corresponding to the subrelation $R(\mathcal{P} - (A + X), B \cap R[X])$, two things happen:

1. Set $X$ of properties disappear from the relation.
2. Set $B - R[X]$ of objects disappear from the relation.

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

Our idea, previously used to maintain Galois sub-hierarchies in [4], is to list into a table $L$, for each pair of properties $(x, y)$, the objects which **prevent** $x$ from dominating $y$: an object $i$ will appear in the list $L[x, y]$ iff $(x, i) \in R$ and $(y, i) \notin R$.

**Example 4.2** *The corresponding lists for our example are given in table L below:*

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

*$c$ will dominate $a$ when objects 1 and 5 have disappeared as $L[c, a] = \{1, 5\}$.*

The table can be queried as follows:

- A property $x$ dominates another property $y$ iff $L[x, y] = \emptyset$.
- $x$ and $y$ are in the same maxmod iff $L[x, y] = L[y, x] = \emptyset$.

Table $L$ contains at most $|\mathcal{P}|.min(|R|, |\overline{R}|)$ bits, since a given column $x$ will contain in its lists only objects which with $x$ define ones of $R$, while row $x$ will contain in its lists only objects which with $x$ define zeroes of $R$.

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 relation. Actually, we are only concerned with the **number** of properties which a property $x$ dominates in a given relation, 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 properties 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 $L[x, y]$. Property $x$ dominates property $y$ 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 properties $y$ such that $T[x, y] = 0$, i.e. the number of properties 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 $D$.

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

**Initializing tables $T$ and $D$**

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

*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 |

## 4.2 Algorithmic use of the domination table

We will now modify Algorithm DF-CONCEPTS, using data structures $T$ and $D$ described in the previous subsection, and used as global variables in order to save space; this requires adding UPDATE primitives.

**Algorithm INHERIT-CONCEPTS**

**input** : Concept $A \times B$, a subset MARKED of $\mathcal{P}$.

**output** : The not yet encountered descendants of $A \times B$.

1. Compute the partition PART of $\mathcal{P} - \mathcal{A}$ into maxmods ;

2. // *Find the set* ND *of non-dominating maxmods of* $R(\mathcal{P} - A, B)$.

**for** $X \in$PART **do**

    Choose $x$ in $X$;

    **if** $D[x] = |X|$ **then**

        ND$\leftarrow$ND + $X$;

Compute the cover of $A \times B$ (*if desirable*);

3.

NEW $\leftarrow$ ND from which any maxmod containing an element of MARKED has been removed;

**for** $X$ ***in*** NEW **do**

    $A' \leftarrow (A + X)$; $B' \leftarrow (B \cap R[X])$;

    PRINT $A' \times B'$;

    PREUPDATE($A$ , $X$);

    INHERIT-CONCEPTS( $A' \times B'$, MARKED );

    POSTUPDATE($A$ , $X$);

    $Y \leftarrow$ union of all maxmods which dominate $X$;

    MARKED $\leftarrow$ MARKED $\cup X \cup Y$;

**Procedure PREUPDATE**

**input** : Intent $A$ of concept $A \times B$, a non dominating maxmod $X$ of $R(\mathcal{P} - A, B)$.

**output** : Tables $T$ and $D$ are modified using $X$ and $A$.

Choose $x$ **in** $X$;
**for** $y$ ***in*** $(\mathcal{P} - A) - X$ **do**
  **if** $T[y, x] = 0$ **then**
    $D[y] \leftarrow D[y] - |X|$;

**for** $j$ ***in*** $B - R[x]$ **do**
  $Z \leftarrow (\mathcal{P} - A) - R[j] - X$;
  $U \leftarrow (\mathcal{P} - A) - Z - X$;
  **for** $(u, z)$ ***in*** $U \times Z$ **do**
    $T[u, z] \leftarrow T[u, z] - 1$;
    **if** $T[u, z] = 0$ **then**
      $D[u] \leftarrow D[u] + 1$;

**POSTUPDATE**

**input** : Intent $A$ of concept $A \times B$, a non dominating maxmod $X$ of $R(\mathcal{P} - A, B)$.

**output** : Tables $T$ and $D$ are modified using $X$ and $A$.

Choose $x$ **in** $X$;
**for** $y$ ***in*** $(\mathcal{P} - A) - X$ **do**
  **if** $T[y, x] = 0$ **then**
    $D[y] \leftarrow D[y] + |X|$;

**for** $j$ ***in*** $B - R[x]$ **do**
  $Z \leftarrow (\mathcal{P} - A) - R[j] - X$;
  $U \leftarrow (\mathcal{P} - A) - Z - X$;
  **for** $(u, z)$ ***in*** $U \times Z$ **do**
    $T[u, z] \leftarrow T[u, z] + 1$;
    **if** $T[u, z] = 1$ **then**
      $D[u] \leftarrow D[u] - 1$;

The algorithm is initially called on the bottom element $(U \times \mathcal{O})$ by INHERIT-CONCEPTS( $(U \times \mathcal{O})$ , $U$ ) on a MARKED set initialized with $U$, where $U$ corresponds to the columns of ones. Tables $T$ and $D$ are initialized from $R$ as described in the previous subsection.

### 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. In this analysis, we consider that the size of a relation includes its property set and object set (for example, $|R| = |\{(x, y) \in R\}| + |\mathcal{P} + \mathcal{O}|$).

- Each step of Algorithm INHERIT-CONCEPTS requires computing the maxmods of subrelation $R(\mathcal{P} - A, B)$, which can be done with Algorithm MAXMOD-PARTITION in $O(\, min(R(\mathcal{P} - A, B), \overline{R}(\mathcal{P} - A, B)\,)\,)$. Using table $D$, finding the set of non-dominating maxmods requires $O(|\mathcal{P} - A|)$ time. Comparing these with MARKED costs $O(|\mathcal{P} - A|)$, thus a concept is processed in global $O(\, min(R(\mathcal{P} - A, B), \overline{R}(\mathcal{P} - A, B))\,)$ time.

- Tables $T$ and $D$ are pre-updated at each step to describe the domination relationships in the new subrelation 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 $A' \times B' = (A + X) \times (B \cap R[X])$, obtained from non-dominating maxmod $X$ of $R(\mathcal{P} - A, B)$.

  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. such that $i \notin B'$.

  An object $i$ will appear in the list $L[x, y]$ iff $(x, i)$ is in the relation and $(y, i)$ is not, which can be translated as: $(x, i) \in R(\mathcal{P} - A, B)$ and $(y, i) \notin R(\mathcal{P} - A, B)$.

  Since we are generating subrelation $R(\mathcal{P} - A', B')$, we do not need the elements of $A'$; thus, the effort required is: $|\mathcal{P} - A'|\,.\,\Sigma_{i \in (B - B')} |\mathcal{P} - R[i]|$.

  However, there is an amortized complexity when moving up along a branch of the spanning tree of the lattice induced by the algorithm: since table $L$ contains at most $|\mathcal{P}|.min(|R|, |\overline{R}|)$ bits, in the worst case, when moving all the way up from the bottom element to the top element of the lattice, the table would be completely emptied, which would cost at most $O(|\mathcal{P}|\, min(|R|, |\overline{R}|))$ time.

The global time complexity of the algorithm is thus bounded by $O(min(R, \overline{R}))$ per generated concept, plus an updating cost of $O(|\mathcal{P}|\, min(|R|, |\overline{R}|))$ 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 stack contains at most $O(|\mathcal{P}|)$ concepts of size $O(|\mathcal{P} + \mathcal{O}|)$ each, MARKED is of size $O(|\mathcal{P}|)$; $T$ contains $O(|\mathcal{P}|\, min(|R|, |\overline{R}|))$ bits.

The global space complexity is thus in $O(|\mathcal{P}|\, min(|R|, |\overline{R}|))$.

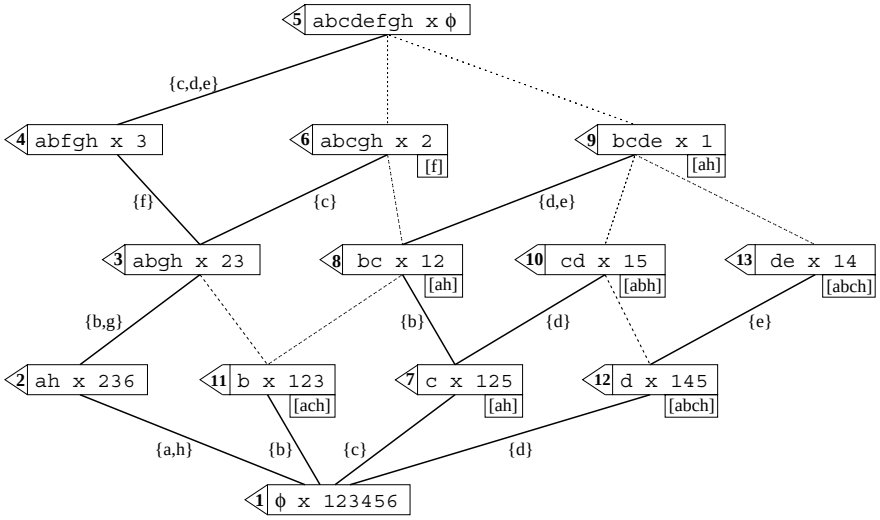


Figure 2: *Concept lattice $\mathcal{L}(R)$ of relation $R$. The concepts are numbered in prefix order following our sample recursive execution described in Section 4; the inherited sets of already processed vertices appear between brackets. The edges of the Depth-First tree are labeled by the non-dominating maxmod used to compute each new concept.*

**Example 4.3** *Let us execute Algorithm INHERIT-CONCEPTS on relation $R$ of Example 2.1, associated with the concept lattice of Figure 2 which is labeled according to the execution.*

**Step 1**: The execution starts with the bottom element $\emptyset \times 123456$. In $R = R(\mathcal{P}, \mathcal{O})$, 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 properties is empty. $ah \times 236$ is chosen to be processed next. $\mathcal{P} - \{a, h\} = \{b, c, d, e, f, g\}$, the future subrelation will be $R(\{b, c, d, e, f, g\}, \{2, 3, 6\})$. The table is accordingly pre-updated: since objects 1, 4 and 5 disappear from $R$, 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 $R[1] = \{f, g\}$, $R[4] = \{b, c, f, g\}$ and $R[5] = \{b, e, f, g\}$.
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)

**Step 2**: Concept $ah \times 236$ is processed. Maxmods of $R'$: $\{b, g\}$, $\{c\}$, $\{d, e\}$, $\{f\}$; 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 subrelation 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 stack is empty and the algorithm terminates.

# 5 Conclusion

In this paper, we explain how to use Bordat's approach to concept generation without requiring a data strucure of exponential size.

Our complexity analysis involves traversed maximal chains of the lattice; we believe that could be simplified and streamlined or even amortized. Moreover, when the lattice is very large, the relation tends to have many ones, thus $|\overline{R}|$ is of order $n$. In this case, we also run as fast as the algorithms which require expo-

nential space, and, in practise, probably even faster, as we are not handicapped by huge data structures.

Implementations remained to be pursued; comparisons of various modern algorithms such as those done by [18] could integrate our new approaches. It is also interesting to investigate on which classes of relations (sparse or dense for example) our various versions would work better, especially using 'real' data as in [9].

## 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. We also thank the referees for their interesting remarks.

## References

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

[2] Berry A., Bordat J-P., Cogis O.: Generating all the minimal separators of a graph. *International Journal of Foundations of Computer Science*, **11** (2000) 397–404.

[3] Berry A., Sigayret A.: Representing a concept lattice by a graph. *Proceedings of DM&DM'02 (Discrete Maths and Data Mining Workshop), 2nd SIAM Conference on Data Mining* (Arlington, VA, April 2002). *Discrete Applied Mathematics, special issue on Discrete Maths and Data Mining*, **144**:1-2 (2004) 27–42.

[4] Berry A., and Sigayret A.: Maintaining class membership information. *Workshop MASPEGHI (MAnaging of SPEcialization/Generalization HIerarchies), LNCS proceedings of OOIS'02 (Object-Oriented Information Systems)*, (Montpellier, Fr, Sept. 2002).

[5] Birkhoff G.: *Lattice Theory.* American Mathematical Society, 3rd Edition, (1967).

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

[7] Chein M.: Algorithme de recherche de sous-matrices premières d'une matrice. *Bull. Math. R.S. Rumania*, **13** (1969).

[8] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, **9**:3 (1990) 251–280.

[9] Fu H., N. Mephu Nguifo E.: Partitioning large data to scale-up lattice-based algorithm. *IEEE Press proceedings of ICTAI'03 (15th IEEE International Conference on Tools with Artificial Intelligence)*, (Sacramento, CA, USA, Nov. 2003) 537–541.

[10] Fu H., N. Mephu Nguifo E.: How well go Lattice Algorithms on currently used Machine Learning TestBeds. *Proc. Conference EGC'04*, (Clermont-Ferrand, Fr, 2004) 373–384.

[11] Ganter B.: Two basic algorithms in concept analysis. *Preprint 831, Technische Hochschule Darmstadt*, (1984).

[12] Ganter B., Wille R.: *Formal Concept Analysis*. Springer, (1999).

[13] Guénoche A.: Construction du treillis de Galois d'une relation binaire. *Mathématiques, Informatique et Sciences Humaines*, **121** (1993) 23–34.

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

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

[16] Huchard M., Roume C., Valtchev P.: When concepts point at other concepts: the case of UML diagram reconstruction. *Proceedings of FCAKDD'02 (Formal Concept Analysis for Knowledge Discovery in Databases), Int. Conf. ECAI'02*, (Lyon, Fr, Juillet 2002) 32–43.

[17] Kloks T., Kratsch D.: Listing all minimal separators of a graph. *SIAM Journal on Computing*, **27** (1998) 605–613.

[18] Kuznetsov S.O., Obiedkov S. A.: Comparing performance of algorithms for generating concept lattices. *Journal for Experimental and Theoretical Artificial Intelligence (JETAI)*, **14**:2-3 (2002) 189–216.

[19] Kuznetsov S. O.: Algorithm for the construction of the set of all concepts and their line diagram. *Preprint MATH-AI-05, TU-Dresden*, (2000).

[20] Liquière M., Sallantin J.: Structural machine learning with Galois lattices and Graphs. *Proceedings of ICML'98 (International Conference on Machine Learning)*, Morgan Kaufmann Ed., (1998) 305–313.

[21] Malgrange Y.: Recherche des sous-matrices premières d'une matrice à coefficients binaires. *2nd congrès de l'AFCALTI*, Gauthier-Villars, (Oct. 1961).

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

[23] Norris E. M.: An algorithm for computing the maximal rectangles of a binary relation. *Rumanian Revue of Pure and Applied Mathematics*, **23**:2 (1978) 243–250.

[24] Nourine L., Raynaud O. A Fast Algorithm for building Lattices. *International Processing Letters*, **71** (1999) 199–204.

[25] Bastide Y., Lakhal L., Pasquier N., Taouil R.: Efficient mining of association rules using closed itemset lattices. *Journal of Information Systems*, **24**:1 (1999) 25–46.

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

[27] Shen H., Li K., Zheng S. Q.: Separators are as simple as cutsets. *Proc. ASIAN'99 (5th Asian Computer Science Conference)*, (Phuket, Thailand, December 1999). LNCS **1742** (1999) 347–358.

[28] Sheng H., Liang W.: Efficient enumeration of all minimal separators in a graph. *Theoretical Computer Science*, **180** (1997) 169–180.

[29] Sigayret A.: *Data mining: une approche par les graphes.* PhD thesis, Université Blaise Pascal (Clermont-Ferrand, Fr), DU 1405 – EDSPIC 269 (2002).

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

[31] Valtchev P., Missaoui R., LebrunP.: A partition-based approach towards building Galois (concept) lattices. *Discrete Mathematics*, **256**:3 (2002) 801–825.