Efficient Concept Generation

Anne Berry, Jean-Paul Bordat, Alain Sigayret

Efficient Concept Generation

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 algorithm which generates each concept exactly once, using graph-theoretic results. This process has a time complexity of $O(|\mathcal{P}|m)$ per maximal chain of the concept lattice, where m denotes the number of non-crosses of the relation, and uses a data structure which is of small polynomial size. This improves the current best-time algorithms for this problem, which require either $O(2^{|\mathcal{P}|})$ space and $O(|\mathcal{P}|^2)$ time per concept, or, alternately, polynomial space but $O(|\mathcal{P}|^3)$ time per concept.

Our algorithm can be used, with no extra cost, to compute the edges of the lattice, and can, just as efficiently, 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 concept lattice. This theory, though studied by mathematicians as far back as the Nineteenth Century (see [1], [10]), was made popular and developed by Wille and his team ([10]), 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 ([24]), the generation of frequent item sets ([23]), machine learning ([15], [17]) and the reorganization of object hierarchies ([13], [5]).

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, even when only part of the lattice is explored.

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 Chein ([8]); 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 (see [11]).

Bordat's algorithm ([7]) 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 ([18]) to $O(n^2)$ per concept.

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

*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.

When the concepts do not need to be stored, but only encountered at least once, the space problem becomes easier, though the running time per concept is higher: the best such algorithm, due to Ganter ([9]), 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 an algorithm which runs in $O(nm)$ time per maximal chain, where m denotes the number of non-crosses of the relation, and requires only small polynomial space ($O(nm)$). This greatly improves [18], as we have a comparable complexity without requiring 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 maximal chain.

The algorithm we introduce here presents similarities with [7]. 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 the sub-relation $R((\mathcal{P} - A), B)$.

One of the new features which is added 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, as 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 [7], Bordat used a bipartite graph to handle the relation. In [4], Berry and Sigayret proposed a different encoding into a 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.

This is algorithmically interesting because, in the past decade, much research has been done on using minimal separators to efficiently solve various graph problems such as chordal embedding ([19], [2]), and in particular several papers deal with the efficient enumeration of minimal separators ([14], [21], [20], [3]).

[4] 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 [4] 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 [4]: if all the vertices of a set A share the same neighborhood, and are not together dominated by 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.

Before presenting our algorithm, we will give some formal definitions and our general algorithmic process. We will also present our data structure, the domination table, 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 $\mathcal{P} \times \mathcal{O}$. Given an element of $\mathcal{P} \times \mathcal{O}$, we will refer to it as a **cross** of the relation if it is in R, as a **non-cross** if it is not. We will use n to denote $|\mathcal{P}|$, and m to denote the number of non-crosses.

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 concepts, ordered by inclusion on the intents, define a lattice, called a **concept lattice** or Galois lattice. A lattice is represented by its Hasse diagram: transitivity and reflexivity arcs are omitted. Concepts are often referred to as **elements** of this lattice. Such a lattice, has a smallest element, called the **bottom element**, and a greatest element, called the **top element**. A path from bottom to top is called a **maximal chain** of the lattice. If the relation R has a full line of crosses, it is easy to compute the concepts of R from the concepts of the sub-relation from which this line has been removed ([4]). In the rest of this paper, we will discuss only relations with no such lines of crosses.

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.

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

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

2.2 Graphs

Given a relation R, [4] 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 $X \subset \mathcal{P}, Y \subset \mathcal{O}$, we will denote by $G_R(X \cup Y)$ the subgraph of G_R induced by vertex set $X \cup Y$.

We will use only external neighborhoods, which we will 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; 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 $N^+(Y) \subset N^+(X)$.

Theorem 2.4 *([4]) A concept $(A+X) \times B'$ covers a concept $A \times B$ iff X, in $G_R((\mathcal{P} - A) \cup B)$, 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 $\{d\}$.

3 Algorithmic Process

Theorem 2.4 can be used to recursively compute the cover of an element, starting with the bottom element. This process will generate each concept exactly as many times as the number of predecessors it has.

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 algorithm works up in a depth-first fashion, when a maxmod X is used to generate a concept $(A+X) \times B$, then **all** concepts containing X in their intent will be defined by the recursive call upon $(A+X) \times B$. If a brother concept of $(A+X) \times B$ uses a maxmod containing some vertex x of X, this concept has already been generated previously. If we are careful to store information on the maxmods which have already been used by a brother concept or by a brother of an ancestor concept, we can avoid computing the same concept more than once.

General algorithmic process on concept $A \times B$:

> **Compute set** ND **of non-dominating maxmods of** $G_R((\mathcal{P} - A) \cup B)$;
> // *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.**

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

The most straightforward way to compute the non-dominating maxmods is to repeatedly find a maxmod X of minimum degree, compute the maxmods which dominate it, remove these and X from the vertex set, before reiterating. Algorithm MCS ([22]) would yield a compatible ordering of the maxmods in linear $O(m)$ time, and computing the maxmods dominating X would cost the same time; globally, with this process, we would obtain the set of non-dominating maxmods in $O(m)$ time per non-dominating maxmod generated, which would add up to a worst-time complexity of $O(nm)$ per concept. 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.

However, in order to improve the worse-time behaviour, we propose a more sophisticated approach to computing the non-dominated maxmods, as described below.

Updating the domination information

In order 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 $G_R((\mathcal{P} - A) \cup B)$ to a second concept $(A + X) \times B'$, covering the first, and represented by the underlying graph $G_R((\mathcal{P} - (A + X) \cup (B - N^+(X))$, two things happen:

1. Set X of properties disappear.
2. Set $N^+(X)$ of objects disappear.

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) is a non-cross of R, then (x, i) is also a non-cross of R. Our idea, used to maintain Galois sub-hierarchies in [5], is to list into a table L, for each pair of properties (x, y), the objects which **prevent** x from dominating y. This means that if for object i, $(x, i) \in R$ and $(y, i) \notin R$, i will appear in the list $L[x, y]$.

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

	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.*

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.

In our example, $L[c, a] = \{1, 5\}$; $(a, 1)$ and $(a, 5)$ are edges of G_R.

Updating the 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 when moving up from a concept to one of its successors.

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 $|X|$. The domination table T we use thus contains numbers between 0 and $|\mathcal{O}|$, $T[x, y]$ representing the size of list $L[x, y]$, thus 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 $x \in X$, $D[x] = |M(x)|$, 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:

	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

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

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

For each x **in** $\mathcal{P}$
  For each y **in** $\mathcal{P}$
    For each z **in** $\mathcal{O}$
      If $(x, z) \in R$ and $(y, z) \notin R$ **then** add 1 to $T[x, y]$;

These tables 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 its original form. The global time required for pre-updating T along a maximal chain of the lattice does not exceed the number of bits contained in L, so the pre-updating will cost $O(nm)$. Clearly, the post-updating costs exactly the same. The updating algorithm is given in the next section.

4 The algorithm

The algorithm is initially called by CONCEPTS$(\emptyset \times \mathcal{O}, \emptyset)$ on an empty MARKED set.
Tables T and D are constructed from G_R as described in the previous section.

Algorithm CONCEPTS
Input: A concept $A \times B$, a set MARKED of vertices of $\mathcal{P}$
Output: The not yet encountered direct successors of $A \times B$.

Begin
Initialization:
$G \leftarrow G_R((\mathcal{P} - A) \cup B)$;
Compute 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 $A \times B$.*
For X **in** ND **do**
$A' \leftarrow A + X$; $B' \leftarrow \mathcal{O} - N^+(X)$;
PRINT$(A' \times B')$;
//3. *Generate the unprocessed descendants of $A \times B$.*
For X **in** ND $-$ MARKED **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(*pre*);
CONCEPTS($A' \times B'$, MARKED);
UPDATE(*post*);
MARKED $\leftarrow$ MARKED $\cup X$;
End.

Algorithm UPDATE
Input: A variable V set to *pre* or to *post*.
Output: Tables T and D are modified using current values of X and $A \times B$.
Begin
Choose a representative x **in** X;
//1. *Update table D by simulating deletion of property set X.*
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 by simulating deletion of objects in $N^+(x)$.*
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** $D[u] \leftarrow D[u] - 1$;
End.

Complexity Analysis

Each step of Algorithm CONCEPTS requires computing the maxmods of the graph, which can be done in $O(m)$ time using the algorithm of Hsu and Ma from [12]; using table D, finding the set of non-dominating maxmods requires $O(n)$ time; comparing these with MARKED costs $O(n)$ time, thus a concept is processed in global $O(m)$ time. As discussed in Section 3, Algorithm UPDATE globally costs $O(nm)$ per maximal chain. The global time complexity is thus in $O(m)$ per concept plus $O(nm)$ per maximal chain.

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 4.1

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. New tables T and D obtained:

	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

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

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.

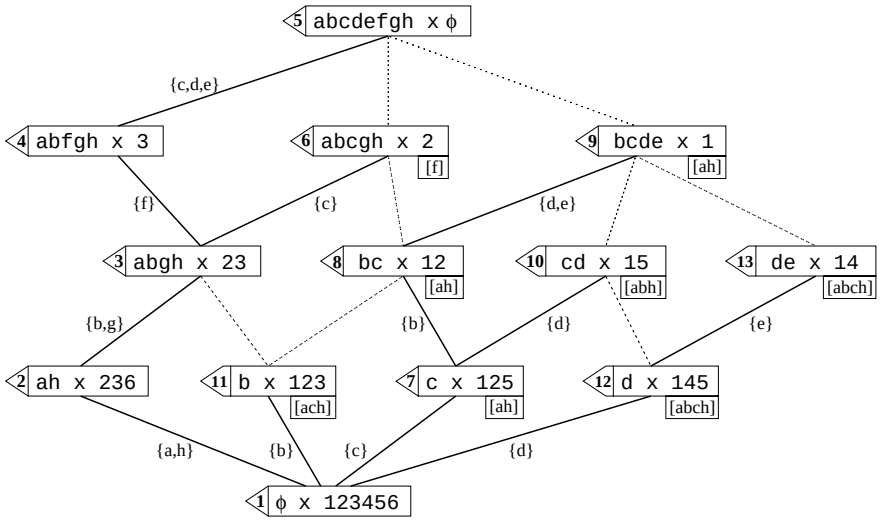

Figure 1: *Concept lattice $\mathcal{L}(R)$ of relation R; the concepts are numbered in prefix order following our sample recursive execution; the inherited sets of already processed vertices appear between brackets. The edges of the Depth-First tree are labeled by the non-dominating maxmod used to compute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

5 Conclusion

In this paper, we propose a new algorithmic approach to concept generation, which enables us to improve all existing algorithms for this problem. Our complexity analysis could probably be streamlined, as $O(nm)$ per maximal chain is a very rough overestimation for the cost of the updating process.

Another promising approach to improve the complexity for this problem would be to use the property that there is a one-to-one correspondence between the maximal chains of the lattice and the minimal triangulations of the underlying co-bipartite graph ([4]) and apply Meister's recent work (see [16]) on the linear-time triangulations of AT-free and Claw-free graphs, a superclass of co-bipartite graphs.

References

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

[2] A. Berry. A Wide-Range Efficient Algorithm for Minimal Triangulation. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99), Baltimore*, pp. 860–861, Jan. 1999.

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

[4] 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.

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

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

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

[8] M. Chein. Algorithme de recherche de sous-matrices premières d'une matrice. *Bull. Math. R.S. Roumanie*, 13, 1969.

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

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

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

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

[13] 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.

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

[15] 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.

[16] D. Meister. Minimal triangulations for some classes of AT-free graphs. *Satellite workshop of WG'02, Prague, 2002.*

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

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

[19] A. Parra and P. Scheffler. How to use the minimal separators of a graph for its chordal triangulation. *Proceedings of the 22nd International Colloquium on Automata, Languages and Programming (ICALP '95), Lecture Notes in Computer Science*, 944:123–134, 1995.

[20] H. Shen. Separators are as simple as cutsets. *Asian Computer Science Conference, Purket, Thailand, December 10-13, 1999, Lecture Notes in Computer Science*, 172:347–358.

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

[22] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs [...]. *SIAM Journal of Computing*, 13:566–579, 1984.

[23] 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.

[24] M. J. Zaki, S. Parthasarathy, M. Ogihara and W. Li. New Algorithms for Fast Discovery of Association Rules. *Proceedings of 3rd International Conference on Database Systems for Advanced Applications*, April 1997.