On the enumeration of pseudo-intents : choosing the order and extending to partial implications

Alexandre Bazin

UNIVERSITÉ PIERRE ET MARIE CURIE

Ecole doctorale ED130

LIP6, ACASA

ON THE ENUMERATION OF PSEUDO-INTENTS : CHOOSING THE ORDER AND EXTENDING TO PARTIAL IMPLICATIONS

Par Alexandre Bazin

Thèse de doctorat d'informatique

Dirigée par Jean-Gabriel Ganascia

Présentée et soutenue publiquement le 30 septembre 2014

Devant un jury composé de :

Madame Karell Bertet, Maitre de conférences, Université de la Rochelle, *Rapporteur*
Monsieur Jean-Gabriel Ganascia, Professeur, UPMC, *Directeur de thèse*
Monsieur Alain Gély, Maitre de conférences, Université de Lorraine, *Examinateur*
Monsieur Henry Soldano, Maitre de conférences, Université Paris-Nord, *Rapporteur*
Madame Annick Valibouze, Professeur, UPMC, *Examinatrice*

Abstract

This thesis deals with the problem of the computation of implications, which are regularities of the form "when there is A there is B", in datasets composed of objects described by attributes. Computing all the implications can be viewed as the enumeration of sets of attributes called pseudo-intents. It is known that pseudo-intents cannot be enumerated with a polynomial delay in the lectic order but no such result exists for other orders. While some current algorithms do not enumerate in the lectic order, none of them have a polynomial delay. The lack of knowledge on other orders leaves the possibility for a polynomial-delay algorithm to exist and finding it would be an important and useful step. Unfortunately, current algorithms do not allow us to choose the order so studying its influence on the complexity of the enumeration is harder than necessary. We thus take a first step towards a better understanding of the role of the order in the enumeration of pseudo-intents by providing an algorithm that can enumerate pseudo-intents in any order that respects the inclusion relation.

In the first part, we explain and study the properties of our algorithm. As with all enumeration algorithms, the first problem is to construct all the sets only once. We propose to use a spanning tree, itself based on the lectic order, to avoid multiple constructions of a same set. The use of this spanning tree instead of the classic lectic order increases the space complexity but offers much more flexibility in the enumeration order. We show that, compared to the well-known Next Closure algorithm, ours performs less logical closures on sparse contexts and more once the average number of attributes per object exceeds 30%. The space complexity of the algorithm is also empirically studied and we show that different orders behave differently with the lectic order being the most efficient. We postulate that the efficiency of an order is function of its distance to the order used in the canonicity test.

In the second part, we take an interest in the computation of implications in a more complex setting : relational data. In these contexts, objects are represented by both attributes and binary relations with other objects. The need to represent relation information causes an exponential increase in the number of attributes so naive algorithms become unusable extremely fast. We propose a modification of our algorithm that enumerates the pseudo-intents of contexts in which relational information is represented by attributes. A quick study of the type of relational information that can be considered is provided. We use the example of description logics as a framework for expressing relational data.

In the third part, we extend our work to the more general domain of association rules. Association rules are regularities of the form "when there is A there is B with x% certainty" so implications are association rules with 100% certainty. Our algorithm already computes a basis for implications so we propose a very simple modification and demonstrate that it can compute the Luxenburger basis of a context along with the Duquenne-Guigues basis. This effectively allows our algorithm to compute a basis for association rules that is of minimal cardinality.

Résumé

Cette thèse traite du problème du calcul des implications, c'est-à-dire des régularités de la forme "quand il y a A, il y a B", dans des ensembles de données composés d'objets décrits par des attributs. Calculer toutes les implications peut être vu comme l'énumération d'ensembles d'attributs appelés pseudo-intensions. Nous savons que ces pseudo-intensions ne peuvent pas être énumérées avec un délai polynomial dans l'ordre lectique mais aucun résultat n'existe, à l'heure actuelle, pour d'autres ordres. Bien que certains algorithmes existants n'énumèrent pas forcément dans l'ordre lectique, aucun n'a un délai polynomial. Cette absence de connaissances sur les autres ordres autorise toujours l'existence d'un algorithme avec délai polynomial et le trouver serait une avancée utile et significative. Malheureusement, les algorithmes actuels ne nous autorisent pas à choisir l'ordre d'énumération, ce qui complique considérablement et inutilement l'étude de l'influence de l'ordre dans la complexité. C'est donc pour aller vers une meilleure compréhension du rôle de l'ordre dans l'énumération des pseudo-intensions que nous proposons un algorithme qui peut réaliser cette énumération dans n'importe quel ordre qui respecte la relation d'inclusion.

Dans la première partie, nous expliquons et étudions les propriétés de notre algorithme. Comme pour tout algorithme d'énumération, le principal problème est de construire tous les ensembles une seule fois. Nous proposons pour cela d'utiliser un arbre couvrant, lui-même basé sur l'ordre lectique, afin d'éviter de multiples constructions d'un même ensemble. L'utilisation de cet arbre couvrant au lieu de l'ordre lectique classique augmente la complexité spatiale mais offre plus de flexibilité dans l'ordre d'énumération. Nous montrons que, comparé à l'algorithme Next Closure bien connu, le nôtre effectue moins de fermetures logiques sur des contextes peu denses et plus de fermetures quand le nombre moyen d'attributs par objet dépasse 30% du total. La complexité spatiale de l'algorithme est aussi étudiée de façon empirique et il est montré que des ordres différents se comportent différemment, l'ordre lectique étant le plus efficace. Nous postulons que l'efficacité d'un ordre est fonction de sa distance à l'ordre utilisé dans le test de canonicité.

Dans la seconde partie, nous nous intéressons au calcul des implications dans un cadre plus complexe : les données relationnelles. Dans ces contextes, les objets sont représentés à la fois par des attributs et par des relations avec d'autres objets. Le besoin de représenter les informations sur les relations produit une augmente exponentielle du nombre d'attributs, ce qui rend les algorithmes classiques rapidement inutilisables. Nous proposons une modification de notre algorithme qui énumère les pseudo-intensions de contextes dans lesquels l'information relationnelle est représentée par des attributs. Nous fournissons une étude rapide du type d'information relationnelle qui peut être prise en compte. Nous utilisons l'exemple des logiques de description comme cadre pour l'expression des données relationnelles.

Dans la troisième partie, nous étendons notre travail au domaine plus général des règles d'association. Les règles d'association sont des régularités de la forme "quand il y a A, il y a B avec une certitude de x%". Ainsi, les implications sont des règles d'association certaines. Notre algorithme calcule déjà une base pour les implications et nous proposons une très simple modification et montrons qu'elle lui permet de calculer la base de Luxenburger en plus de la base de Duquenne-Guigues.

Cela permet à notre algorithme de calculer une base de cardinalité minimale pour les règles d'association.

Remerciements

Je tiens à remercier tous ceux qui, à un titre ou à un autre, m'ont apporté aide et soutien pour l'élaboration et la rédaction de ma thèse.

J'exprime tout d'abord ma profonde reconnaissance à Monsieur le Professeur Jean-Gabriel Ganascia, mon directeur de thèse, qui m'a chaleureusement accueilli dans son équipe et dont les conseils, dispensés avec attention et rigueur, m'ont guidé dans mon travail.

J'assure de ma gratitude Madame Karell Bertet et Monsieur Henry Soldano, qui ont accepté de consacrer une part significative de leur temps à l'étude de mon manuscrit ainsi que Madame la Professeur Annick Valibouze et Monsieur Alain Gély qui m'honorent de leur participation au jury.

Mes remerciements vont aux membres permanents ou non, anciens ou actuels, de l'équipe ACASA au sein de laquelle j'ai travaillé pendant quatre ans et tissé des liens d'amitié.

Ils vont encore aux personnels technique et administratif du LIP6, dont la compétence et la gentillesse m'ont permis de travailler dans les meilleures conditions possibles.

Les membres de la communauté francophone des treillis m'ont amicalement soutenu de leurs remarques et de leurs conseils. Je leur dois beaucoup. Qu'ils sachent que je leur en sais profondément gré.

Enfin, ma pensée va vers mes parents, qui m'ont soutenu de façon indéfectible tout au long de ces années, ainsi que vers ma famille et mes amis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Research Context

Data mining is the science of finding interesting information in data. More often than not, this information takes the form of regularities that either help predict future results or understand the nature of the phenomenon that generated the data. Depending on the kind of information one wants to extract, different mathematical frameworks are used. We mainly differentiate methods using probabilities and approximations from those manipulating sets, lattices or graphs. In this work, we are interested in the latter.

Formal concept analysis is a mathematical framework introduced by Rudol Wille [104] in the 1980s. Based on lattice theory and manipulating intuitive "concepts", it naturally finds its uses in knowledge discovery, data analysis and visualization. Of particular interest is the notion of implication. Implications are regularities of the form "if there is X, then there is always Y". Various fields, such as association rules mining and databases, are concerned with the problem of finding those implications in data composed of sets of objects associated with attributes. The problem of computing a minimal set of implications that summarizes them all, the Duquenne-Guigues basis, has been extensively studied in formal concept analysis. While algorithms exist, their efficiency is limited by multiple factors. First, the number of implications to find can be exponential in the size of the input data. Hence, algorithms are exponential and nothing can be done about it. Second, batch algorithms usually compute the set of implications in the so-called lectic order and it has been shown that this does not guarantee a polynomial delay between two implications. There is no such result for the enumeration of implications in other orders so this leaves the possibility for a polynomial-delay algorithm to exist. However, the effects of the enumeration order in the computation of the Duquenne-Guigues basis is poorly understood. Since batch algorithms do not allow for other orders and since other approaches prevent fine-tunings, studying the role of the order is harder than necessary.

1.2 Contributions

This work aims at providing a first step towards a better understanding of the role of the enumeration order in the computation of the Duquenne-Guigues basis of implications, both in traditional object-attributes and relational data. If this role is to be studied, it is necessary to first be able to efficiently enumerate in different orders. We thus provide an algorithm that allows for any order that extends the inclusion order to be used in the enumeration of the elements of the basis.

Evidently, this freedom comes at a price in terms of time and space and, compared to specialized algorithms that enumerate in a single order, we risk a loss of efficiency that would render the algorithm unusable in practice. In order to ensure this is not the case, we empirically studied the time and space complexity of the algorithm. For the time, we compared it with Next Closure, the most studied algorithm for this problem, using the number of logical closures - the most expensive operation and the main difference between the two algorithms. For the space, we chose three orders we believe to be good representatives of how different orders produce different behaviors and studied the memory consumption of our algorithm

when enumerating with them.

Once sure the algorithm works correctly and efficiently on traditional data, i.e. sets of objects described by sets of attributes, we showed it could be used on relational data – objects described by both attributes and relations with other objects – through the same kind of method used by existing algorithms. We use the example of description logics and provide a more general formalization of the representation of relational knowledge by attributes.

Finally, we went back to the more general notion of association rules to which implications belong. We showed that the properties of our algorithm make it easily modifiable to compute a basis for association rules along with just the minimal basis for implications.

1.3 Organization

This thesis is organized as follows :

Chapter 2 contains the notions needed to understand the principles behind our proposed algorithm. It provides some basic definitions in order theory, lattice theory and formal concept analysis. It also presents known algorithms - most notably Next Closure - for the problem of computing the Duquenne-Guigues basis and important results on the complexity of the enumeration and recognition of pseudo-intents as well as a brief overview of the different methods for computing the logical closure of sets.

Chapter 3 introduces our proposed algorithm and its properties. It starts with an explanation of the method we use for the construction of attribute sets and continues with the algorithm itself. Then, it presents and analyzes the results of our empirical comparison with Next Closure using, as a metric, the number of logical closures performed in the average and worst cases. Finally, we provide an empirical study of the space complexity of the enumeration in three different orders with our algorithm.

Chapter 4 deals with with the use of our algorithm on relational data. It contains a brief overview of description logics as an example of a formalism for expressing relational knowledge along with related works on the subject. It also presents how to use our algorithm on data in which attributes are used to represent relational knowledge. It ends with an analysis of the type of relational knowledge it is able to handle.

Chapter 5 addresses the modification of our algorithm that allows for a basis of association rules to be computed. After some basic definitions, notions and history concerning association rules and their mining, the method that allows us to efficiently compute the Luxenburger basis along with the Duquenne-Guigues basis is explained.

Chapter 6 presents potentially good ideas that will be the subject of future works and draws conclusions.

Chapter 2

Definitions, Results and Algorithms

Contents

2.1 Basic Definitions

2.1.1 Formal Concept Analysis

Formal concept analysis is a mathematical framework for the detection of regularities in data and representation of knowledge.

Definition 1 *A formal context is a triple* $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ *in which* $\mathcal{O}$ *is a set of objects,* $\mathcal{A}$ *a set of attributes and* $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$ *a binary relation between objects and attributes.*

A formal context represents knowledge, or data, on a set of objects. If, for an object o and an attribute a, the pair (o, a) is in $\mathcal{R}$ then we say that a *describes* o. By extension, if all the attributes in a set A describe an object o, then we will say that A *describes* o. Attributes are essentially unary predicates, and objects are constants.

A formal context can be represented by a cross table, as seen in Figure 2.1.

	a	b	c	d	e
o1	×	×			
o2		×		×	×
o3		×	×	×	
o4			×		×
o5				×	×

Figure 2.1: Context 1

The formal context represented in Figure 2.1 will be used as a running example throughout this work. For the sake of simplicity, sets of attributes $\{a, b, c\}$ will be abbreviated abc from now on.

Formal concept analysis makes use of two operators, both noted $.'$, that are defined as follows :

$$.' : 2^{\mathcal{A}} \mapsto 2^{\mathcal{O}}$$
$$A' = \{o \in \mathcal{O} \mid \forall a \in A, (o, a) \in \mathcal{R}\}$$

$$.' : 2^{\mathcal{O}} \mapsto 2^{\mathcal{A}}$$
$$O' = \{a \in \mathcal{A} \mid \forall o \in O, (o, a) \in \mathcal{R}\}$$

The $.'$ operator maps an attribute set A to the set of all the objects that are described by A. Applied on an object set O, it maps it to the set of all the attributes the objects of O have in common. The composition of these two operators forms a Galois connection and, thus, the two compositions are closure operators on $\mathcal{O}$ and

$\mathcal{A}$ respectively. A set X such that $X = X''$ is said to be *closed under* $.''$ or more simply *closed* when there is no ambiguity on the operator. Computing the closure of a set is in $O(|\mathcal{O}| \times |\mathcal{A}|)$. In our example, the closure of a is ab and the closure of bcd is bcd.

Definition 2 *An attribute set I that is closed under $.''$ is called an intent. An object set E closed under $.''$ is called an extent.*

Definition 3 *A pair (E, I) in which E is an extent and I an intent such that $E' = I$ and $I' = E$ is called a formal concept.*

When two formal concepts (E, I) and (K, J) are such that $K \subseteq E$ and/or $I \subseteq J$, we will say that (E, I) *is more general than* (K, J) and note $(E, I) \leq (K, J)$. In a given formal concept $\mathcal{C}$, the set of all concepts together with $\leq$ forms the *concept lattice of* $\mathcal{C}$. This lattice is a complete lattice.

Figure 2.2 represents the concept lattice of Context 1.

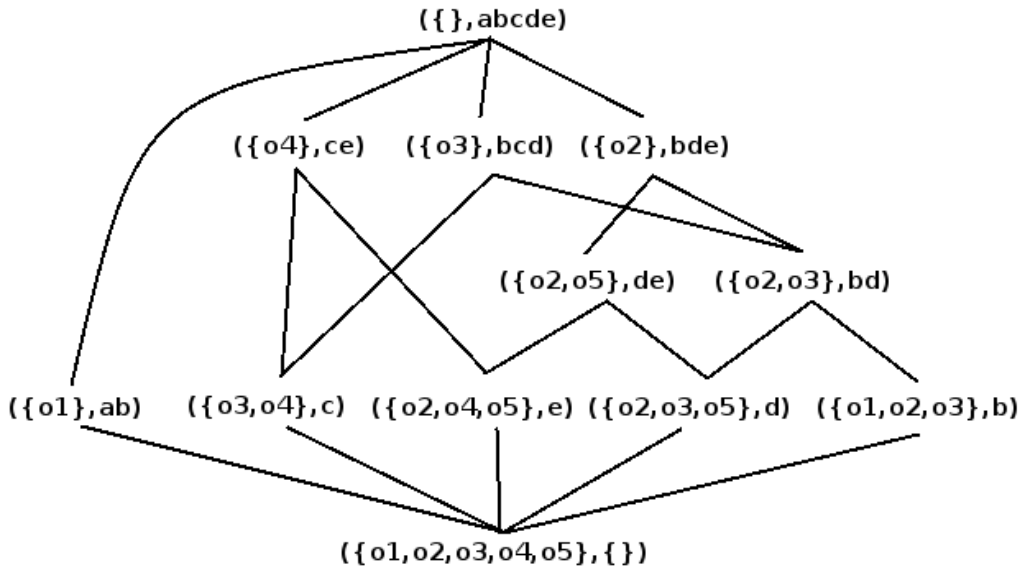


Figure 2.2: Concept Lattice of Context 1

Computing the concept lattice of a context, or even just the set of formal concepts, is an important problem that has been extensively studied. We will look into it in Section 2.3.

2.1.2 Implications

The other major FCA problem is the computation of implications.

Definition 4 *An implication between two attribute sets A and B, noted $A \to B$, is said to hold in a context $\mathcal{C}$ if and only if $A' \subseteq B'$.*

From this definition, it is clear that, for any two attribute sets A and B such that $A \subseteq B$, the implication $B \to A$ holds. Most notably, an implication $A \to A$ always holds. Finding the set of all the implications that hold is one of the great problems in FCA. Of course, as it is often the case with this kind of problem, we are more interested in an "interesting" subset than in the whole set. For example, we say that implications $A \to B$ with $B \subseteq A$ are *trivial*. From there, the problem is trying to find all non-trivial implications that hold in a given context.

Definition 5 *A set $\mathcal{L}$ of implications is a cover if and only if all the implications in $\mathcal{L}$ hold and we can derive all the other implications from $\mathcal{L}$ using Armstrong's rules :*

$$\frac{B \subseteq A}{A \to B}, \quad \frac{A \to B}{A \cup C \to B \cup C}, \quad \frac{A \to B, B \to C}{A \to C}$$

The number of implications that hold can be quite high and, as many of them are redundant, it is necessary to focus on the interesting ones. For example, if an implication $A \to B$ holds, then $A \to C$ also holds for every $C \subseteq B$. In order to take that into account, we can use the following proposition.

Proposition 1 *An implication $A \to B$ holds if and only if $B \subseteq A''$.*

Proof Every object described by A is also described by A''. Hence, $A \to A''$ holds and the same can be said about $A \to B$ for any $B \subseteq A''$.

Let C be an attribute set such that $C \not\subseteq A''$. From the definition of a closure operator, we deduce that there is an attribute $i \in C$ such that $A' \setminus i' \neq \emptyset$. Since there is an object described by A but not by i, the implication $A \to C$ does not hold. Thus, $A \to B$ holds only if $B \subseteq A''$. □

This proposition states that $\mathcal{L} = \{A \to A'' \mid A \subseteq \mathcal{A} \text{ and } A \neq A''\}$ is a cover for all the implications of a formal context. In our running example, this cover is :

- $a \to ab$
- $ac \to abcde$
- $ad \to abcde$
- $ae \to abcde$
- $bc \to bcd$
- $be \to bde$
- $cd \to bcd$
- $abc \to abcde$
- $abd \to abcde$
- $abe \to abcde$

- $acd \rightarrow abcde$
- $ace \rightarrow abcde$
- $ade \rightarrow abcde$
- $bce \rightarrow abcde$
- $cde \rightarrow abcde$
- $abcd \rightarrow abcde$
- $abce \rightarrow abcde$
- $abde \rightarrow abcde$
- $acde \rightarrow abcde$
- $bcde \rightarrow abcde$

Evidently, some of these implications are still redundant. For example, if we know that $abc \rightarrow abcde$ is valid, it is obvious that $abcd \rightarrow abcde$ is valid too. As such, this cover is not minimal.

Definition 6 *A basis $\mathcal{L}$ of implications is a minimal cover.*

Given the potentially very high number of implications that hold in a context, trying to find minimal sets of implications that summarize them all is therefore natural as well as interesting. Many bases have been proposed in the literature, each with different strengths and weaknesses. We will see examples of bases in Section 2.2.

2.1.3 Logical Closures

Definition 7 *If $\mathcal{L}$ is a set of implications then the logical closure of an attribute set A under $\mathcal{L}$, noted $\mathcal{L}(A)$ is defined as*

$$A^+ = A \cup \{C \mid B \rightarrow C \in \mathcal{L} \text{ and } B \subseteq A\}$$
$$\mathcal{L}(A) = A^{++\ldots+} \text{ (up to saturation)}$$

The logical closure is a closure operator. Sets closed under $\mathcal{B}$ are also closed under $.''$ so they form a lattice isomorphic to the concept lattice.

Definition 8 *If $\mathcal{L}$ is a set of implications then the logical **pseudo**-closure of an attribute set A under $\mathcal{L}$, noted $\mathcal{L}(A)$ is defined as*

$$A^\circ = A \cup \{C \mid B \rightarrow C \in \mathcal{L} \text{ and } B \subset A\}$$
$$\mathcal{L}^-(A) = A^{\circ\circ\ldots\circ} \text{ (up to saturation)}$$

The logical pseudo-closure is also a closure operator. The difference between the logical closure and pseudo-closure is that, in the latter, the conclusion of an implication is added to a set only if the premise is **strictly** contained in the set.

Let us suppose that $\mathcal{L} = \{b \to abe, de \to c\}$. We have $\mathcal{L}(b) = abe$, $\mathcal{L}(bd) = abcde$, $\mathcal{L}^-(b) = b$ and $\mathcal{L}^-(bd) = abcde$.

2.2 Bases of Implications

2.2.1 Duquenne-Guigues Basis

The most studied basis and the focus of our work is the Duquenne-Guigues basis. It is based on the notion of pseudo-intent.

Definition 9 *An attribute set P is a pseudo-intent if and only if $P \neq P''$ and $Q'' \subset P$ for every pseudo-intent $Q \subset P$.*

A pseudo-intent is a set of attributes that is not an intent and contains the closure of every pseudo-intents that are its subsets. As such, the definition is recursive. The closure of pseudo-intents are called *essential intents*.

Definition 10 *The set of all the implications of the form $P \to P''$ in which P is a pseudo-intent is called the Duquenne-Guigues basis of the context.*

The Duquenne-Guigues basis, also called *canonical basis*, has first been proposed in [57] and is the smallest (cardinality-wise) of all the bases.

In our running example, the set of pseudo-intents is $\{a, bc, cd, be, abd, bcde\}$ and the Duquenne-Guigues basis is therefore :

- $a \to ab$
- $bc \to bcd$
- $cd \to bcd$
- $be \to bde$
- $ab \to abcde$
- $bcde \to abcde$

It is clear here that computing the Duquenne-Guigues basis of a context is enumerating the pseudo-intents. This is not a trivial task, as presented in Section 2.4, and it has gathered much attention since it was first proposed.

2.2.2 Canonical Direct Basis

While the Duquenne-Guigues basis is the basis with the least cardinality, the *canonical direct basis*, noted here $\mathcal{B}_d$ is the smallest basis for which the logical closure $\mathcal{B}_d(A)$ is equal to A^+. In other words, the logical closure can be computed with a single pass through the canonical direct basis (hence the "direct"). However, the size of the canonical direct basis being much greater than the size of the Duquenne-Guigues basis, this advantage is relative.

The canonical direct basis was initially known under five independent definitions, shown to be equivalent in [21] by Bertet and Monjardet.

- The *optimal constructive basis* ([22]) defined as the direct basis with the least cardinality.
- The *weak implication basis* ([91]) defined using minimal transversals of a family.
- The *canonical iteration-free basis* ([103]) defined using minimal generators of sets.
- The *left minimal basis* ([99]) in which premises have minimal cardinality.
- The *dependence relation's basis* ([81]) defined using the dependency relation between irreducible elements of a lattice.

2.3 Computing the Set of Concepts

The computation of the set of all concepts in a given formal context presents two facets common to enumeration problems in general :

- how to generate all concepts
- how to avoid generating the same concept multiple times

Several strategies can be used to generate concepts. There are two kinds of algorithms : those beginning with the top concept $(\mathcal{O}, \mathcal{O}')$ and those beginning with the bottom concept $(\mathcal{A}', \mathcal{A})$. However, objects and attributes can be swapped so the principles are the same. Ultimately, algorithms differ mostly on the methods employed to generate new intents. For example, some algorithms add an attribute to already computed intents and compute the closure of the new set. The result is a new intent. Others intersect already known intents. The intersection of closed set being closed, the result is also an intent. The choice of a generation method is closely tied to the space complexity and the data structures employed.

Checking whether a concept has already been generated can be done using specific data structures or properties of the set. Some algorithms maintain a tree of concepts that allows for an efficient search of every newly generated concept. Others narrow the search space by dividing the set of all known concepts into smaller sets. For example, a hash function in the form of the cardinality of intents can be used.

Algorithm 1 Next

```
1: for every attribute i in decreasing order do
2:    B = A ⊕ {i}
3:    if min(B \ A) = i then
4:       Return B
5:    end if
6: end for
```

Newly generated concepts are then searched only in the set of concepts with an intent of the same cardinality, which makes sense. A third category of algorithms uses some form of lexicographical order on the set of all concepts. A newly generated concept is to be considered if it could not have been generated from another concept greater (or lesser) in the chosen order. The main advantage of this technique is that it usually only requires that we know of a *current* concept to be able to test whether new ones should be considered and is, for this reason, more efficient in terms of space complexity.

2.3.1 Next Closure

The Next Closure ([49]) algorithm is of particular interest because of its use in both the computation of the concept set and the Duquenne-Guigues basis. In fact, it can be used to compute the closed sets for any closure operator. It uses a canonicity test based on the *lectic order*.

Assuming the existence of a linear order on the set of attributes, a set A is lesser than a set B in the lectic order, noted $A \leq B$, if and only if $\min((A \cup B) \setminus (A \cap B)) \in B$. As such, the lectic order is a linear order on the set of intents and, consequently, the set of concepts. Next Closure computes the concepts in this order. In order to achieve this, it uses the $\oplus$ operator, defined as follows for any attribute set A and attribute $i \notin A$:

$$A \oplus \{i\} = (\{a \in A \mid a < i\} \cup \{i\})''$$

Hence, the set $A \oplus \{i\}$ is the closure of the set containing i and the attributes of A lesser than i. If the set A is the *current* set, then the *next* set, the one that immediately follows A in the lectic order, is equal to $A \oplus \{i\}$ where i is the greatest attribute such that $\min((A \oplus \{i\}) \setminus A) = i$. Using this property, Algorithm 1, called Next, computes the set immediately following its input set. It computes $A \oplus \{i\}$ for every attribute in decreasing order and, for each one, checks whether the property holds. It performs, at most, $|\mathcal{A}|$ computations of the $\oplus$ operator. Knowing that the closure of a set can be computed in $O(|\mathcal{A}| \times |\mathcal{O}|)$, Next is in $O(|\mathcal{A}|^2 \times |\mathcal{O}|)$, i.e. it is polynomial in the size of the context.

Next Closure computes all the concepts by starting with the closure of the empty set of attributes and generating all the intents one by one using Next. It stops once it reaches $\mathcal{A}$. The lectic order being a total order on the set of intents, generating in this fashion ensures that every concept is found and that an intent is constructed only from its predecessor in the order, which plays the role of the

Algorithm 2 Next Closure

1: $A = \emptyset''$
2: $Concepts = \emptyset$
3: **while** $A \neq \mathcal{A}$ **do**
4: $\quad Concepts = Concepts \cup \{(A', A)\}$
5: $\quad A = Next(A)$
6: **end while**
7: Return $Concepts$

canonicity test. Algorithm 2 is the version of Next Closure that enumerates the formal concepts.

Note that, in its classic form, Next Closure does not construct the concept lattice because the relations between the concepts are lost.

2.4 Computing the Set of Pseudo-Intents

The problem of computing the set of all pseudo-intents is closely related to the problem of computing intents. Indeed, both involve the enumeration of sets of attributes that respect certain properties. Moreover, in the case of batch algorithms, the search space for pseudo-intents contains the search space for intents.

2.4.1 Search Space

We have defined, in Section 2.2.1, a pseudo-intent as a set of attributes that is not closed and contains the closure of all its subsets that are pseudo-intents. It is not difficult to see that an intent is a set of attributes that is closed and contains the closure of all its subsets (or it would not be closed). If it contains the closure of all its subsets, it obviously contains the closure of all its subsets that are pseudo-intents. Under this definition, if a set contains the closure of all its subsets that are pseudo-intents, knowing whether it is an intent or a pseudo-intent only requires knowing whether it is closed or not.

An attribute set closed under $\mathcal{B}^-$ contains the conclusion of every implication which premise it strictly contains. Hence, in order to compute the set of pseudo-intents, and with it the Duquenne-Guigues basis, we must search for them in the set of attribute sets closed under $\mathcal{B}^-(.)$ and, in doing so, enumerate intents. It is currently unknown whether it is possible to enumerate pseudo-intents without intents.

Definition 11 *The lattice Φ is the set of attribute sets closed under $\mathcal{B}^-(.)$ ordered by inclusion.*

Figure 2.3 represents the lattice Φ corresponding to our running example.

Batch algorithms, most notably Next Closure, enumerate the entirety of Φ to find the set of pseudo-intents.

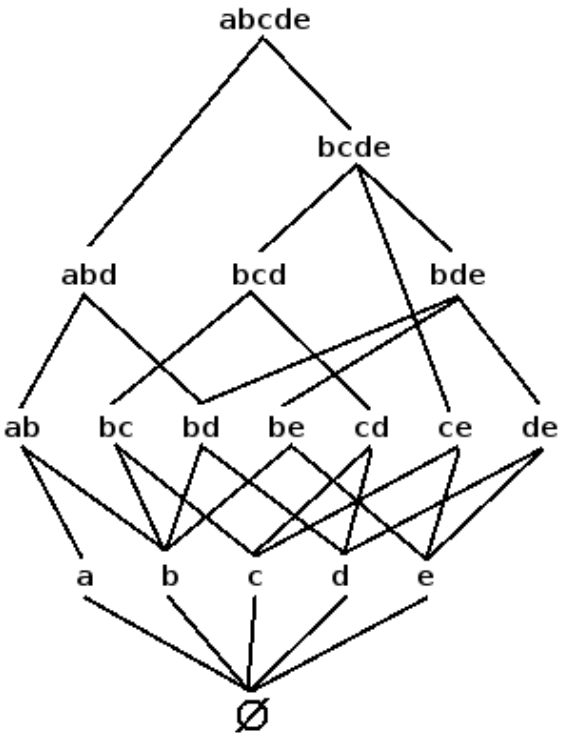


Figure 2.3: Lattice Φ of Context 1

2.4.2 Next Closure

The $\mathcal{B}^-$ operator being a closure operator, the Next Closure algorithm can be used to enumerate all the elements of the lattice Φ. As an element of Φ is either an intent or a pseudo-intent, computing the closure of every attribute set enumerated is enough to obtain the set of pseudo-intents. As such, the Next Closure presented in Algorithm 2 can easily be modified to also enumerate pseudo-intents instead of just intents. Only the closure operator has to be changed. The $\oplus$ operator used in the Next algorithm for the enumeration of pseudo-intents is as follows :

$$A \oplus \{i\} = \mathcal{B}^-(\{a \in A \mid a < i\} \cup \{i\})$$

When enumerating in the lectic order, the successor B of a set A is such that every subset of B has already been enumerated. Consequently, every pseudo-intent $P \subset B$ is known once it reaches B. If we use $\mathcal{I} \subseteq \mathcal{B}$ to denote the subset of the Duquenne-Guigues basis which premises are lesser than B in the lectic order, then for any subset C of B we have $\mathcal{I}^-(C) = \mathcal{B}^-(C)$. This means that, at any given step of the algorithm, the set of implications already found is enough to allow the computation of the logical closure under $\mathcal{B}^-$.

The version of Next Closure used for the computation of pseudo-intents is illustrated in Algorithm 3 ([17]). It starts with $\emptyset$ and enumerates the elements of Φ using Next. For each attribute set, it computes its closure. If it is not closed, it is a pseudo-intent and the corresponding implication is added to the list. The algorithm ends when $\mathcal{A}$ is reached.

The algorithm can be optimized somewhat by considering the fact that, if B

Algorithm 3 Next Closure for Pseudo-Intents

```
1: A = ∅
2: Impl = ∅
3: while A ≠ 𝒜 do
4:    B = A''
5:    if A ≠ B then
6:       Impl = Impl ∪ {A → B}
7:    end if
8:    A = Next(A)
9: end while
10: Return Impl
```

Algorithm 4 Improved Next Closure for Pseudo-Intents

```
1: A = ∅
2: Prev = ∅
3: Impl = ∅
4: while A ≠ 𝒜 do
5:    B = A''
6:    if A ≠ B then
7:       Impl = Impl ∪ {A → B}
8:       if min(B \ A) > min(A \ Prev) then
9:          A = B
10:      end if
11:   end if
12:   Prev = A
13:   A = Next(A)
14: end while
15: Return Impl
```

is a pseudo-intent that immediately follows a set A (in the lectic order), the set B'' immediately follows B if $\min(B'' \setminus B) > \min(B \setminus A)$. Let us suppose we are in this case and we have $i = \min(Next(B) \setminus B)$, meaning that i is the attribute that generates $Next(B)$. If $i < \min(B'' \setminus B)$, then $B \oplus \{i\}$ is lectically greater than B''. We have a contradiction because it would skip B'' in the enumeration.. If $i > \min(B'' \setminus B)$, then $B \oplus i$ contains B but not B'', which means it is not closed under $\mathcal{B}^-$. Consequently, we have $i = \min(B'' \setminus B)$ and, hence, $Next(B) = B''$.

On the contrary, if $\min(B\text{"} \setminus B) < \min(B \setminus A)$, then $B \oplus j$ with $j > \min(B \setminus A)$ cannot be the result of $Next(B)$ because it would contain $\min(B \setminus A)$. We can thus continue computing as if $A \oplus \min(B \setminus A)$ had been rejected by Next. Algorithm 4 incorporates this optimization.

Next Closure enumerates every attribute set in Φ and, for each of them, computes its closure and its successor. Computing the closure of a set can be done in polynomial time and the Next algorithm requires at most $\mathcal{A}$ applications of the $\mathcal{B}^-$ operator. A logical closure can be computed in $O(|\mathcal{A}| \times |\mathcal{B}|)$ so Next Closure is in $O(|\Phi| \times |\mathcal{A}|^2 \times |\mathcal{B}|)$. One of the biggest advantages of Next Closure is its space complexity. At any given time, we only need to know the current set, along

with the context, to be able to compute both the closure and the successor. This constant space complexity is especially important when we consider that the size of Φ can be - and usually is - exponential in the size of the context.

In our running example, NEXT CLOSURE enumerates the elements of Φ in the order depicted in Figure 2.4.

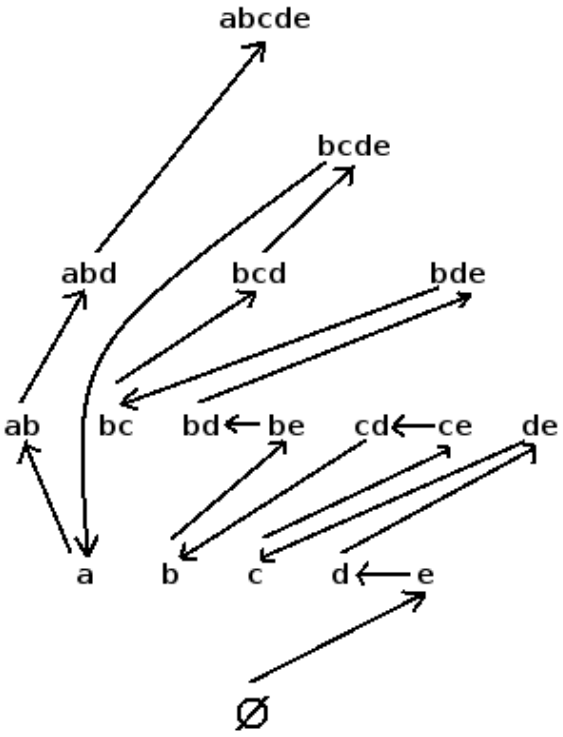


Figure 2.4: Lectic order on Φ

The algorithm runs as follows :

The current set is $\emptyset$. It is an intent.
$\emptyset \oplus e = e$ is the next set.
The current set is e. It is an intent.
$e \oplus d = d$ is the next set.
The current set is d. It is an intent.
$d \oplus e = de$ is the next set.
The current set is de. It is an intent.
$de \oplus c = c$ is the next set.
The current set is c. It is an intent.
$c \oplus e = ce$ is the next set.
The current set is ce. It is an intent.
$ce \oplus d = cd$ is the next set.
The current set is cd. It is a pseudo-intent so we add $cd \rightarrow bcd$ to the set of implications.
$ce \oplus b = b$ is the next set
The current set is b. It is an intent.
$b \oplus e = be$ is the next set.

The current set is be. It is a pseudo-intent so we add $be \rightarrow bde$ to the set of implications.
$b \oplus d = bd$ is the next set.
The current set is bd. It is an intent.
$bd \oplus e = bde$ is the next set.
The current set is bde. It is an intent.
$bde \oplus c = bc$ is the next set.
The current set is bc. It is a pseudo-intent so we add $bc \rightarrow bcd$ to the set of implications. The new current set is bcd.
$bcd \oplus e = bcde$ is the next set.
The current set is $bcde$. It is a pseudo-intent so we add $bcde \rightarrow abcde$ to the set of implications.
$bcd \oplus a = a$ is the next set.
The current set is a. It is a pseudo-intent so we add $a \rightarrow ab$ to the set of implications. The new current set is ab.
$ab \oplus e = abde$ is not the next set ($d < e$).
$ab \oplus d = abd$ is the next set.
The current set is abd. It is a pseudo-intent so we add $abd \rightarrow abcde$ to the set of implications.
$ab \oplus c = abcde$ is the next set.
The current set is $abcde$ and the algorithm stops.

2.4.3 Non-Batch Methods

Even though we are interested mainly in batch algorithms in the present work, other methods have been proposed to build the Duquenne-Guigues basis.

In [84], Obiedkov and Duquenne proposed an attribute-incremental algorithm based around the idea that the closure of a set can change when a new attribute (column) is added to the context. It uses a sequence of contexts $\mathcal{C}_x = (\mathcal{O}, \mathcal{A}_x, \mathcal{R}_x)$ in which $\mathcal{A}_x$ contains the x first attributes of $\mathcal{A}$. The algorithm computes the set of intents and pseudo-intents in the context $\mathcal{C}_{x+1}$ from the intents and pseudo-intents in the context $\mathcal{C}_x$.

Sets A are said to be $x - modified$ if the closure of $A \setminus \{x\}$ in $\mathcal{C}_x$ contains x and $x - stable$ otherwise. In other words, x-stable sets are sets which closure is not modified when adding the attribute x to the context. By differentiating attribute sets that are modified when adding new attributes, it is possible to reduce the number of sets to consider at each step. For each attribute x, the algorithm computes the set of intents and pseudo-intents in the context $\mathcal{C}_x$ using the intents and pseudo-intents of $\mathcal{C}_{x-1}$ and treating them differently if they are x-stable or x-modified. Experiments show that this algorithm seems to considerably outperform Next Closure on both public and randomly generated contexts.

In [101], Valtchev and Duquenne proposed to use a divide-and-conquer approach in which the set of attributes is split in two, which splits the context itself. The algorithm computes the lattices of concepts and pseudo-intents corresponding to these two contexts. It then computes the direct product of these two lattices and use the result as a sort of search space for the Duquenne-Guigues basis of the whole context. As with the attribute-incremental method, experiments suggest that this

approach outperforms Next Closure when the total runtime is the considered metric.

2.4.4 Problems

The enumeration of all the pseudo-intents of a given context is difficult because of both inherent properties and lack of knowledge.

The first problem is the size of the basis itself. Even though the number of pseudo-intents is often exponentially smaller than the total number of implications that hold in the context, the size of the Duquenne-Guigues basis of a context $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ can still be exponential in the size of the context, $|\mathcal{O}| \times |\mathcal{A}|$, as demonstrated by Kuznetsov in [68]. Of course, if there is an exponential number of entities to be found, a polynomial algorithm is out of the question.

The second problem is the search space. Currently, we do not know whether it is possible to enumerate all the pseudo-intents without also enumerating the intents of the context. While it is obvious that the number of intents can be exponential in the size of the context, there are also cases in which the number of intents is exponential in the number of pseudo-intents. As such, algorithms such as Next Closure that enumerate both intents and pseudo-intents are not even output-polynomial.

Even though the problem cannot be solved by an existing output-polynomial algorithm, we could wonder whether there is an algorithm with a polynomial delay - meaning that, given a pseudo-intent, another one could be found in polynomial time. Representing an important step toward an answer, Distel and Sertkaya have shown in [38] that pseudo-intents could not be enumerated **in lexicographic order** with a polynomial delay. This of course implies that Next Closure does not have a polynomial delay. In the same work, the authors have tried to show whether the problem could be solved in polynomial time when removing the restrictions on the order of enumeration. They managed to show that, among other results, the problem of deciding whether an additional pseudo-intent has yet to be found resides in NP, and linked it to well-known problems in graph theory. However, they did not manage to find a lower bound to the problem.

While these results are all but encouraging, at the time of this work, nothing proves that we cannot find an output-polynomial algorithm or an algorithm with a polynomial delay, provided it does not enumerate pseudo-intents in a lexicographic order.

2.4.5 Results on Pseudo-Intents

Pseudo-intents have been studied for many years, yet they are not that well understood. Some of their properties may help facilitate their enumeration. In [14], Babin and Kuznetsov investigated the complexity of recognizing various types of intents and pseudo-intents. They showed that the problem of deciding whether a set P is a pseudo-intent in a given context $\mathcal{C}$ is coNP-complete, recognizing the lectically greatest pseudo-intent is coNP-hard and recognizing an essential intent is NP-complete.

In [90], Rudolph proposed a non-recursive characterization of pseudo-intents using the notion of *incrementor*. Incrementors are sets that, once added to the context

Algorithm 5 Naive Logical Closure

```
1: stable = false
2: while stable == false do
3:    stable = true
4:    for every implication X → Y in ℒ do
5:       if A ⊆ X then
6:          A = A ∪ Y
7:          if A changed then
8:             stable = false
9:          end if
10:      end if
11:   end for
12: end while
13: Return A
```

as the extent of a new object, only "slightly" modify the intents of the contexts in that the old intents stay the same and only the new set becomes an intent. Pseudo-intents are showed to be a subset of the incrementors. This non-recursive definition, while not used in our present work, appears very promising as it could help relax the inclusion constraint on the order in which pseudo-intents can be enumerated.

In [54], Gély et al. attempted to explain the exponential number of pseudo-intents. They supposed that so-called *P-clone attributes* are the reason for the combinatorial explosion. Attributes a and b are P-clone attributes if replacing a by b (b by a) in every pseudo-intent that contains a (b) gives a pseudo-intent. The goal is to reduce the size of the Duquenne-Guigue basis by identifying and removing P-clone attributes. This is an approach that tries to reduce the size of the output instead of finding more efficient algorithms that compute the whole set.

2.5 Algorithms for Computing the Logical Closure

Given a set of implications $\mathcal{L}$, the logical closure of an attribute set A under $\mathcal{L}$, $\mathcal{L}(A)$, defined in Section 2.1.3 is a crucial operation that takes a significant amount of time. Three main algorithms can be used to compute $\mathcal{L}(A)$. The naive method, illustrated in Algorithm 5, is simply going through every implication until one of them adds an attribute to the set, at which point we begin again with the new set. It stops when all the implications have been considered and the set has not been modified, which means a fixed point has been reached. For each implications, the algorithm must test the inclusion of A in the premise, which can be done in $O(|A|)$ using an adequate representation. It must go through the set of implications a maximum of $|\mathcal{L}(A) \setminus A| + 1$ times so the algorithm runs in $O(|A| \times |\mathcal{L}| \times |\mathcal{L}(A) \setminus A| + 1)$.

LinClosure is a linear-time algorithm proposed in [18]. It associates to each implication in $\mathcal{L}$ a counter - initially equal to the size of the premise - and to each attribute a list of the implications in which premises it appears. Each attribute of A is used to decrement the counter of the corresponding implications and when

Algorithm 6 LinClosure

```
Initialization of the lists and counters
for every attribute a in A do
   for every implication X → Y in list(a) do
      Decrement the counter of X → Y
      if the counter reaches 0 then
         A = A ∪ Y
      end if
   end for
end for
Return A
```

Algorithm 7 Wild's Closure

```
Initialization of the lists and counters
stable = false
while stable == false do
   stable = true
   Imp = ⋃_{a∉A} list(a)
   for every implication X → Y in L \ Imp do
      A = A ∪ Y
      if A changed then
         stable = false
      end if
   end for
end while
Return A
```

one reaches zero, the conclusion is added to A. The algorithm stops when all the attributes have been used. The manipulation of the counters is in $O(|\mathcal{L}(A)| \times |\mathcal{L}|)$ but it is important to note that the initialization of the lists and counters can take some time too. Algorithm 6 illustrates LinClosure.

The third algorithm, Wild's Closure, has been proposed in [103]. It also uses lists of implications associated to each attribute but no counters. Instead, it computes the union of the lists associated to attributes that are not in A. Obviously, implications that are not in this union have premises that contain only attributes of A so they are used to modify A. This is then repeated until the fixed point is reached. This algorithm has the same complexity as LinClosure.

A comparison of these three algorithms in the context of the enumeration of pseudo-intents can be found in [17]. Results tend to show the naive algorithm as more efficient in general, probably because the other two are handicapped by the initialization phase.

Chapter 3

Choosing the Enumeration Order

Contents

3.1 Motivation

The bulk of our work is centered on the enumeration of pseudo-intents. We saw in Section 2.4.4 that pseudo-intents cannot be enumerated in lectic (or reverse lectic) order with a polynomial delay and that no such result exists for the enumeration in other orders. Unfortunately, known batch algorithms (Next Closure) work only with the lectic order and the others (attribute-incremental and divide-and-conquer approaches) do not allow for much customization in the choice of the order. On the applicative side of things, algorithms enumerate too many sets which becomes problematic as soon as contexts get decently big. Solutions would be to reduce the number of intents enumerated with pseudo-intents or, more easily, reduce the number of operations needed to compute the sets.

We believe that the order of enumeration is the key and that it would be possible to either find an order in which we can enumerate pseudo-intents with a polynomial delay or, if it is impossible, to prove it. In the latter case, the order could help optimize the runtime of the algorithms. Having an algorithm that allows for the enumeration of pseudo-intents in a chosen order would be a good first step as it would help us get a better understanding of the influence of the order on the enumeration.

In this chapter, we propose an algorithm that enumerates pseudo-intents in any order that respects the inclusion order. First, we present an algorithm, Successor, for the construction of attribute sets that uses a canonicity test similar to that of Next but does not force us to enumerate in the lectic order. Then, we use Successor to build an algorithm that enumerates the elements of Φ in any order and we empirically compare it with Next Closure using the number of logical closures as a metric. Lastly, we empirically study the effects of three different orders on the space complexity of the algorithm and use the results to infer the effects of all possible orders.

3.2 Construction of Attribute Sets

Constructing the Duquenne-Guigues basis requires constructing attributes sets, be they intents or pseudo-intents. The Next algorithm is efficient because of its canonicity test that allows for a very low space complexity and an easy generation of attribute sets. However, the Next algorithm works specifically for an enumeration in the lectic order. If we are to consider other orders, we need to find another construction method that uses another canonicity test. We wish to keep the efficiency of the lectic order while not being forced to enumerate in the lectic order. In order to do this, we propose a *successor relation* that is defined using the lectic order.

Definition 12 ***Successor relation*** *For any two attribute sets $A, B \in \Phi$, B is a successor of A (noted $A \rightsquigarrow B$) if and only if A is the lectically greatest strict subset of B in Φ.*

When $A \rightsquigarrow B$, the set A is said to be the predecessor of B and the set B a successor of A. An attribute set always has a single predecessor and can have any number of successors, including none. The only notable exceptions are $\emptyset$ which doesn't have a predecessor and $\mathcal{A}$ which never has a successor.

Proposition 2 *For any $A \in \Phi$, there is a unique finite sequence $A_1, ..., A_n$ of elements of Φ such that $\emptyset \rightsquigarrow A_1 \rightsquigarrow ... \rightsquigarrow A_n \rightsquigarrow A$.*

Proof Every non-empty set in the lattice Φ having a strict subset as a predecessor, such a sequence exists. The number of attributes is finite so the sequence itself is finite. The predecessor of an attribute set is unique so the sequence is unique. □

Corollary 1 *The successor relation $\rightsquigarrow$ defines a spanning tree of the covering graph of Φ*

Figure 3.1 is the spanning tree of the lattice Φ corresponding to our running example. $\{bc, bd, be\}$ is the set of successors of b.

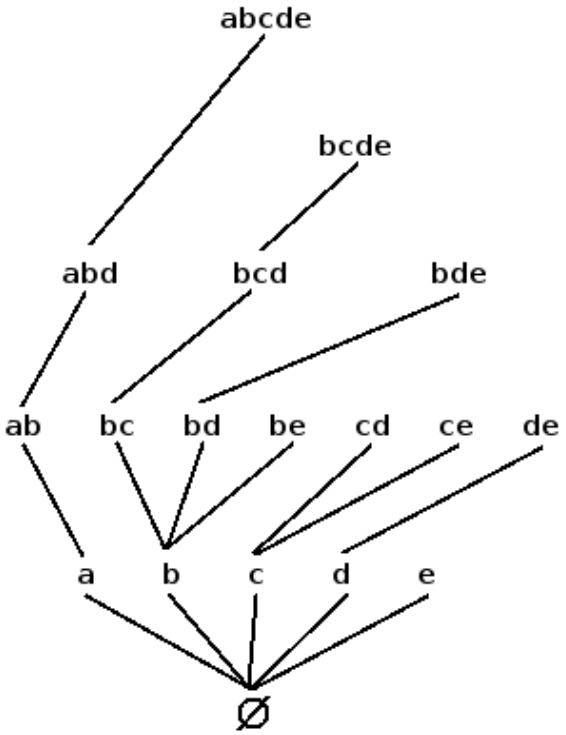


Figure 3.1: Spanning tree of the covering graph of Φ for Context 1 induced by the successor relation

When enumerating in the lectic order, constructing the next attribute set from the current one ensures that each set is considered only once because the order is total and each set is constructed from a single other set. The predecessor A of an element $B \in \Phi$ being unique (and always existing for non-empty sets), the same property is found when constructing attribute sets from their predecessor. From Proposition 2, we know that every set in Φ can be found by starting from $\emptyset$ and constructing the lattice along the edges of the tree. As such, using the partial order induced on Φ by $\rightsquigarrow$ instead of the lectic order would allow for the same efficiency in the canonicity test.

On that ground, in order to enumerate the elements of Φ in using $\rightsquigarrow$, we must be able to efficiently recognize whether a set B is a successor of a given set.

Algorithm 8 Successors(A)

```
1: Successors = ∅
2: for every attribute i greater than min(A) do
3:    B = A ⊕ i
4:    if A ⊂ B then
5:       if i = min(B \ A) and ∀j ∈ B \ A, A ⊕ j = B then
6:          Successors = Successors ∪ {B}
7:       end if
8:    end if
9: end for
10: return Successors
```

Proposition 3 *For any two sets $A, B \in \Phi$ such that $A \subset B$, we have $A \rightsquigarrow B$ if and only if $B = A \oplus i$ for every $i \in B \setminus A$.*

Proof $\Leftarrow$. We know that $A \oplus i$ is lectically greater than A and is a subset of B if $i \in B$. If $\forall i \in B \setminus A, A \oplus i = B$, then B has no strict subset in Φ lectically greater than A. As such, $A \rightsquigarrow B \Leftarrow \forall i \in B \setminus A, A \oplus i = B$.

$\Rightarrow$. If $A \rightsquigarrow B$ and $\exists i \in B \setminus A$ such that $A \oplus i \subset B$, then B has a subset in Φ lectically greater than A, which contradicts the hypothesis. As such, $A \rightsquigarrow B \Rightarrow \forall i \in B \setminus A, A \oplus i = B$. $\square$

To recognize a successor B of A, it is necessary to know $A \oplus i$ for every $i \in B \setminus A$. By extension, knowing the set of successors of A is knowing the value of $A \oplus i$ for every attribute i. Obviously, if an attribute i is lesser than the minimal attribute of A, the set $A \oplus i$ is not a superset of A and cannot be a successor. It is thus unnecessary to compute the value of $A \oplus i$ for any $i < \min(A)$.

Algorithm 8, Successors, computes the set of successors using Proposition 3. In its most basic form, the algorithm has to compute $A \oplus i$, i.e. perform a logical closure, once for every attribute greater than $\min(A)$. As we saw in Section 2.3.1, the Next algorithm performs a logical closure for every attribute greater than some attribute j for which it finds that $j = \min((A \oplus j) \setminus A)$. In practice, this attribute j is greater than $\min(A)$ for nearly every attribute set. The only exceptions are the sets A such that the cardinality of $Next(A)$ is 1, i.e. once for every attribute in $\mathcal{A}$. This means that Next outperforms Successors in the majority of cases.

This is where using the structure of the lattice Φ is important as it helps us reduce the number of required logical closures.

Proposition 4 *Algorithm 8 stops and returns the set of successors of the input set A.*

Proof The number of attribute is finite so the loop will eventually end and the algorithm stop. The set $A \oplus i$ is equal to $\{i\}$ when $i < \min(A)$ so it cannot be a successor. According to Proposition 3, every successor is equal to $A \oplus i$ for some $i > \min(A)$ so the algorithm finds them all. $\square$

Proposition 5 *If B is a successor of A with $i = \min(B \setminus A)$, then $B \oplus j = A \oplus j$ for every attribute $j < i$.*

Proof If $j < i$, then $\mathcal{B}^-(\{b \in B \mid b < j\} \cup \{j\}) = \mathcal{B}^-(\{a \in A \mid a < j\} \cup \{j\})$ because $i = \min(B \setminus A)$. Thus, we have $B \oplus j = A \oplus j$. $\square$

This proposition states that the value of $A \oplus i$ can be passed on to the successors of A that differ only on attributes greater than i. If we have $A \rightsquigarrow B$ and want to compute the successors of B, then the successors of A have already been computed and we know the value of $A \oplus i$ for all i. As such, we only have to compute $B \oplus i$ for i greater than $\min(B \setminus A)$ and this greatly reduces the number of operations. Whether this property alone makes Successors outperform Next depends on the context. Empirical results and their analysis can be found in Section 3.4.

In our running example of Context 1, the algorithm would run as follows for the set bd :

- $bd \oplus e = bde$ is computed, the attribute e is the minimal new attribute and also the only one. Thus, bde is a successor of bd
- $bd \oplus d$ is not computed since d is already in bd
- $bd \oplus c = bc$ is not computed because $b \rightsquigarrow bd$ and $c < d$ so we already know that $bd \oplus c = b \oplus c$
- $bd \oplus b$ is not computed since b is already in bd
- The other attributes are lesser than b

Here, the algorithm performed a single logical closure to compute a single successor. If we now want to compute the successors of the set ce, the algorithm would run as follows :

- $ce \oplus e$ is not computed since e is already in ce
- $ce \oplus d = cd$ is not computed because $c \rightsquigarrow ce$ and $d < e$ so we already know that $ce \oplus d = c \oplus d$
- $ce \oplus c$ is not computed since c is already in ce
- The other attributes are lesser than c

In this case, the set ce has no successor and the algorithm returns the empty set without performing logical closures.

An important point is the case of the pseudo-intents. Indeed, if a set A is a pseudo-intent, then any superset B of A that does not contain A'' is not closed under $\mathcal{B}^-$ and, consequently, is not an element of Φ. As such, A'' is the only upper cover of A. This means that, in Φ, although an attribute set can have any number of successors, pseudo-intents have either one or no successor. Hence, for any pseudo-intent A and set $X \in \Phi$ such that $X \rightsquigarrow A$, we have the following :

- $A \oplus i = A''$ for any attribute $i \in A'' \setminus A$ greater than $\min(A \setminus X)$
- $A \oplus i = X \oplus i$ for any i lesser than $\min(A \setminus X)$

When computing the successors of a pseudo-intent A, the value of $A \oplus i$ is already known for every attribute $i \in A'' \setminus A$ and the value of $A \oplus j$ for $j \notin A''$ is known not to be relevant. Thus, no additional computation is required to obtain the set of successors of a pseudo-intent.

Proposition 6 *If $A \rightsquigarrow B$, $i \notin \bigcup\{o' \mid o \in A'\}$ and $\mathcal{B}^-(A \oplus i) \neq \mathcal{A}$, then $B \oplus i$ is either $A \oplus i$ or $\mathcal{A}$.*

Proof If $i \notin \bigcup\{o' \mid o \in A'\}$, then $(A \oplus i)'' = \mathcal{A}$. Let us suppose that $B \neq A \oplus i$ is a successor of A. If $i < \min(B \setminus A)$ then $B \oplus i = A \oplus i$. If $i > \min(B \setminus A)$, then $A \oplus i \subseteq B \oplus i$. As such, knowing that the only superset of $A \oplus i$ in Φ is $\mathcal{A}$, we have that $B \oplus i$ is either $A \oplus i$ or $\mathcal{A}$. □

This proposition states that, once an attribute i that never appears together with A in the context is found, it is unnecessary to compute $B \oplus i$ for the successors B of A because the result is either an intent or a pseudo-intent that is equal to $A \oplus i$ and is therefore already found. Obviously, this proposition plays a greater role in reducing the number of logical closures in contexts in which many small sets of attributes never appear together, i.e. very sparse contexts. Empirical results and their analysis can be found in Section 3.4.

In our running example, we know that $bc \rightsquigarrow bcd$ because $bc'' = bcd$ and the attribute d is the least attribute in $bcd \setminus bc$ and is greater than c, the minimal attribute in $bc \setminus b$. Similarly, we know that we do not have $cd \rightsquigarrow bcd$ because $\min(bcd \setminus cd) = b$ is lesser than $\min(cd \setminus c)$ and that $c \oplus b$ is not a superset of c.

As discussed in Section 2.5, the algorithm for computing the logical closure of a set with the best worst-case complexity is LinClosure in $O(|\mathcal{A}| \times |\mathcal{B}|)$. Successors performs, at most, $|\mathcal{A}|$ logical closures so the algorithm is in $O(|\mathcal{A}|^2 \times |\mathcal{B}|)$. Thus, Next and Successors have the same worst-case complexity. Nevertheless, we will see in Section 3.4 that they behave differently in practice.

3.3 Algorithm

Now that we can construct attribute sets from their subsets, we can enumerate the elements of the lattice Φ. Our aim was to be able to enumerate in orders less binding than the lectic order, and the successor relation can be used to achieve that. Since we want it to be possible to enumerate in any order that respects the successor relation, we can only provide a model on which algorithms for different orders can be built and optimized.

3.3.1 General Algorithm

The basic form of the algorithm (see Algorithm 9) is classic. We start with $\emptyset$ and for each element of Φ found, we compute its closure. If it is not closed, it is a pseudo-intent and we add the new implication. Then, we compute its successors. The algorithm ends when it reaches $\mathcal{A}$. Even though it is quite similar to Next Closure, two problems that are easily solved by the lectic order and the Next algorithm reappear here, namely choosing a set to consider and checking whether it is indeed closed under $\mathcal{B}^-$.

Algorithm 9 Enumeration of Pseudo-Intents - Basic Form

```
1: Sets = {∅}
2: Impl = ∅
3: while Sets is not empty do
4:    Pick a set A in Sets
5:    if A is indeed in Φ then
6:       B = A''
7:       if A ≠ B then
8:          Impl = Impl ∪ {A → B}
9:       end if
10:      Compute A ⊕ i for every attribute i and add it to Sets
11:   end if
12: end while
13: return Impl
```

The first problem, choosing a set, is effectively choosing an order of enumeration. As we want to be able to enumerate in any order that respects the inclusion relation, we need a way to make sure that we do not consider a set until after all its subsets have been taken into account. The most obvious solution would be to sort the set of sets to obtain the desired order, which can be changed during the algorithm, even though some orders would require more computations than others. The second problem, checking whether the chosen set is closed under $\mathcal{B}^-$, is linked to the successors relation. When a set A is considered, we know the closure of every subset of A but not necessarily all the subsets of $A \oplus i$. The set $A \oplus i$ can be thought to be a successor of A, and thus an element of Φ, when first computed but a pseudo-intent P such that $A < P < A \oplus i$ can exist, and thus the value of $A \oplus i$ is not necessarily correct. For this reason, we have to complete the logical closure before checking whether $A \oplus i$ is a successor of A and taking it into consideration - which would effectively amount to updating every set with new implications. Algorithm 10 illustrates this.

A brief study of the algorithm for computing logical closures in multiple steps can be found in Section 3.6.

The enumeration algorithm has to consider every element of Φ, compute their closure and, in the case of intents, compute their successors. Given that Successors is in $O(|\mathcal{A}|^2 \times |\mathcal{B}|)$ and computing the closure of a set is in $O(|\mathcal{A}| \times |\mathcal{B}|)$, enumerating pseudo-intents using Successor is in $O(|\Phi| \times |\mathcal{A}|^2 \times |\mathcal{B}|)$. This result holds for any such algorithm in which testing whether a set is in Φ can be done with no computation other than the logical closure.

3.3.2 Enumeration in Order of Increasing Cardinality

For example, we can imagine an algorithm that enumerates in the following order :

- Start with $\emptyset$
- Compute the closure of all the elements of Φ of cardinality n

Algorithm 10 Enumeration of Pseudo-Intents - Basic Form

```
1: Sets = {∅}
2: Impl = ∅
3: while Sets is not empty do
4:    Pick the first set A in Sets
5:    Update the logical closure of A
6:    if A has not changed then
7:       if A is a successor of the set it has been generated from then
8:          B = A''
9:          if A ≠ B then
10:            Impl = Impl ∪ {A → B}
11:         end if
12:         for every attribute i ∉ A greater than min(A) do
13:            Add Impl⁻({a ∈ A | |a < i} ∪ {i}) to Sets while preserving the order
14:         end for
15:      end if
16:   else
17:      Place the new A in Sets according to the chosen order
18:   end if
19: end while
20: return Impl
```

- Generate the successors of all the sets of cardinality n
- Increment the cardinality

With this order, testing whether an attribute set is an element of Φ is easy. Indeed, if we are considering attribute sets of cardinality n, then every pseudo-intent of cardinality lesser than n has been found. As such, we only have to complete the logical closures of sets and, if they do not change, they are elements of Φ. Algorithm 11 enumerates in this order. The SUCCESSORS algorithm is divided in two parts : the $\oplus$s are computed first and the canonicity (successor) test is performed once we are sure the set is an element of Φ. As we will study more in depth in Section 3.5.1, this requires that we keep the sets and the successor relations in memory for some time.

Storing the sets before finishing their logical closure takes space but no additional time. See Section 3.6 for details about the computation of logical closures in multiple steps.

Proposition 7 *Algorithm 11 stops and returns the Duquenne-Guigue basis of the input context.*

Proof When considering a given cardinality n, only successors of sets of cardinality n are constructed. These successors have cardinalities strictly greater than n, the number of attribute is finite and $\mathcal{A}$ has no successor so the algorithm eventually reaches $\mathcal{A}$ and stops.

Every set has a predecessor of lesser cardinality. Let us suppose that an element of Φ of cardinality n has not been found by the algorithm. The set of successors of

Algorithm 11 Enumeration of Pseudo-Intents by order of increasing cardinality

```
1: Sets = {∅}
2: Impl = ∅
3: Card = 0
4: while Card < |A| do
5:   for every A ∈ Sets of cardinality Card do
6:     if A = Impl(A) and A is a successor of its predecessor then
7:       B = A''
8:       if A ≠ B then
9:         Impl = Impl ∪ {A → B}
10:      end if
11:      for every attribute i do
12:        Sets = Sets ∪ Impl⁻({a ∈ A | |a < i} ∪ {i})
13:      end for
14:    else
15:      Sets = Sets ∪ Impl(A)
16:    end if
17:  end for
18: end while
19: return Impl
```

its predecessor has not been computed or it would have been found. The algorithm computes the successors of every set it considers, therefore the predecessor itself has not been found. Following the same reasoning, we eventually find that $\emptyset$ has not been found and this contradicts the fact that the algorithm is initialized with it. We then conclude that every element of Φ is enumerated by the algorithm. A pseudo-intent is closed under $\mathcal{B}^-$ therefore it is an element of Φ. Thus, every pseudo-intent is enumerated by the algorithm. □

3.3.3 Example

Using Algorithm 11 on our running example would produce the following :

The algorithm starts with $\emptyset$. It is closed so we generate its successors.

$$\left|\begin{array}{l} \emptyset \oplus e = \mathbf{e} \\ \emptyset \oplus d = \mathbf{d} \\ \emptyset \oplus c = \mathbf{c} \\ \emptyset \oplus b = \mathbf{b} \\ \emptyset \oplus a = \mathbf{a} \end{array}\right|$$

The set of sets is then $\{a, b, c, d, e\}$. They are all still closed under $Impl^-(.)$. Among them, only a is a pseudo-intent with $a'' = ab$ so we add $a \to ab$ to the basis. We then generate the successors of a, b, c, d and e. The set a being a pseudo-intent, we construct only ab from it. There are no attributes greater than $\min(e) = e$ so e has no successors.

$$\left|\begin{array}{l|l|l} b \oplus e = \mathbf{be} & c \oplus e = \mathbf{ce} & d \oplus e = \mathbf{de} \\ b \oplus d = \mathbf{bd} & c \oplus d = \mathbf{cd} & \\ b \oplus c = \mathbf{bc} & & \end{array}\right|$$

The set of sets is then $\{ab, bc, bd, be, cd, ce, de\}$. They are all still closed under $Impl^-(.)$. Among them, bc, be and cd are pseudo-intents so we add $bc \rightarrow bcd$, $be \rightarrow bde$ and $cd \rightarrow bcd$ to $Impl$. We then generate the successors of ab, bc, bd, ce and de. The set bc being a pseudo-intent, we only construct bcd from it. The set de has no successors for the same reasons as the set e in the previous step and we already know that $c \oplus d = cd$ so $ce \oplus d$ is known and ce has no successors. Similarly, $bd \oplus c$ is known to be bc, which has already been found.

$$\left| \begin{array}{l} ab \oplus e = abde \\ ab \oplus d = \mathbf{abd} \\ ab \oplus c = abcd \end{array} \right| \left| \begin{array}{l} bd \oplus e = \mathbf{bde} \end{array} \right|$$

The set of sets is then $\{abd, bcd, bde, abde, abcd\}$. All the sets with 3 attributes are still closed under $Impl^-(.)$. Among them, abd is a pseudo intent so we add $abd \rightarrow abcde$ to $Impl$. We then generate the successors of abd, bcd and bde. The set abd can only have one successor, $abcde$, but it is the set of all attributes so we do not add it. We already know $bd \oplus c$ so we also know $bde \oplus c$. As such, computing $bde \oplus c$ is not needed and no other attribute is available so bde has no successors.

$$\left| \; bcd \oplus e = \mathbf{bcde} \; \right|$$

The set of sets is then $\{abde, abcd, bcde\}$. The logical closures of the sets $abde$ and $abcd$ are updated to $abcde$ and they are removed. Only $bcde$ has a cardinality of 4 and is still closed under $Impl^-(.)$. It is a pseudo-intent so we add $bcde \rightarrow abcde$ to $Impl$. There are no new intents so we continue with the elements of $Sets$ of cardinality 5. There are no more sets so we stop.

The algorithm stops having produced the following implications :

- $a \rightarrow ab$
- $bc \rightarrow bcd$
- $be \rightarrow bde$
- $cd \rightarrow bcd$
- $abd \rightarrow abcde$
- $bcde \rightarrow abcde$

It has performed 16 logical closures where Next Closure would have performed 16 (see Section 2.4.2). It is thus just as efficient in this particular context. However, as will be presented in Section 3.4, there is a significant number of cases in which using Successors provides an advantage over using Next in terms of number of logical closures computed.

Examples of enumerations in other orders (lectic and reverse lectic) can be found in Section 3.5.2.

3.4 Number of Logical Closures

As we have seen in Section 3.2, the SUCCESSORS algorithm (Algorithm 8) behaves differently than NEXT (Algorithm 1) and usually performs more logical closures. This is counterbalanced by Proposition 5 and Proposition 6 which reduce the amount of logical closures by re-using closures that are already computed when possible. Proposition 5 applies consistently while Proposition 6 is most useful in sparse contexts. This leads us to speculate that SUCCESSORS should equal or outperform NEXT on very sparse contexts. In order to test this theory, we implemented, in Java, both a version of Algorithm 10 and the slightly optimized version of NEXT CLOSURE seen, for example, in [84] and counted the number of logical closures performed by both on randomly generated contexts. We ran the algorithm on randomly generated contexts composed of 50 objects, $nbAtt$ attributes and an average of $avDesc$ attributes per object. The contexts were constructed by randomly associating attributes to objects with a probability of $\frac{avDesc}{nbAtt}$. We ran the algorithms 5000 times for each $nbAtt$ and each $avDesc$ between 1 and $nbAtt-2$. In these tests, we used the increasing cardinality order. However, as the number of logical closures performed by SUCCESSOR do not, in any way, depend on the enumeration order, these results hold for any order. For each context, we computed the ratio

$$\rho = \frac{\text{number of logical closures for us}}{\text{number of logical closures for Next Closure}}$$

3.4.1 Numerical Results

	5	6	7	8	9	10	11	12	13	14	15
1	0.98	0.94	0.89	0.86	0.84	0.82	0.81	0.80	0.79	0.78	0.78
2	1.01	1.01	1.001	0.98	0.95	0.93	0.91	0.88	0.86	0.84	0.82
3	1.002	1.01	1.03	1.03	1.01	0.99	0.97	0.96	0.94	0.93	0.91
4		1.004	1.02	1.04	1.05	1.04	1.02	1.01	0.99	0.97	0.96
5			1.008	1.03	1.05	1.06	1.05	1.04	1.03	0.01	1.00
6				1.01	1.03	1.05	1.07	1.07	1.06	1.05	1.03
7					1.01	1.04	1.06	1.07	1.07	1.07	1.06
8						1.02	1.04	1.06	1.08	1.08	1.07
9							1.03	1.05	1.07	1.08	1.08
10								1.04	1.05	1.07	1.09
11									1.05	1.06	1.08
12										1.06	1.07
13											1.07

Table 3.1: Average ρ for $nbAtt$ between 5 and 15 and $avDesc$ between 1 and $nbAtt-2$

Table 3.1 shows the average ρ for different values of $nbAtt$ and $avDesc$. These empirical results seem to back up our theory as ρ is below 1 for small values of $nbDesc$ that generate sparse contexts. We observe that our algorithms perform significantly less logical closures for $nbDesc = 1$, extremely sparse contexts, but

the ratio quickly increases and Next Closure starts outperforming us when the average number of attribute per object is roughly a third of the total number of attributes. An unpredicted phenomenon is the very slight reduction of the ratio when approaching extremely dense contexts.

Now that we know the differences between our algorithm and Next Closure on average, we can take a look at the variance. Table 3.2 presents these results.

	5	6	7	8	9	10	11	12	13	14	15
1	0.0081	0.0088	0.0082	0.0070	0.0062	0.0060	0.0053	0.0054	0.0052	0.0052	0.0052
2	0.0014	0.0038	0.0047	0.0052	0.0051	0.0048	0.0047	0.0045	0.0042	0.0039	0.0036
3	0.0001	0.0011	0.0023	0.0029	0.0030	0.0029	0.0027	0.0027	0.0026	0.0028	0.0026
4		0.0002	0.0011	0.0018	0.0022	0.0020	0.0020	0.0019	0.0018	0.0018	0.0017
5			0.0005	0.0012	0.0018	0.0017	0.0016	0.0016	0.0015	0.0014	0.0013
6				0.0010	0.0015	0.0018	0.0017	0.0014	0.0013	0.0012	0.0011
7					0.0015	0.0020	0.0020	0.0016	0.0013	0.0011	0.0010
8						0.0024	0.0023	0.0022	0.0016	0.0013	0.0011
9							0.0032	0.0026	0.0022	0.0017	0.0013
10								0.0054	0.0033	0.0026	0.0018
11									0.0090	0.0040	0.0030
12										0.0131	0.0046
13											0.0211

Table 3.2: Variance of ρ for $nbAtt$ between 5 and 15 and $avDesc$ between 1 and $nbAtt - 2$

The variance of ρ is low, which means that the average results in Table 3.1 should hold for most contexts. We observe that, when we increase the average description size, the variance initially decreases until $avDesc$ reaches approximately half of the total number of attributes, at which point it begins to increase again.

The results in the average case and the variance are interesting but extremes are important in computer science. Thus, we noted the highest values of ρ encountered for each $nbAtt$ and $avDesc$. Table 3.3 presents those results. Unsurprisingly, the ratio ρ of the worst case encountered increases with both the number of attributes and the average description size. Even though the variance indicates that most cases are around the mean, this last table shows that there are specific cases in which our algorithm performs significantly (though linearly) more operations.

3.4.2 Curves

Knowing the average values of ρ, its variance and its worst-case should be enough to decide whether our algorithm should be used on a given formal context (provided we are only interested in the number of logical closures). But, in order to get a better understanding of the variations of the performance on different contexts and the effects of Proposition 5 and Proposition 6, we provided some graphical representations. For each couple ($nbAtt$, $avDesc$), we plotted the value of ρ obtained on each of the 5000 randomly generated contexts. Figures 3.2 to 3.9 present these results, both sorted and in the order in which they have been obtained for $nbAtt = 10$ and $avDesc$ between 1 and 8. The red line is at $\rho = 1$ so points under it mean our algorithm outperformed Next Closure (in terms of number of logical closures) on this particular context.

	5	6	7	8	9	10	11	12	13	14	15
1	1.36	1.25	1.19	1.18	1.19	1.14	1.11	1.13	1.08	1.10	1.12
2	1.27	1.37	1.31	1.27	1.21	1.18	1.18	1.08	1.10	1.05	1.04
3	1.20	1.34	1.30	1.30	1.28	1.21	1.17	1.17	1.13	1.10	1.07
4		1.24	1.30	1.37	1.39	1.21	1.20	1.17	1.16	1.13	1.10
5			1.62	1.34	1.34	1.27	1.26	1.25	1.17	1.16	1.14
6				1.32	1.33	1.37	1.30	1.27	1.30	1.19	1.17
7					1.57	1.37	1.46	1.31	1.27	1.32	1.28
8						1.65	1.47	1.38	1.32	1.29	1.25
9							1.79	1.55	1.36	1.35	1.27
10								1.65	1.54	1.61	1.44
11									1.85	1.57	1.49
12										1.75	1.65
13											1.83

Table 3.3: Maximal values of ρ for $nbAtt$ between 5 and 15 and $avDesc$ between 1 and $nbAtt - 2$

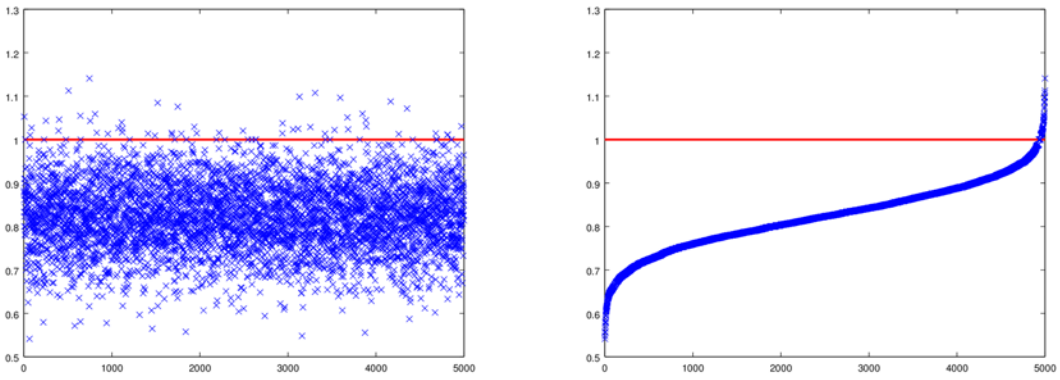


Figure 3.2: $nbAtt = 10$ and $avDesc = 1$

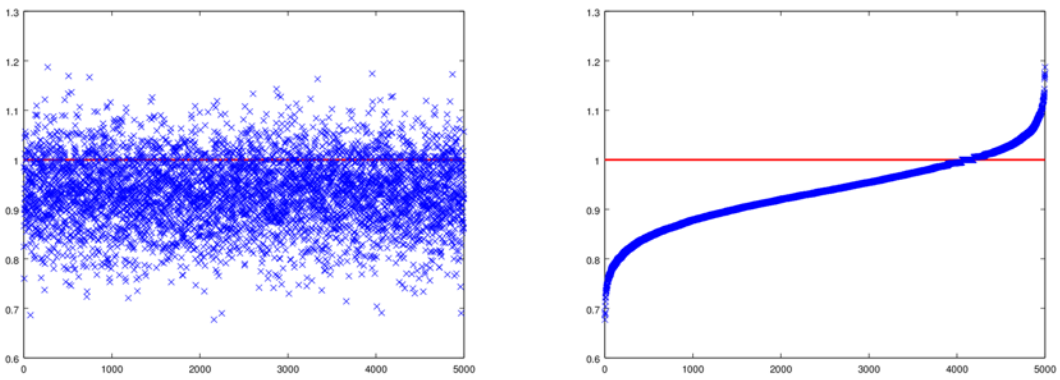


Figure 3.3: $nbAtt = 10$ and $avDesc = 2$

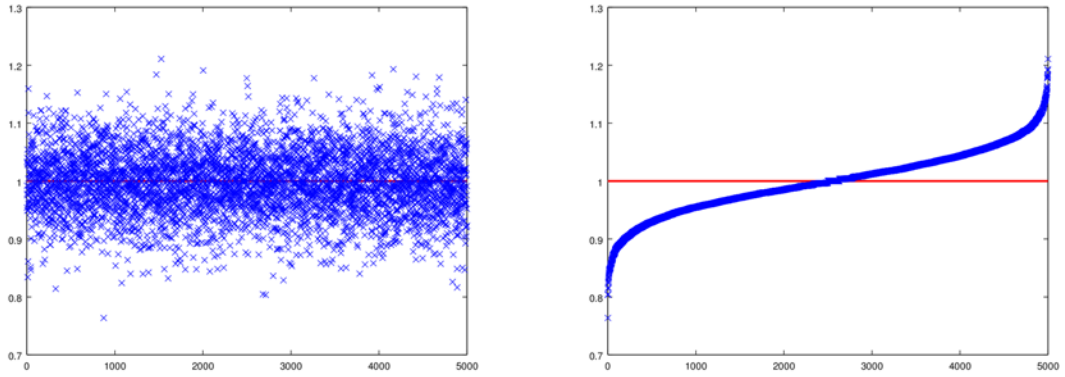


Figure 3.4: $nbAtt = 10$ and $avDesc = 3$

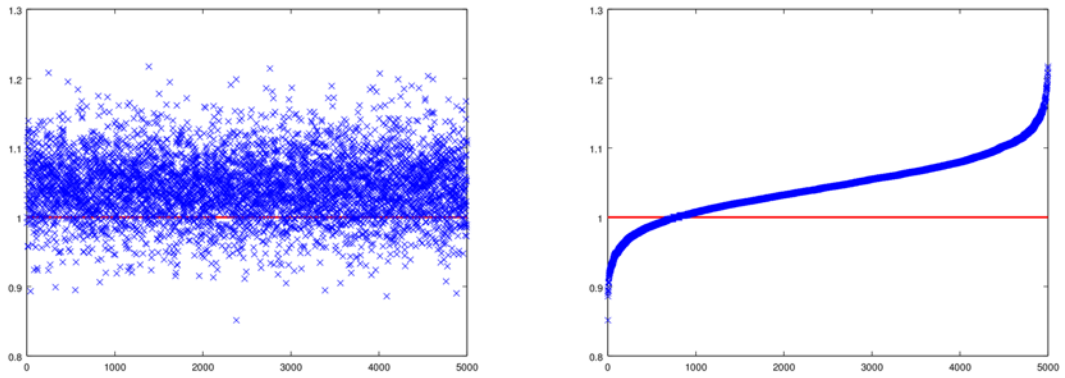


Figure 3.5: $nbAtt = 10$ and $avDesc = 4$

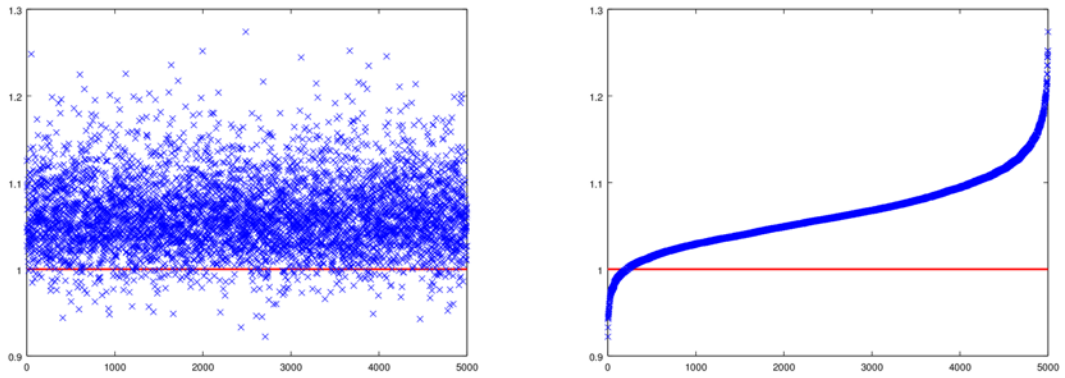


Figure 3.6: $nbAtt = 10$ and $avDesc = 5$

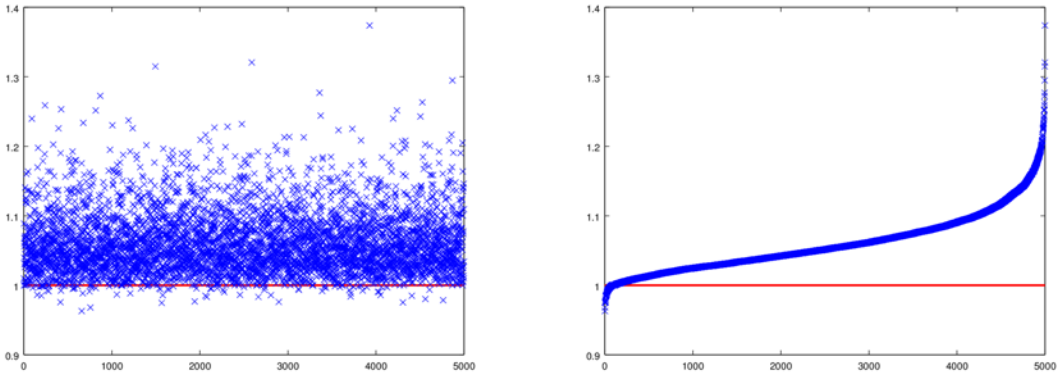


Figure 3.7: $nbAtt = 10$ and $avDesc = 6$

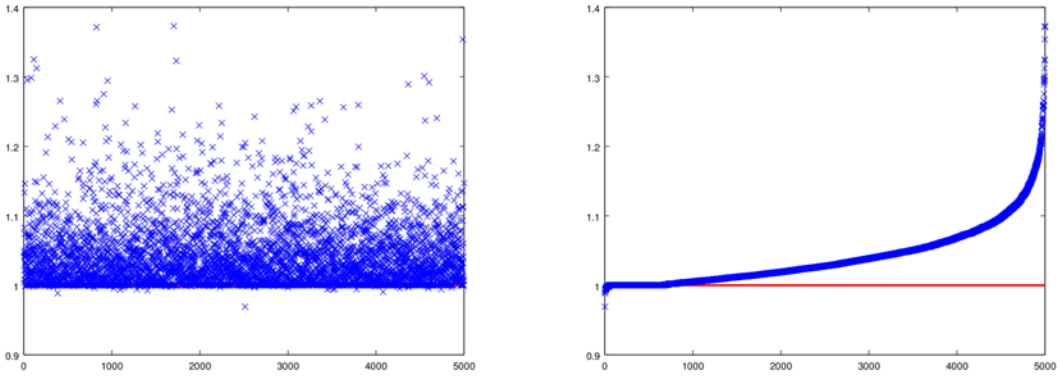


Figure 3.8: $nbAtt = 10$ and $avDesc = 7$

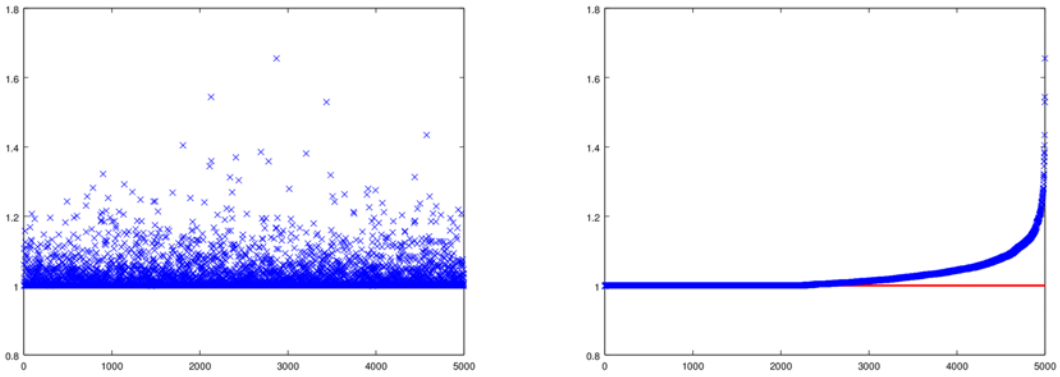


Figure 3.9: $nbAtt = 10$ and $avDesc = 8$

We observe that, as the average description size increases, Next Closure outperforms us on more contexts. This is what the average results showed so this is not a surprise. More interesting is the distribution of the values of ρ. We clearly see that most contexts are very close to the mean with only a handful significantly under or above. This unsurprinsingly corresponds to a normal distribution. However, when

the contexts become dense ($avDesc > 6$), the number of cases under the mean is drastically reduced with no examples below 1 at $avDesc = 8$. We believe this is due to Proposition 6 playing virtually no role when the contexts are extremely dense. This disappearance of the best cases is what causes the reduction of the variance despite the worst cases becoming even worse. We consider that this confirms our hypothesis that Proposition 5 reduces the overall number of logical closures while Proposition 6 only creates interesting cases when the contexts are sparse.

3.5 Orders and Space Complexity

As we have mentioned in Section 3.4, the number of logical closures performed by the enumeration algorithm is the same for all the possible orders (assuming we still want to use Proposition 5 and Proposition 6), and only the space complexity changes. As such, some orders use more space, for seemingly the same result, but there are cases in which using such orders may prove useful.

For example, we know that computing the closure of a set A can be done in a time linear in the size of the context. While this theoretical result always holds, there may be applications in which the context itself is not readily accessible. Scanning the context to compute the closure of a set could cost time or money, and we may want to minimize the number of times we have to access the description of each object. The order presented in Section 3.3.2 considers the elements of Φ in groups of cardinality. Algorithm 11 can thus compute the closure of every attribute set of the same cardinality at the same time by calling the description of each object once and using it for every attribute set. This would require a total number of $|\mathcal{A}|$ scans of the context instead of potentially $2^{|\mathcal{A}|}$. In return, many attribute sets must be stored in memory at the same time. The biggest set of attribute sets of the same cardinality is for $n = \lfloor \frac{|\mathcal{A}|}{2} \rfloor$ with a total of $\frac{|\mathcal{A}|!}{n!(|\mathcal{A}|-n)!}$ possible sets. When computing their successors, we need to know the value of $A \oplus i$ for every attribute i and every set A of cardinality n so, even though the memory used varies throughout the execution, we must be able to store a maximum of $\frac{|\mathcal{A}|!}{n!(|\mathcal{A}|-n)!}|\mathcal{A}|$ sets. This is not surprising given that this algorithm corresponds to a breadth-first search and thus requires more space.

To study the effect of the order on the number of attribute sets stored in memory at the same time, we implemented Algorithm 10 that allows us to choose the order of enumeration by sorting the set of attribute sets in any specified order. We chose to consider 3 possibilities :

- Increasing cardinality order
- Lectic order
- Reverse lectic order

The increasing cardinality order is the one used in Algorithm 11. Attribute sets are treated one by one and not as groups of the same cardinality. While the algorithm could be optimized for each order (most notably, the number of attribute sets stored could have been reduced), this allowed us to empirically compare the effects of the orders themselves on the space complexity. We ran the algorithm

on randomly generated contexts composed of 50 objects, *nbAtt* attributes and an average of *avDesc* attributes per object. The contexts were constructed by randomly associating attributes to objects with a probability of $\frac{avDesc}{nbAtt}$. With each order, we ran the algorithm 5000 times for each *nbAtt* and each *avDesc* between 1 and $nbAtt - 2$. We used the size of the *Sets* set of attribute sets as a measure of the space consumption. We studied the average number of sets, the average maximal number of sets and the maximal maximal number of sets as well as the evolution of the number of sets over a single run of the algorithm.

3.5.1 Results for Each Order

Increasing Cardinality

We started with the increasing cardinality order. Table 3.4 shows the average number of sets simultaneously stored during the course of the algorithm for different values of *nbAtt* and *avDesc*.

	5	6	7	8	9	10	11	12	13	14	15
1	6.7	10.5	14.5	19.1	24.2	29.7	35.4	41.7	48.3	55.0	62.0
2	7.1	13.7	22.1	31.4	40.1	48.3	56.0	64.0	72.3	80.9	89.8
3	7.0	14.3	26.8	43.3	61.3	80.3	98.2	115	129	140	152
4		14.0	27.8	52.2	84.1	118	151	183	216	245	271
5			27.2	53.6	100	160	225	283	336	386	437
6				52.1	102	189	301	418	522	611	689
7					99.5	195	354	554	760	944	1091
8						189	369	655	1003	1353	1654
9							360	698	1206	1796	2369
10								678	1302	2196	3189
11									1265	2399	3949
12										2335	4371
13											4240

Table 3.4: Average number of attribute sets simultaneously stored in memory when enumerating by order of increasing cardinality

As expected, the average size of the set of attribute sets increases with the number of possible attributes. For a given number of attributes, the size tends to increase, at least initially, with the average size of the descriptions. However, it appears important to note that the size starts decreasing once the context becomes saturated enough. We believe that this is due to the fact that fewer implications are valid in those extremely dense contexts, which increases the size of Φ. When most attribute sets are closed under $\mathcal{B}^-$, the cardinality difference between new sets and the one that generated them is smaller and, as a result, they are used more quickly. Table 3.5 shows the average maximal number of sets in memory along with the maximal number found during the 5000 runs of the algorithm.

The maximal number of attribute sets once again increases with the number of attributes and the size of the descriptions. Figure 3.11 shows the evolution of the

*avDesc**nbAtt*	5	6	7	8	9	10	11	12	13	14	15
1	12—14	18—25	25—39	33—54	41—61	50—77	60—83	70—102	81—115	92—127	103—141
2	12—14	22—26	35—46	50—75	64—104	76—120	88—151	99—169	110—203	122—200	134—244
3	12—14	22—24	38—49	62—93	88—152	116—186	142—249	168—334	189—352	207—388	223—380
4		22—25	39—49	73—92	121—171	171—262	219—351	263—511	308—592	349—678	385—770
5			39—55	76—93	142—186	235—315	337—509	430—686	514—900	590—1077	666—1262
6				76—96	145—180	264—345	439—600	637—938	820—1291	978—1646	1113—1972
7					146—191	270—350	496—657	826—1188	1185—1737	1522—2456	1800—2956
8						271—360	505—672	944—1225	1529—2322	2145—3304	2690—4198
9							498—701	982—1361	1791—2529	2809—3905	3847—5900
10								959—1362	1883—2580	3293—4716	4995—7680
11									1829—2616	3478—4975	5899—8891
12										3378—5208	6317—9680
13											6061—9927

Table 3.5: Average maximal and maximal number of attribute sets simultaneously stored in memory when enumerating by order of increasing cardinality

number of attribute sets during the execution of the algorithm for *nbAtt* between 9 and 15 for $avDesc = 4$ and $avDesc = 6$.

The number of attributes steadily increases then stabilizes. We suppose that the algorithm generates sets increasingly faster while the cardinality increases and, once it reaches the cardinality for which the lattice is the widest, many sets of the second half of the lattice have already been constructed and the destruction rate catches up with the construction rate.

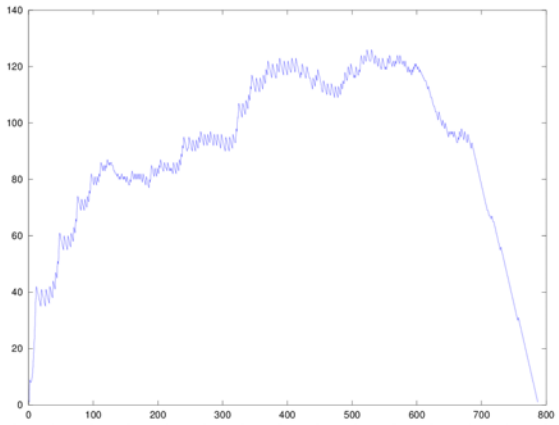
Lectic Order

We then looked at the lectic order. While enumerating in lectic order could be done with Next Closure instead, this order has some interesting effects on the space complexity. Table 3.6 shows the average number of sets simultaneously stored during the course of the enumeration in lectic order for different values of *nbAtt* and *avDesc*.

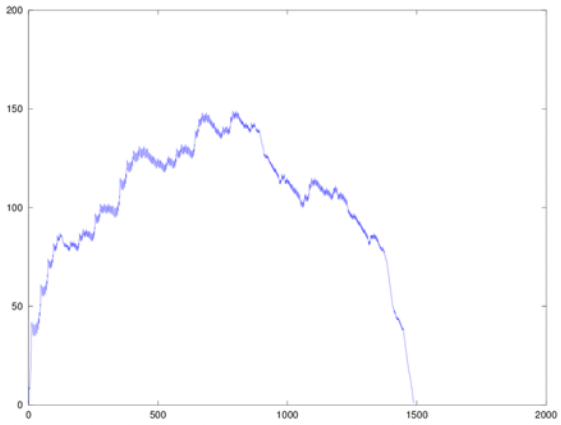
*avDesc**nbAtt*	5	6	7	8	9	10	11	12	13	14	15
1	3.4	5.0	6.6	8.3	10.1	12.3	14.6	17.1	20.0	22.9	25.8
2	3.9	6.9	10.9	15.0	18.4	21.3	23.6	25.7	27.7	29.9	32.3
3	3.9	7.1	13.5	21.4	29.2	36.9	43.8	50.5	55.3	58.3	62.1
4		7.0	13.8	26.0	41.5	57.2	71.2	83.7	96.3	108	117
5			13.2	26.4	49.7	79.3	109.7	136.1	159	179	200
6				25.6	51.1	94.0	148.2	203.1	253	294	330
7					50.0	97.7	176.2	272.5	370	458	530
8						96.7	186.8	326.2	491	656	802
9							187.9	354.3	599	871	1145
10								358.8	667	1085	1536
11									691	1238	1947
12										1306	2265
13											2419

Table 3.6: Average number of attribute sets simultaneously stored when enumerating in lectic order

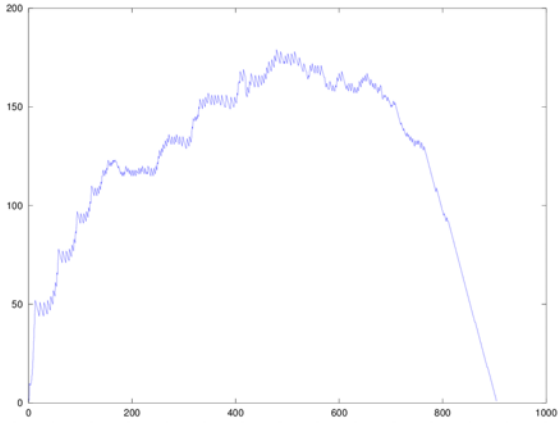
Expectedly, the evolution of the average number of attribute sets during the enumeration in lectic order follows the same rules as in the increasing cardinality



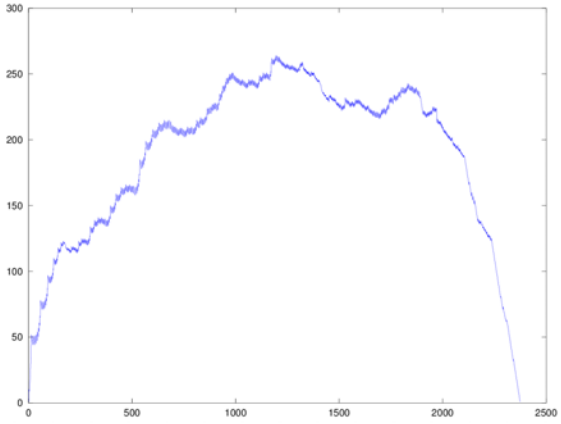
(a) $nbAtt = 9$ and $avDesc = 4$



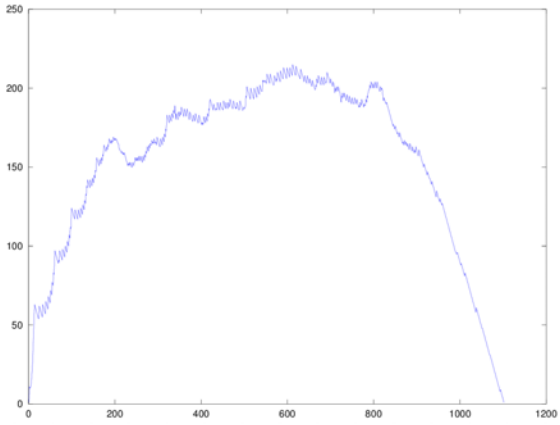
(b) $nbAtt = 9$ and $avDesc = 6$



(c) $nbAtt = 10$ and $avDesc = 4$

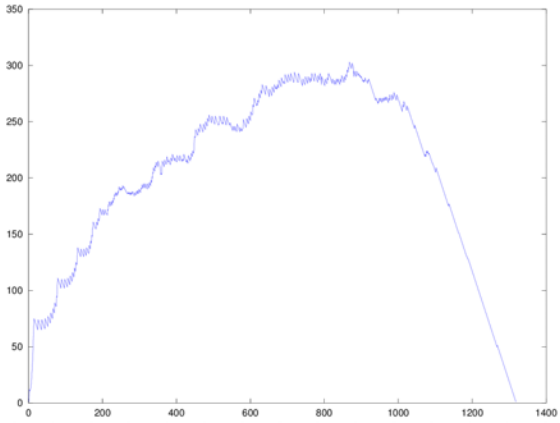


(d) $nbAtt = 10$ and $avDesc = 6$

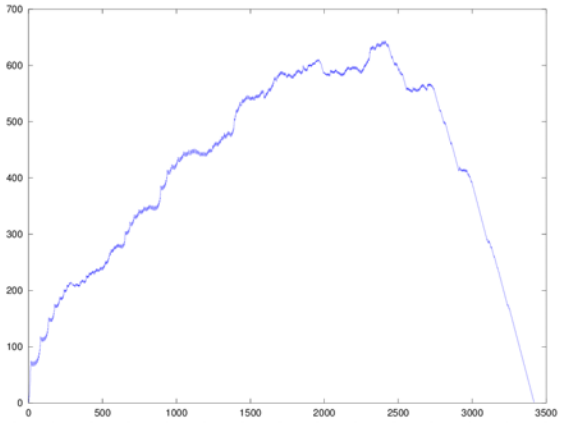


(e) $nbAtt = 11$ and $avDesc = 4$

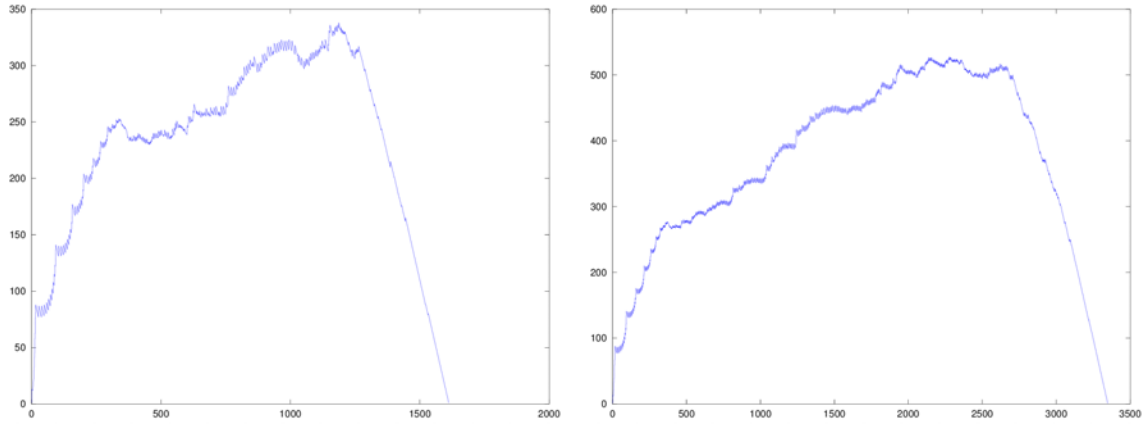
(f) $nbAtt = 11$ and $avDesc = 6$



(g) $nbAtt = 12$ and $avDesc = 4$

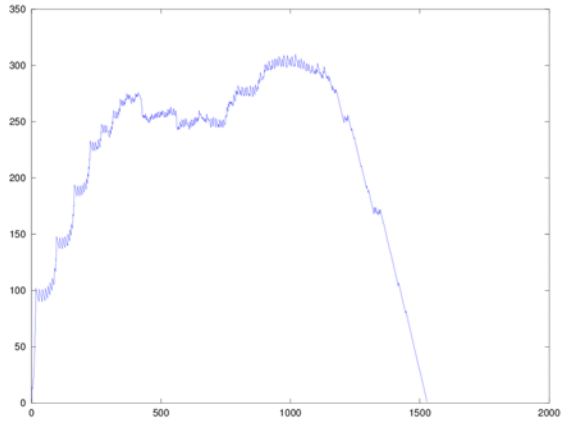


(h) $nbAtt = 12$ and $avDesc = 6$

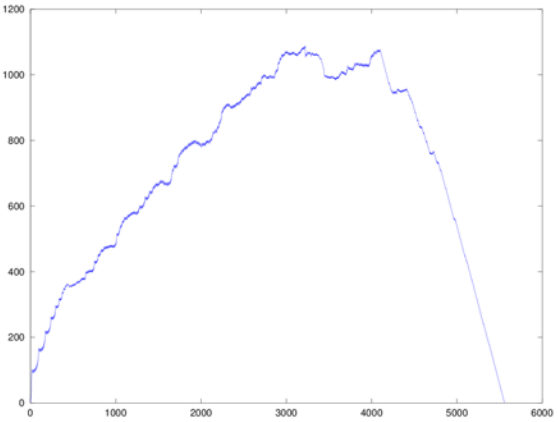


(a) $nbAtt = 13$ and $avDesc = 4$

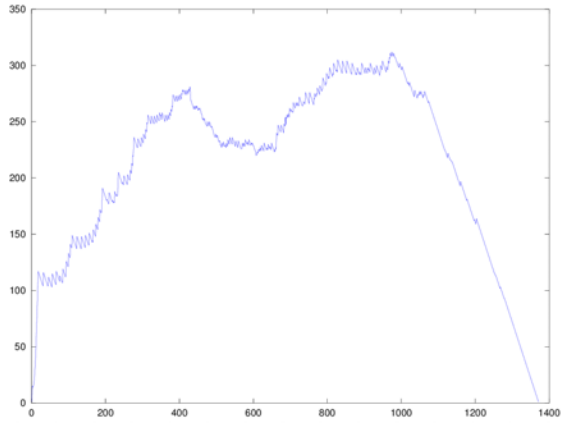
(b) $nbAtt = 13$ and $avDesc = 6$



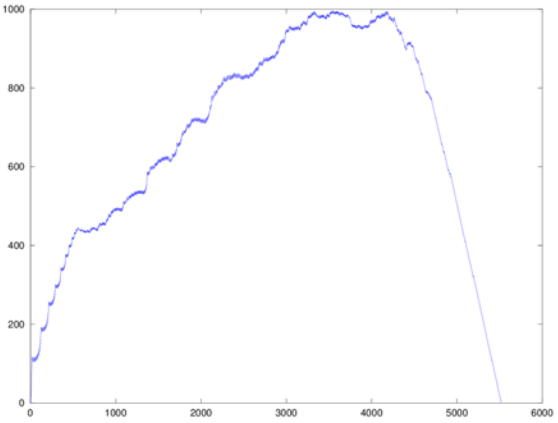
(c) $nbAtt = 14$ and $avDesc = 4$



(d) $nbAtt = 14$ and $avDesc = 6$



(e) $nbAtt = 15$ and $avDesc = 4$



(f) $nbAtt = 15$ and $avDesc = 6$

Figure 3.11: Evolution of the number of attribute sets during the enumeration in increasing cardinality order

order as it increases with both the number of attributes and the size of the descriptions. Table 3.7 shows the average maximal number of sets in memory along with the maximal number found during the 5000 runs of the enumeration in lectic order.

*avDesc**nbAtt*	5	6	7	8	9	10	11	12	13	14	15
1	7—11	10—21	13—37	17—42	21—46	25—64	30—77	36—90	42—99	48—133	54—154
2	7—11	14—25	23—44	31—67	38—95	44—104	49—136	54—135	58—134	63—159	67—151
3	7—10	16—24	29—50	45—84	60—130	75—154	88—203	100—238	110—289	116—298	124—307
4		26—29	32—52	58—95	88—160	117—235	142—333	165—405	187—415	207—478	225—520
5			32—56	63—99	111—185	167—332	221—430	268—609	308—755	341—819	376—952
6				64—99	123—197	210—372	310—607	405—752	489—1127	557—1221	617—1292
7					126—213	237—405	392—688	563—1084	729—1474	872—1787	985—2004
8						248—436	451—779	720—1280	1007—2045	1275—2656	1508—2959
9							479—882	847—1622	1310—2541	1769—3560	2208—4314
10								910—1776	1572—3284	2343—5167	3088—6124
11									1723—3536	2881—5616	4158—9757
12										3197—6588	5173—11592
13											5818—12813

Table 3.7: Average maximal and maximal number of attribute sets simultaneously stored in memory when enumerating in lectic order

Here, even though the evolution of the maximal number and average maximal number of sets follows the same rules as the previous order, we observe that the difference between the average and the all-time maximum is much greater. Figure 3.13 shows the evolution of the number of attribute sets during the enumeration in lectic order for *nbAtt* between 10 and 15 for $avDesc = 4$ and $avDesc = 6$.

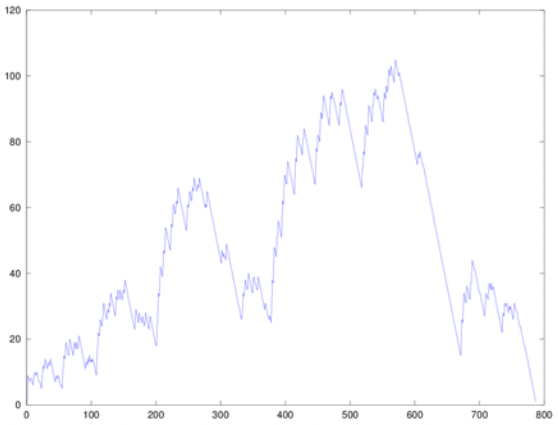
A clear pattern emerges here. The size of *Sets* rapidly increases, stabilizes then decreases slightly multiple times during the computation. We assume that each dip corresponds to an attribute. After a singleton containing the attribute i, the algorithm considers all the elements of Φ for which i is the smallest attribute. Due to the lectic order being used in the successor relation, the new sets are quickly used and the size of *Sets* stabilizes before, ultimately, starting to decrease. As we progress in the computation, the attribute i becomes smaller and more sets begin with i. Thus, this pattern gets bigger.

Reverse Lectic Order

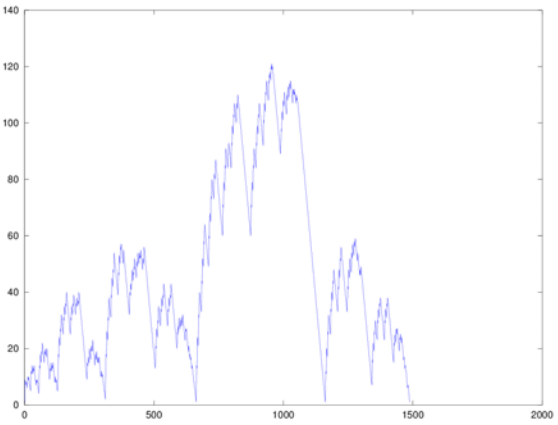
Finally, we looked at the reverse lectic order. We chose it because of its extreme incompatibility with the lectic order used for the generation of the sets, which, we thought, would induce a high spatial complexity. Table 3.8 shows the average number of sets simultaneously stored during the course of the enumeration in reverse lectic order for different values of *nbAtt* and *avDesc*.

Once again, the space required increases with the number of possible attributes and the average size of a description. But it increases faster than for the two previous orders. Table 3.9 shows the maximal and average maximal number of sets for the reverse lectic order.

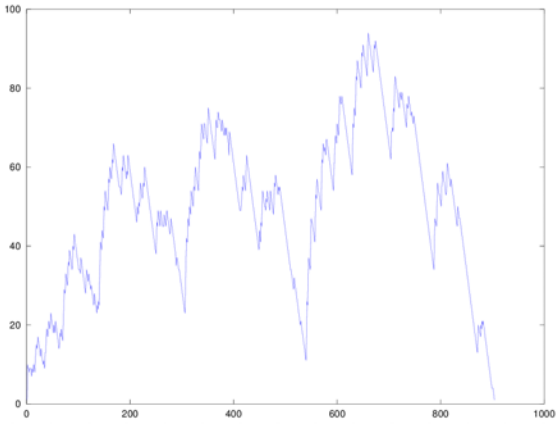
The increase in the average and all-time maximal number of attributes is also much steeper than for the previous orders. Figure 3.15 shows the evolution of the number of attribute sets during the enumeration in reverse lectic order for *nbAtt* between 10 and 15 for $avDesc = 4$ and $avDesc = 6$.



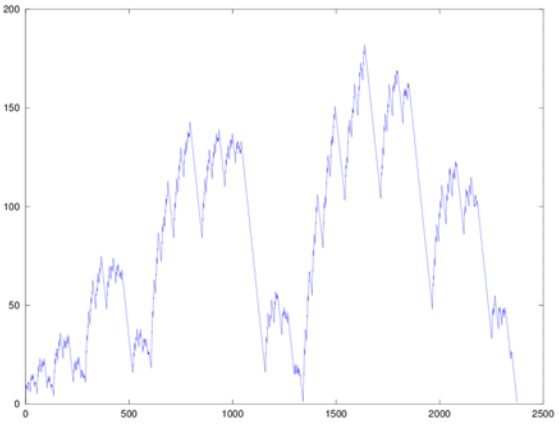
(a) $nbAtt = 9$ and $avDesc = 4$



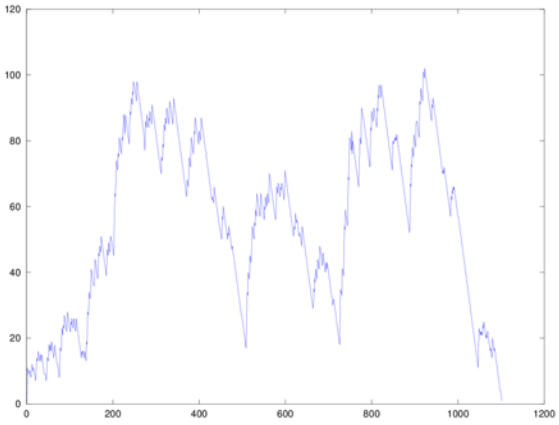
(b) $nbAtt = 9$ and $avDesc = 6$



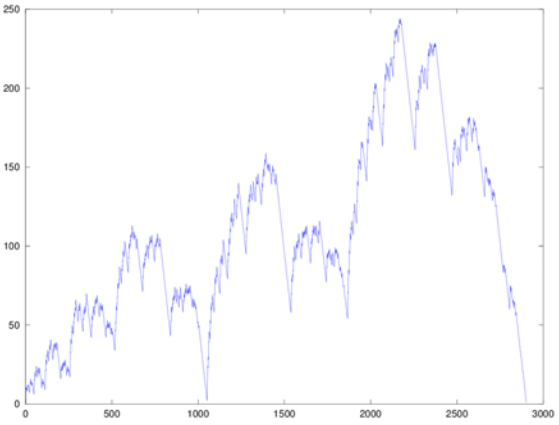
(c) $nbAtt = 10$ and $avDesc = 4$



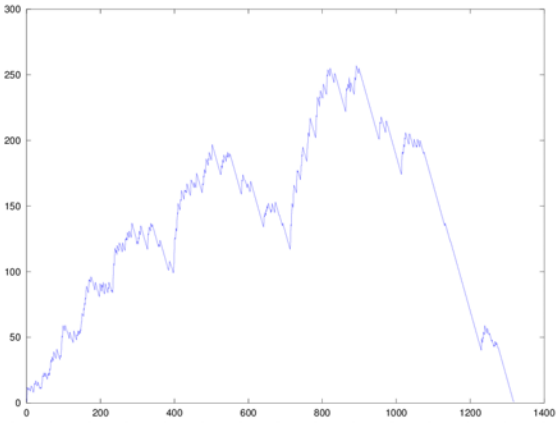
(d) $nbAtt = 10$ and $avDesc = 6$



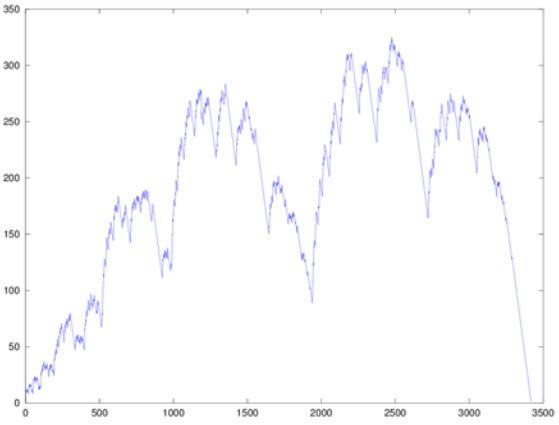
(e) $nbAtt = 11$ and $avDesc = 4$



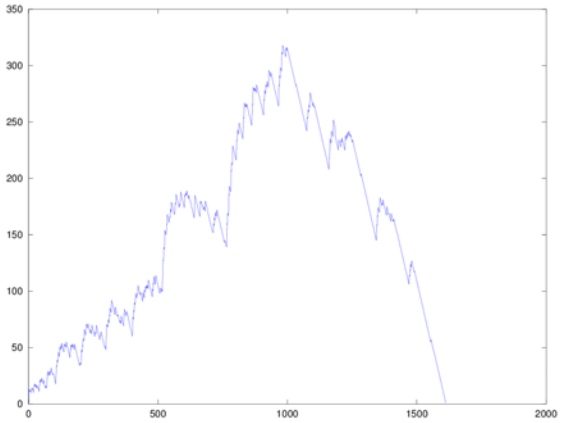
(f) $nbAtt = 11$ and $avDesc = 6$



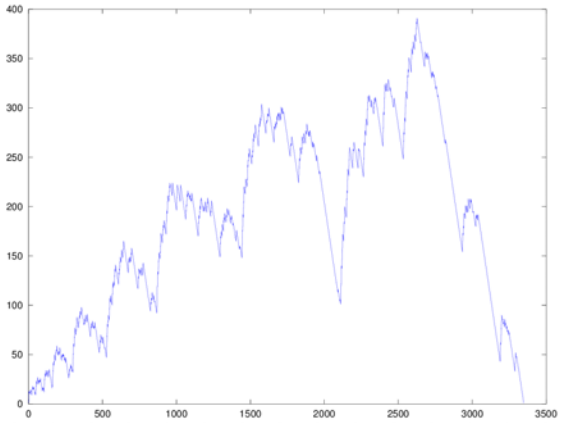
(g) $nbAtt = 12$ and $avDesc = 4$



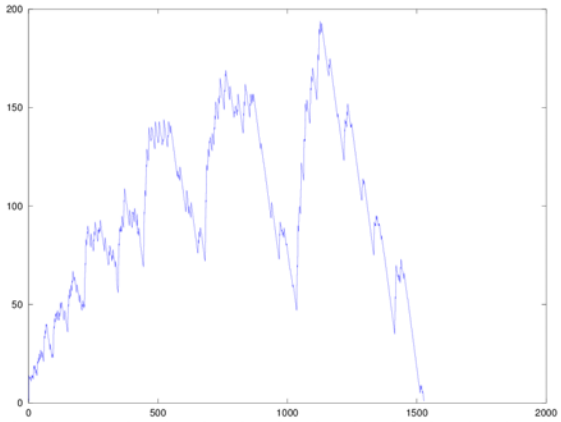
(h) $nbAtt = 12$ and $avDesc = 6$



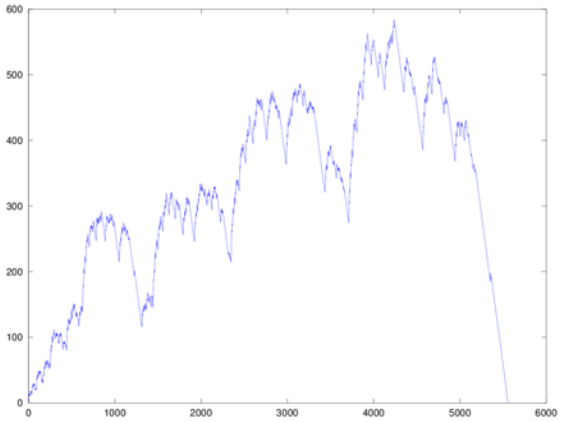
(a) $nbAtt = 13$ and $avDesc = 4$



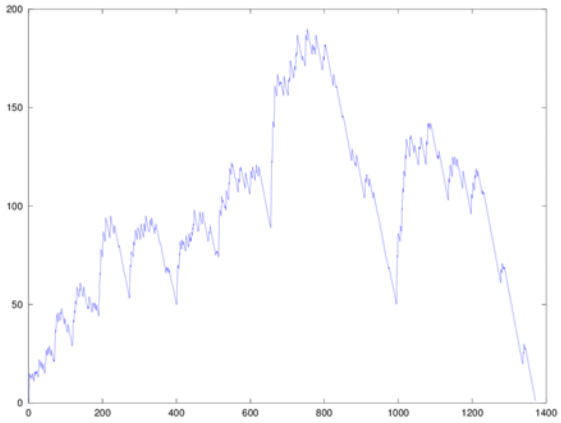
(b) $nbAtt = 13$ and $avDesc = 6$



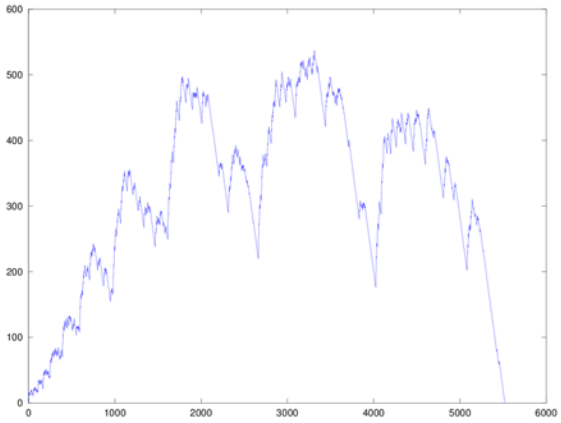
(c) $nbAtt = 14$ and $avDesc = 4$



(d) $nbAtt = 14$ and $avDesc = 6$

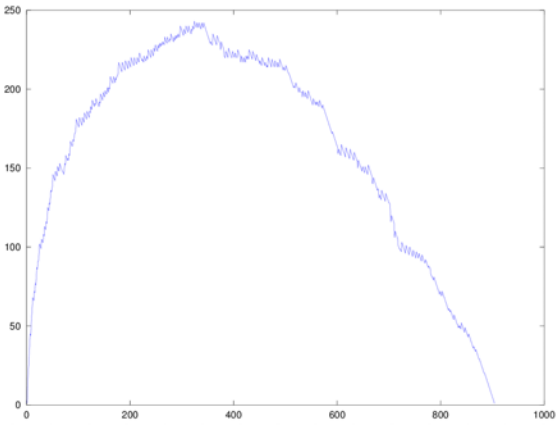


(e) $nbAtt = 15$ and $avDesc = 4$

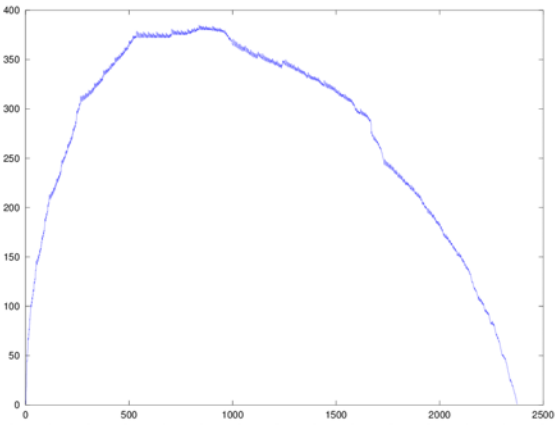


(f) $nbAtt = 15$ and $avDesc = 6$

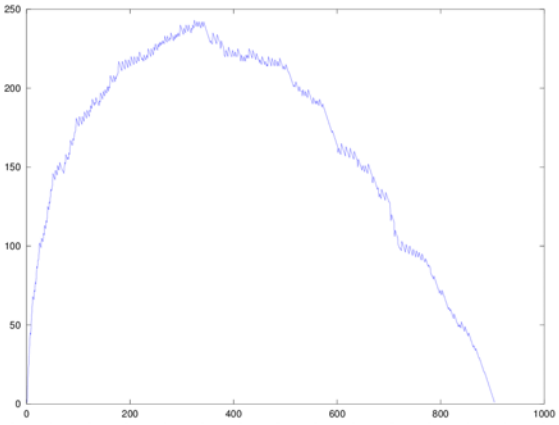
Figure 3.13: Evolution of the number of attribute sets during the enumeration in lectic order



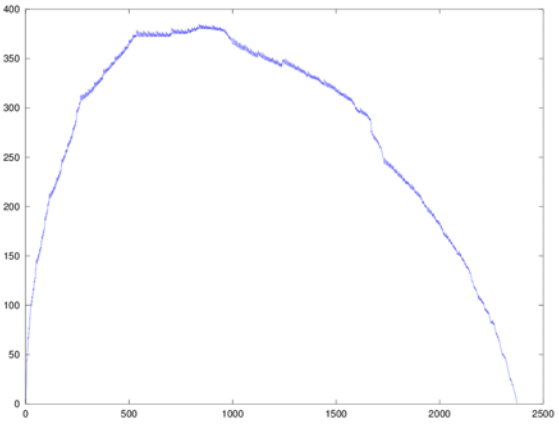
(a) $nbAtt = 9$ and $avDesc = 4$



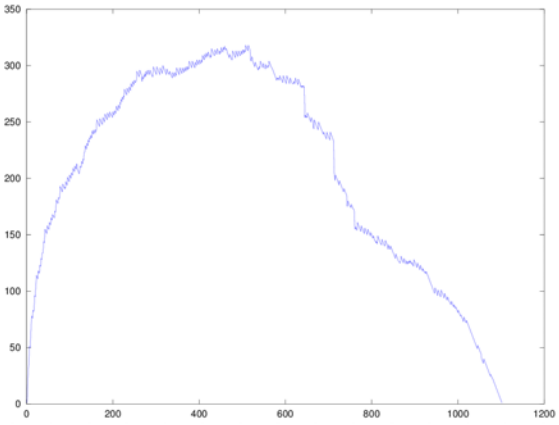
(b) $nbAtt = 9$ and $avDesc = 6$



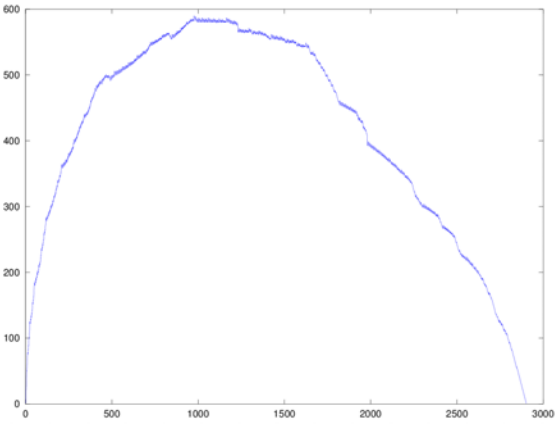
(c) $nbAtt = 10$ and $avDesc = 4$



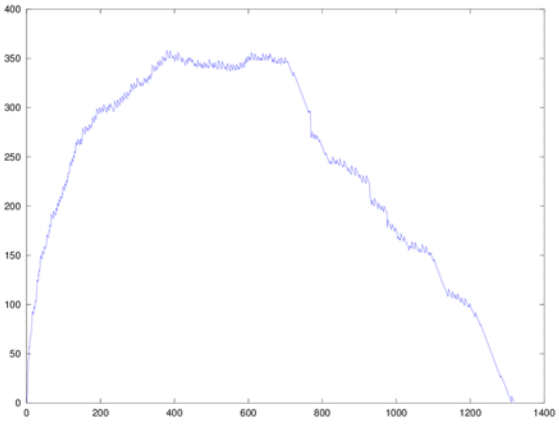
(d) $nbAtt = 10$ and $avDesc = 6$



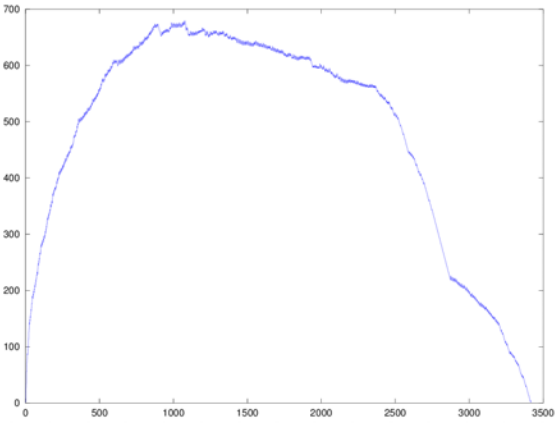
(e) $nbAtt = 11$ and $avDesc = 4$



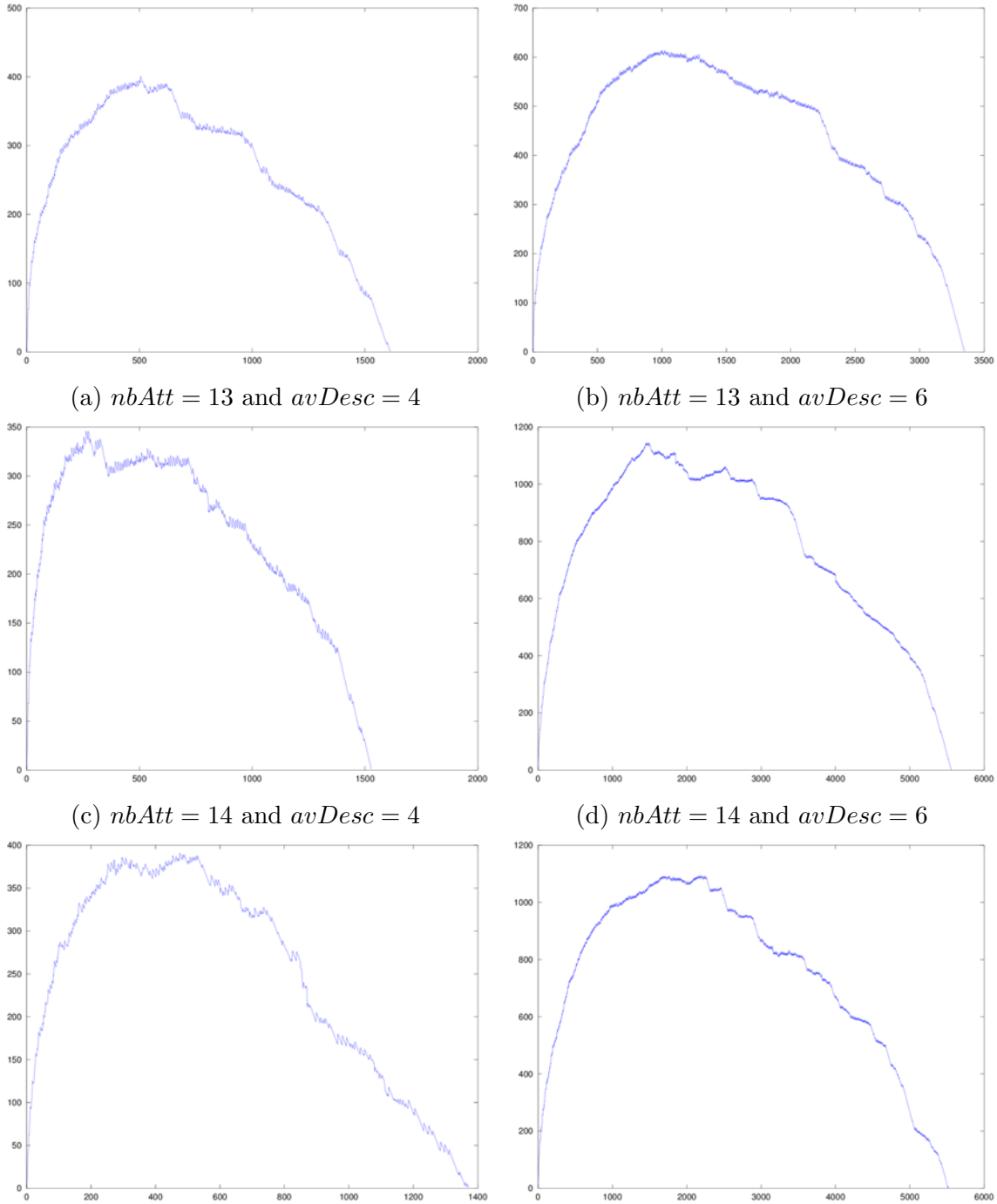
(f) $nbAtt = 11$ and $avDesc = 6$



(g) $nbAtt = 12$ and $avDesc = 4$



(h) $nbAtt = 12$ and $avDesc = 6$



(a) $nbAtt = 13$ and $avDesc = 4$

(b) $nbAtt = 13$ and $avDesc = 6$

(c) $nbAtt = 14$ and $avDesc = 4$

(d) $nbAtt = 14$ and $avDesc = 6$

(e) $nbAtt = 15$ and $avDesc = 4$

(f) $nbAtt = 15$ and $avDesc = 6$

Figure 3.15: Evolution of the number of attribute sets during the enumeration in reverse lectic order

*avDesc**nbAtt*	5	6	7	8	9	10	11	12	13	14	15
1	8.5	12.8	17.2	21.7	26.5	31.4	36.6	42.11	47.6	53.4	59.6
2	10.5	19.0	29.1	39.8	50.1	60.2	69.4	78.8	88.2	96.9	105
3	10.9	21.5	38.6	58.9	79.8	100	120	139	156	171	187
4		21.9	43.0	75.3	113	151	189	224	258	288	317
5			44.3	84.3	143	211	279	342	400	454	508
6				87.4	163	269	388	502	605	697	781
7					170	311	499	699	891	1059	1196
8						329	587	911	1244	1552	1802
9							625	1094	1644	2182	2659
10								1177	2011	2935	3795
11									2184	3648	5179
12										3987	6529
13											7137

Table 3.8: Average number of attribute sets simultaneously stored in memory when enumerating in reverse lectic order

*avDesc**nbAtt*	5	6	7	8	9	10	11	12	13	14	15
1	14—18	22—33	29—52	36—66	44—76	52—92	61—100	70—117	79—132	88—140	98—164
2	17—18	30—35	46—66	63—95	80—138	96—169	111—186	126—219	140—242	154—277	167—306
3	17—18	34—35	58—68	89—129	122—189	155—237	185—291	216—350	243—408	267—460	292—531
4		34—35	65—68	111—133	167—242	226—325	284—423	337—521	389—656	437—787	482—952
5			67—68	124—133	208—262	307—426	410—587	504—828	591—927	673—1121	756—1312
6				130—133	238—262	389—503	560—855	727—1065	880—1375	1018—1674	1144—1817
7					250—262	452—519	719—1032	1004—1457	1279—2042	1524—2394	1728—2738
8						482—519	853—1032	1312—1885	1782—2885	2218—3379	2578—4065
9							917—1032	1592—2057	2371—3724	3123—4904	3792—6074
10								1728—2057	2934—4106	4243—6696	5440—8812
11									3218—4106	5341—7829	7502—12142
12										5903—8203	9581—15628
13											10676—16396

Table 3.9: Average maximal and maximal number of attribute sets simultaneously stored in memory when enumerating in reverse lectic order

The general appearance of the curve is once again very different. In reverse lectic order, the number of attribute sets in the memory skyrockets to its maximum before diminishing slowly.

3.5.2 Comparisons between Orders

In this section, we present a comparison of the three orders based on the data presented for each one. We compared the evolution of the average, average maximal and all-time maximal size of the set of attribute sets when the average size of a description varies. From now on, the increasing cardinality order will be in red, the lectic order in blue and the reverse lectic order in green.

Average Number of Attribute Sets

We started by comparing the evolution of the average number of attribute sets for the three orders. In Figure 3.16, each curve shows the average number of attribute sets for a given *nbAtt* and an *avDesc* between 1 and $nbAtt - 1$.

We observe that the lectic order clearly has the lowest space requirements in

average and the slowest increase of the three orders considered here. The reverse lectic order has the highest space requirements and the fastest increase and the increasing cardinality order is in-between. It is interesting to remark that the curve of the increasing cardinality order starts decreasing slightly when $avDesc$ approaches $nbAtt$ while the curve of the lectic order stabilizes and that of the reverse lectic order keeps increasing.

Average Maximal Number of Attribute Sets

We then made the same comparison for the average maximal number of sets - arguably the most interesting data, as the amount of memory needed ultimately depends on it. In Figure 3.17, each curve shows the average maximal number of attribute sets for a given $nbAtt$ and an $avDesc$ between 1 and $nbAtt - 1$.

Once again, the lectic order seems to be the most efficient with the slowest increase while the reverse lectic order is the least efficient. The increasing cardinality order once again behaves differently as its average maximal number of set starts stabilizing early when $avDesc$ increases while the lectic order keeps increasing longer before finally stabilizing, and the reverse lectic order does not stabilize at all.

Maximal Number of Attribute Sets

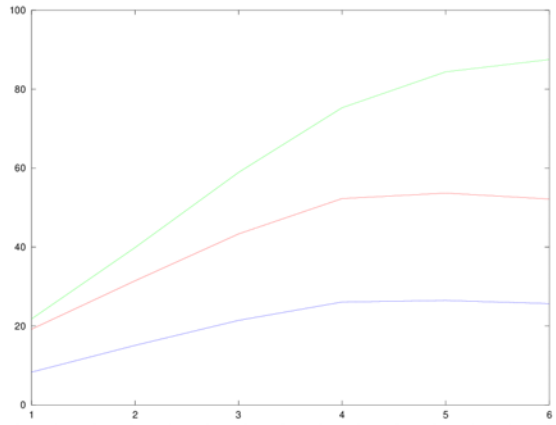
Lastly, we considered the evolution of the maximal number of sets simultaneously stored in memory over the 5000 runs of each algorithm. In Figure 3.18, each curve shows the maximal number of attribute sets found simultaneously for a given $nbAtt$ and an $avDesc$ between 1 and $nbAtt - 1$.

These are, perhaps, the most interesting curves because we clearly see that the lectic order, the order with the best results on average, has specific cases in which it performs worse than the increasing cardinality order. When the contexts become denser, the maximal number of sets found for the lectic order exceed that of the increasing cardinality order. From this, we can deduce that the variance of the maximal space requirements for the enumeration in lectic order increases with the average number of attributes per object and becomes much higher than for the enumeration in increasing cardinality order when the contexts become denser.

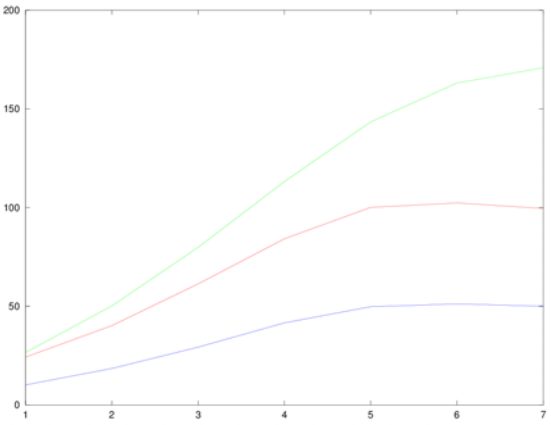
Comparison of the Behaviours of the Orders

To better understand why these orders behave differently, we must analyse the three behaviours on the same contexts. Figure 3.20 shows the evolution of the number of sets in the memory during the execution of the algorithm on a single context for the three orders plotted on the same graph.

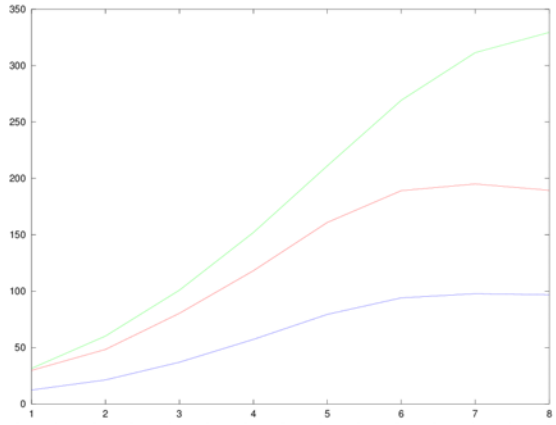
The most obvious difference between the orders is that the lectic order is the only one for which the amount of sets decreases before increasing again in rapid successions. In increasing cardinality order and reverse lectic order, the amount of sets increases roughly once before plummeting. This effect is most clear in the reverse lectic order in which a very high number of sets are generated in the beginning with hardly any set removed. We believe that this is due to the fact that sets B are generated from their lectically greatest subset A. When B is constructed and added, all its subsets $C \not\subseteq A$ are greater than A in reverse lectic order and, thus,



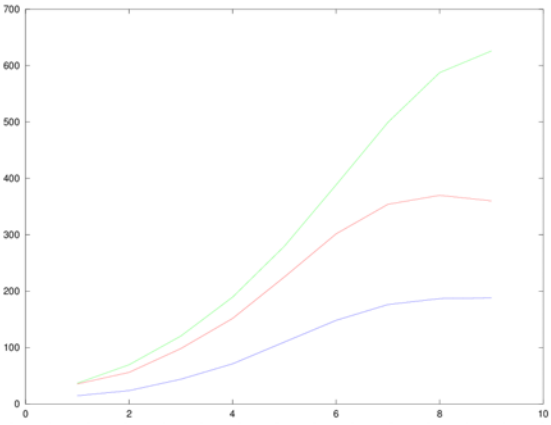
(a) $nbAtt = 8$



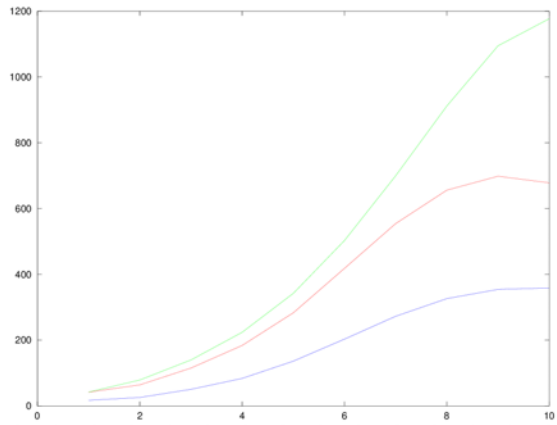
(b) $nbAtt = 9$



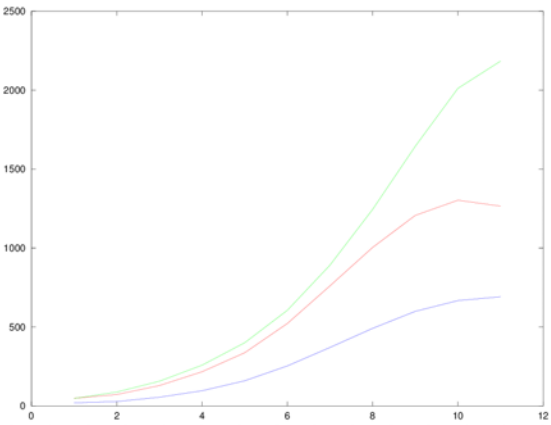
(c) $nbAtt = 10$



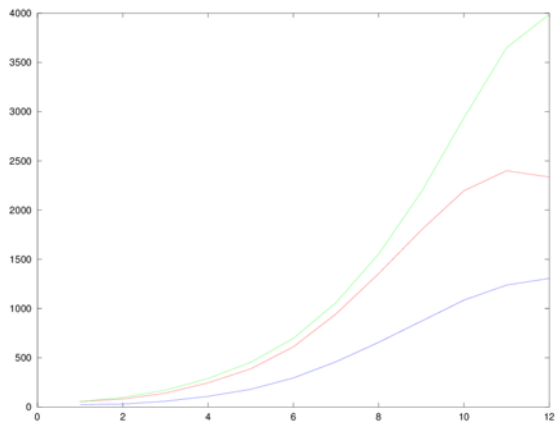
(d) $nbAtt = 11$



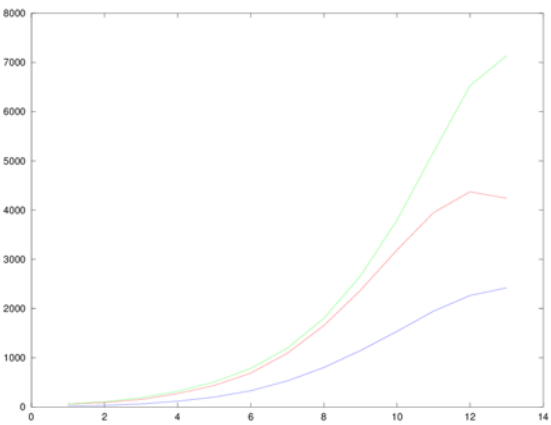
(e) $nbAtt = 12$



(f) $nbAtt = 13$

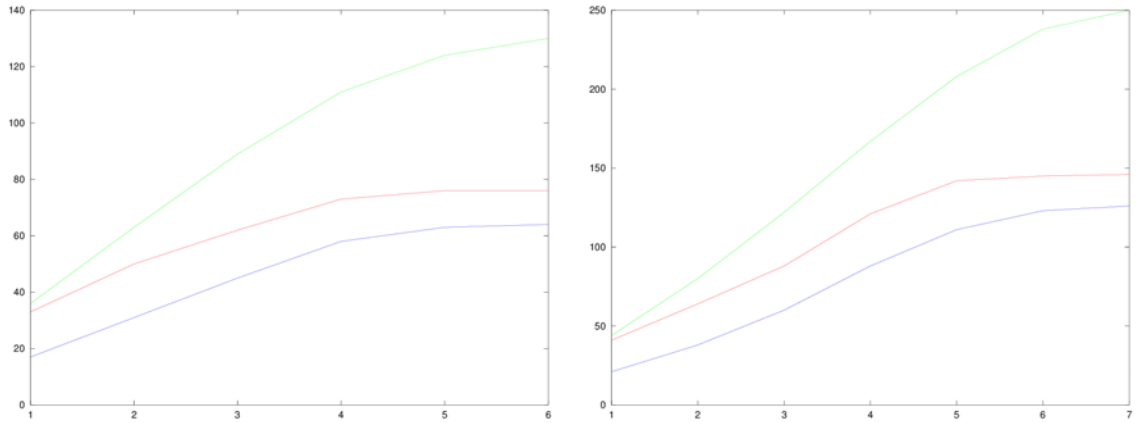


(g) $nbAtt = 14$

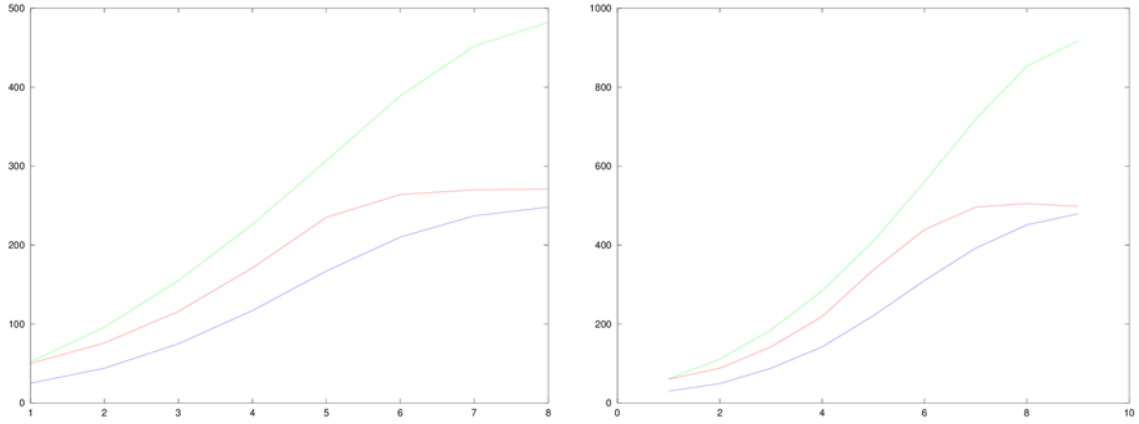


(h) $nbAtt = 15$

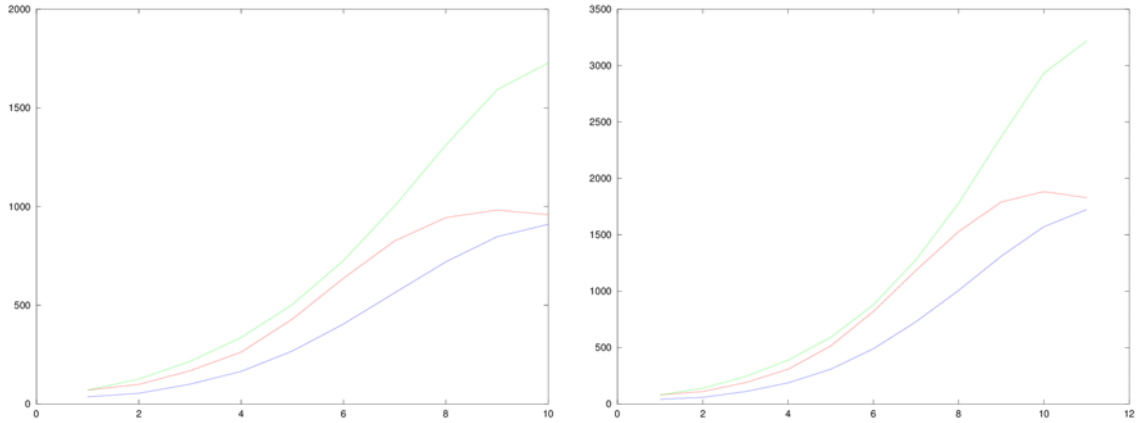
Figure 3.16: Comparison of the average number of attribute sets (Green : Reverse lectic, Red : Increasing cardinality, Blue : Lectic)



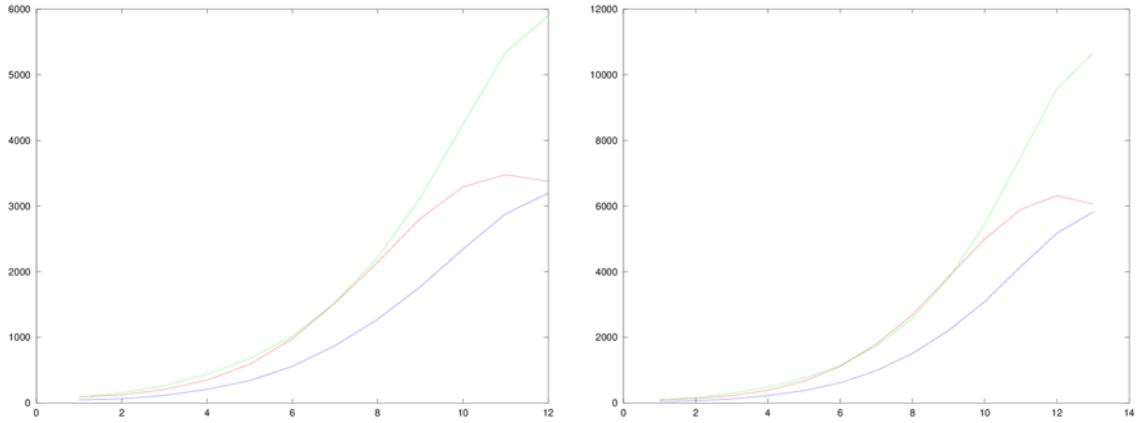
(a) $nbAtt = 8$ (b) $nbAtt = 9$



(c) $nbAtt = 10$ (d) $nbAtt = 11$

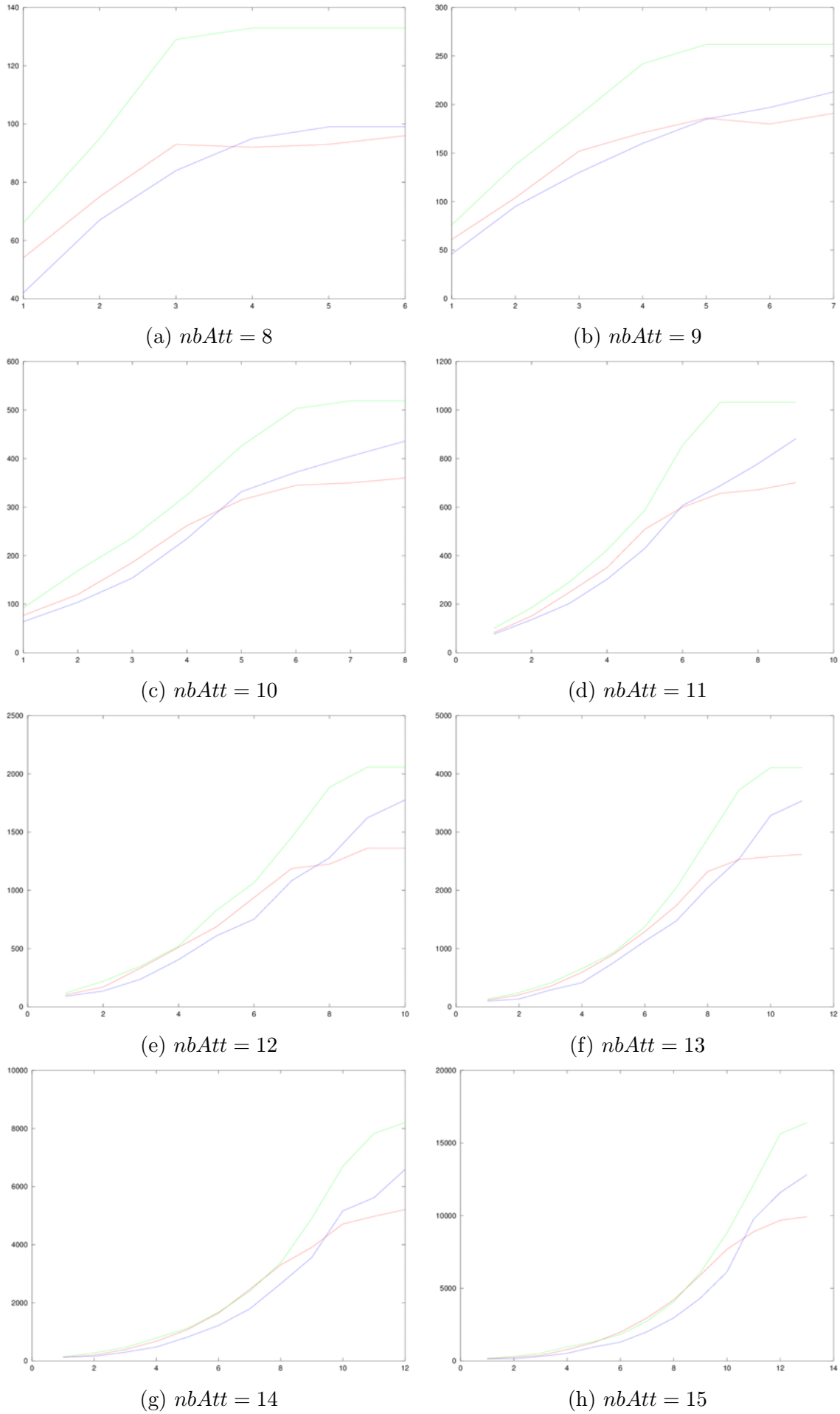


(e) $nbAtt = 12$ (f) $nbAtt = 13$



(g) $nbAtt = 14$ (h) $nbAtt = 15$

Figure 3.17: Comparison of the average maximal number of attribute sets (Green : Reverse lectic, Red : Increasing cardinality, Blue : Lectic)



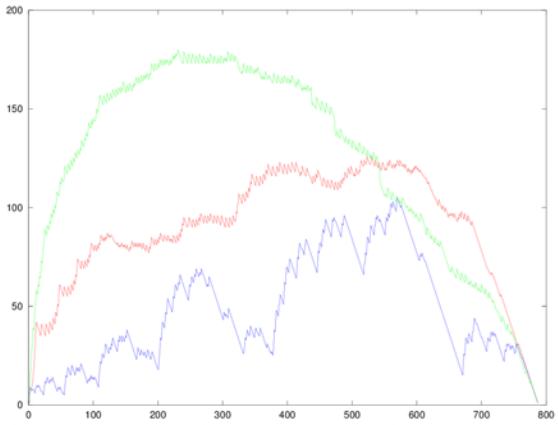
(a) $nbAtt = 8$ (b) $nbAtt = 9$

(c) $nbAtt = 10$ (d) $nbAtt = 11$

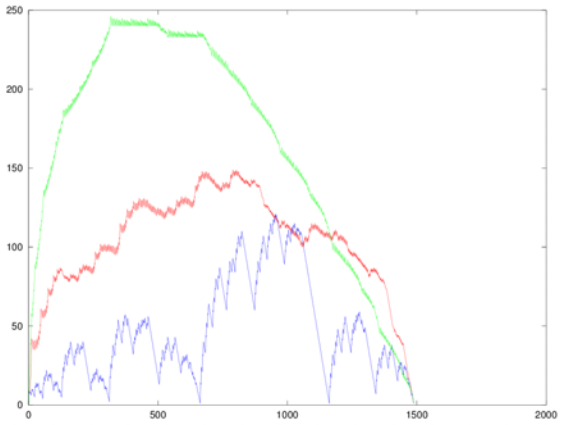
(e) $nbAtt = 12$ (f) $nbAtt = 13$

(g) $nbAtt = 14$ (h) $nbAtt = 15$

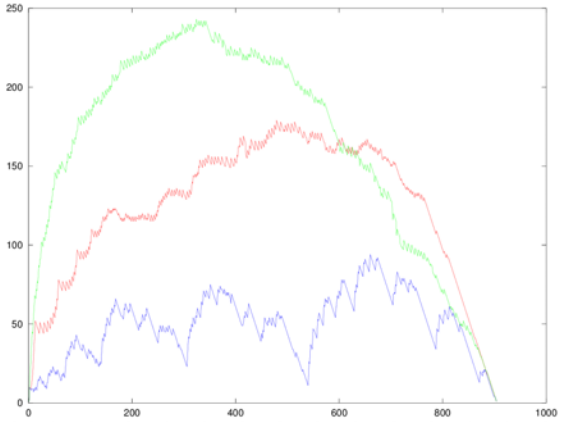
Figure 3.18: Comparison of the worst cases for the maximal number of attribute sets (Green : Reverse lectic, Red : Increasing cardinality, Blue : Lectic)



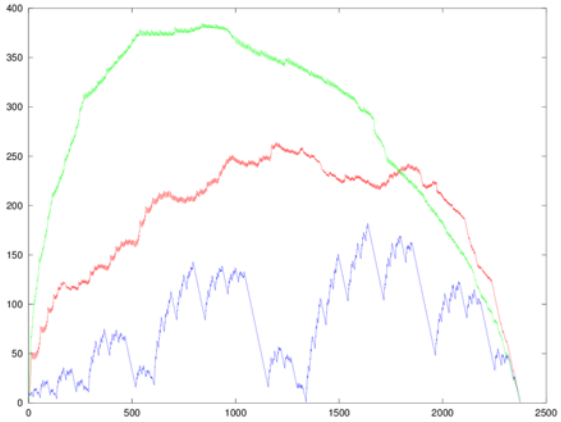
(a) $nbAtt = 9$ and $avDesc = 4$



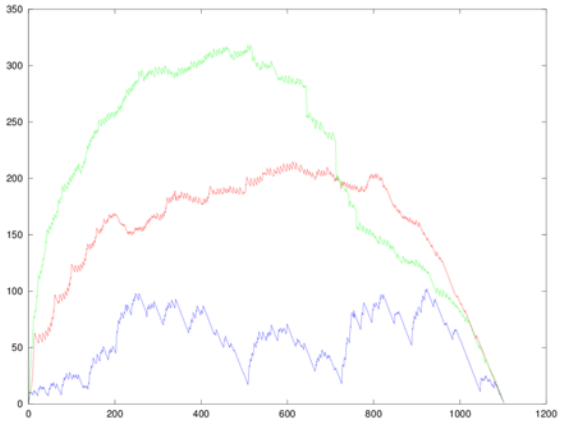
(b) $nbAtt = 9$ and $avDesc = 6$



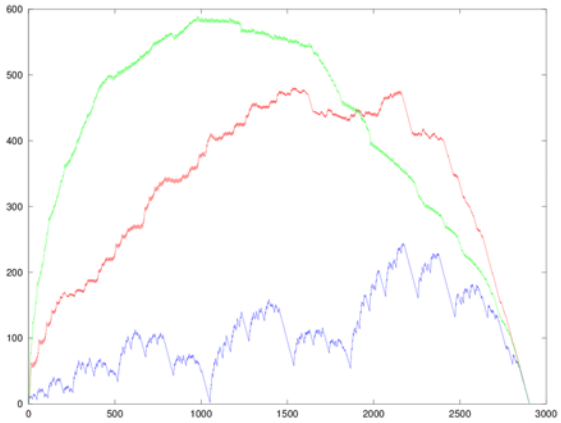
(c) $nbAtt = 10$ and $avDesc = 4$



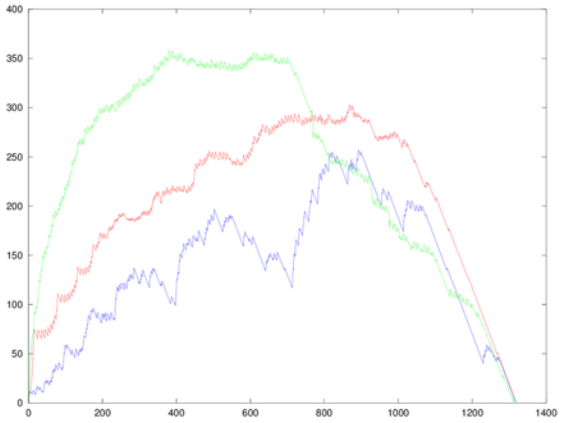
(d) $nbAtt = 10$ and $avDesc = 6$



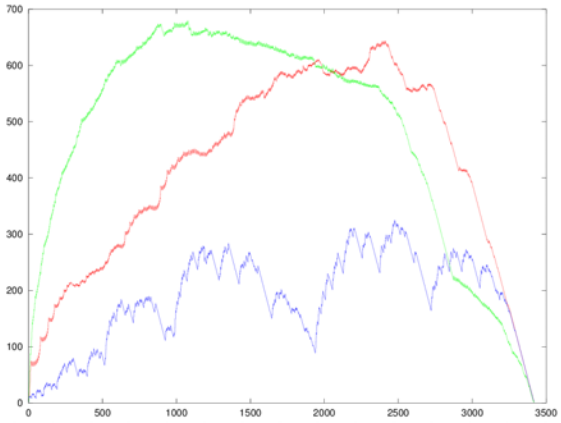
(e) $nbAtt = 11$ and $avDesc = 4$



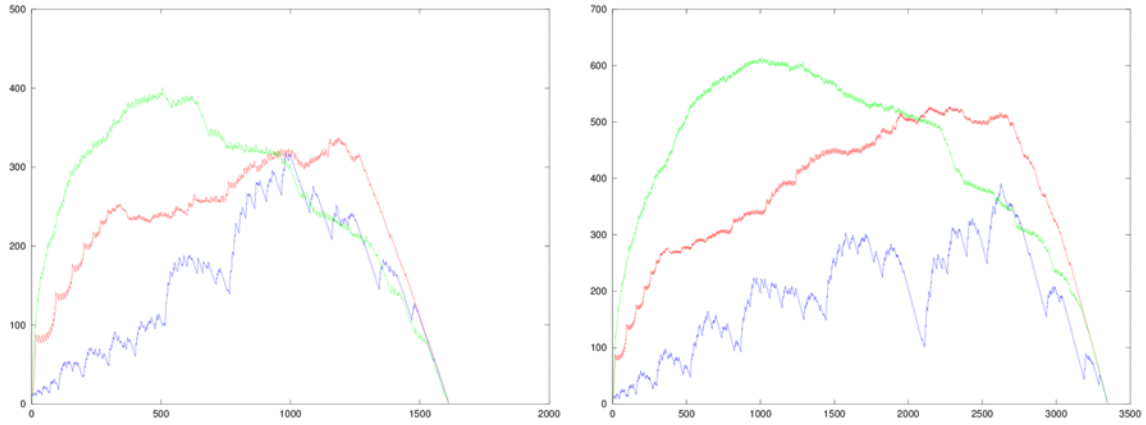
(f) $nbAtt = 11$ and $avDesc = 6$



(g) $nbAtt = 12$ and $avDesc = 4$

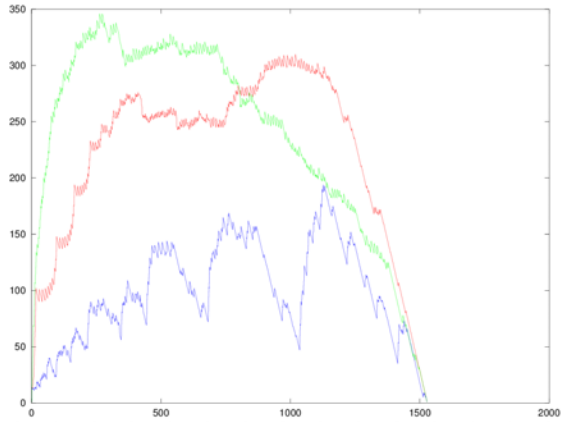


(h) $nbAtt = 12$ and $avDesc = 6$

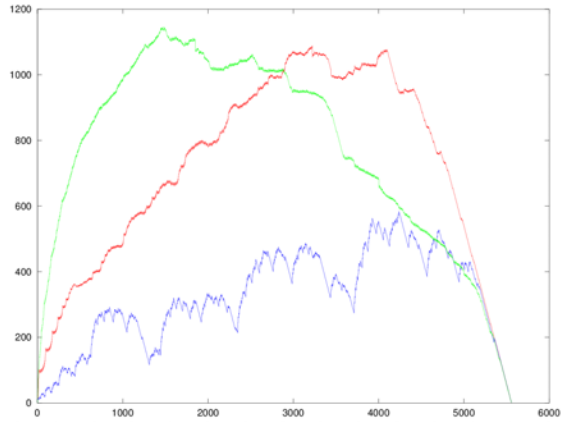


(a) $nbAtt = 13$ and $avDesc = 4$

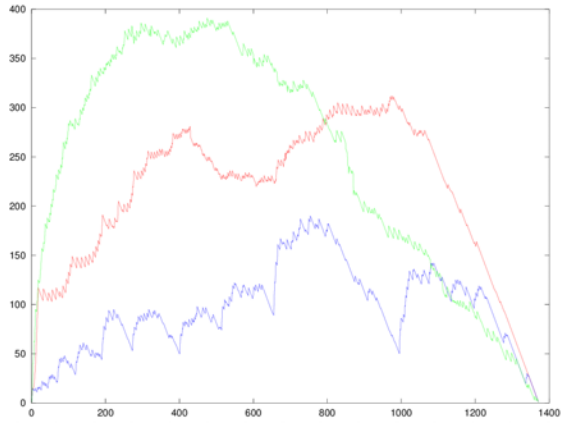
(b) $nbAtt = 13$ and $avDesc = 6$



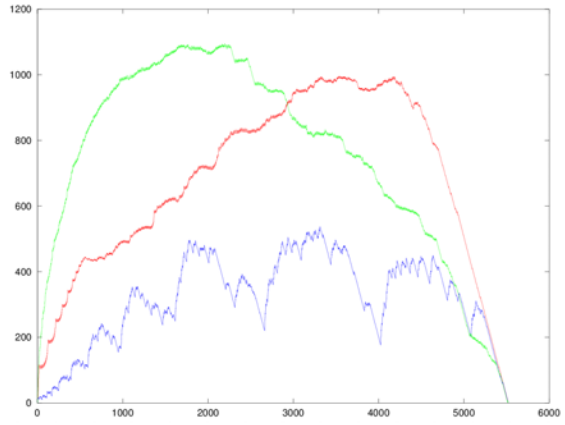
(c) $nbAtt = 14$ and $avDesc = 4$



(d) $nbAtt = 14$ and $avDesc = 6$



(e) $nbAtt = 15$ and $avDesc = 4$



(f) $nbAtt = 15$ and $avDesc = 6$

Figure 3.20: Comparison of the behaviors of the three orders on the same contexts (Green : Reverse lectic, Red : Increasing cardinality, Blue : Lectic)

not yet considered. As such, it is necessary to wait until we have considered every C (and their successors) before we can remove B from the memory. Furthermore, attribute sets with the highest number of successors are the smallest in reverse lectic order. This most likely causes the number of sets to rise quickly until attribute sets that have one or no successor are reached, at which point the amount of sets starts decreasing steadily.

The same effect seems to play a role in the increasing cardinality order. However, this order being closer to the lectic order, sets are treated sooner after their construction which slows the increase. Besides, each cardinality has some attribute sets with a small number of successors. This averages the number of successors and slows the overall increase in attribute sets in memory.

Example of Enumeration in the Lectic Order

To give an even better idea of the behaviours of the different orders, here are the three algorithms on our running example.

First, the lectic order :

$\emptyset'' = \emptyset$.
From $\emptyset$ we construct a, b, c, d and e.

We have e, d, c, b, a. The set e is the smallest and is a successor.
$e'' = e$.
From e we do not construct anything because no attribute is greater than e.

We have d, c, b, a. The set d is the smallest and is a successor.
$d'' = d$.
From d we construct de.

We have de, c, b, a. The set de is the smallest and is a successor.
$de'' = de$.
From de we do not construct anything.

We have c, b, a. The set c is the smallest and is a successor.
$c'' = c$.
From c we construct cd and ce.

We have ce, cd, b, a. The set ce is the smallest and is a successor.
$ce'' = ce$.
From ce we construct cd.

We have cd (from c), cd (from ce), b, a. The set cd (from c) is the smallest and is a successor.
$cd'' = bcd$ so $cd \to bcd$ is added to the basis.
From cd we do not construct anything because $cd \oplus e = bcde$ and b is lesser than c.

We have cd (from ce), b, a. The set cd (from ce) is the smallest but is not a successor.
We have b, a. The set b is the smallest and is a successor.

$b'' = b$
From b we construct bc, bd and be.

We have be, bd, bc, a. The set be is the smallest and is a successor.
$be'' = bde$ so $be \to bde$ is added to the basis.
From be we construct bde.

We have bd, bde, bc, a. The set bd (from b) is the smallest and is a successor.
$bd'' = bd$.
From bd we construct bc, bde.

We have bde (from be), bde (from bd), bc (from b), bc (from bd), a. bde (from be) is the smallest but is not a successor.
We have bde (from bd), bc (from b), bc (from bd), a. bde (from bd) is the smallest and is a successor.
$bde'' = bde$.
From bde we construct bc.

We have bc (from b), bc (from bd), bc (from bde), a. The set bc (from b) is the smallest and is a successor.
$bc'' = bcd$ so $bc \to bcd$ is added to the basis.
From bc we construct bcd.

We have bc (from bd), bc (from bde), bcd, a. The set bc (from bd) is the smallest but is not a successor.
We have bc (from bde), bcd, a. The set bc (from bde) is the smallest but is not a successor.
We have bcd, a. The set bcd is the smallest and is a successor.
$bcd'' = bcd$.
From bcd we construct $bcde$.

We have $bcde$, a. The set $bcde$ is the smallest and is a successor.
$bcde'' = abcde$ so $bcde \to abcde$ is added to the basis.
From $bcde$ we do not construct anything.

We have a. The set a is the smallest and is a successor.
$a'' = ab$ so $a \to ab$ is added to the basis.
From a we construct ab.

We have ab. The set ab is the smallest and is a successor.
$ab'' = ab$.
From ab we construct $abcd$, abd and $abde$.

We have abd, $abde$, $abcd$. The set abd is the smallest and is a successor.
$abd'' = abcde$ so $abd \to abcde$ is added to the basis.
From abd we do not construct anything because $abcde$ is the final set.

We have $abde$ and $abcd$. The logical closure of $abde$ is updated to $abcde$. It is

removed from the list.
We have *abcd*. The logical closure of *abcd* is updated to *abcde*. It is removed from the list.
We have no more sets. The algorithm ends.

The pattern previously described is clearly visible here. The amount of sets in memory is low when we reach singletons and increases fast in-between.

Example of Enumeration in the Reverse Lectic Order

Now, let us enumerate in reverse lectic order on the same running example :

$\emptyset'' = \emptyset$. From $\emptyset$ we construct a, b, c, d and e.
We have a, b, c, d and e. The set a is the smallest and is a successor.
$a'' = ab$ so $a \to ab$ is added to the basis.
From a we construct ab.

We have b, ab, c, d and e. The set b is the smallest and is a successor.
$b'' = b$.
From b we construct bc, bd, be.

We have ab, c, bc, d, bd, e and be. The set ab is the smallest and is a successor.
$ab'' = ab$.
From ab we construct abc, abd and abe.

We have c, bc, abc, d, bd, abd, e, be and abe. The set c is the smallest and is a successor.
$c'' = c$.
From c we construct cd and ce.

We have bc, abc, d, bd, abd, cd, e, be, abe and ce. The set bc is the smallest and is a successor.
$bc'' = bcd$ so $bc \to bcd$ is added to the basis.
From bc we construct bcd.

We have abc, d, bd, abd, cd, bcd, e, be, abe and ce. The logical closure of abc is updated to $abcd$.
We have d, bd, abd, cd, bcd, $abcd$, e, be, abe and ce. The set d is the smallest and is a successor.
$d'' = d$.
From d we construct de.

We have bd, abd, cd, bcd, $abcd$, e, be, abe, ce and de. The set bd is the smallest and is a successor. $bd'' = bd$.
From bd we construct bc and bde.

We have bc (from bd), abd, cd, bcd, $abcd$, e, be, abe, ce, de and bde. The set bc is the smallest but is not a successor.
We have abd, cd, bcd, $abcd$, e, be, abe, ce, de and bde. The set abd is the smallest and is a successor.

$abd'' = abcde$ so $abd \to abcde$ is added to the basis.
From abd we do not construct anything.

We have cd, bcd, $abcd$, e, be, abe, ce, de and bde. The set cd is the smallest and is a successor.
$cd'' = bcd$ so $cd \to bcd$ is added to the basis.
From cd we do not construct anything because bcd has an attribute lesser than c.

We have bcd, $abcd$, e, be, abe, ce, de and bde. The set bcd is the smallest and is a successor.
$bcd'' = bcd$.
From bcd we construct $bcde$.

We have $abcd$, e, be, abe, ce, de, bde and $bcde$. The logical closure of $abcd$ is updated to $abcde$ (and removed).
We have e, be, abe, ce, de, bde and $bcde$. The set e is the smallest and is a successor.
$e'' = e$.
From e we do not construct anything.

We have be, abe, ce, de, bde and $bcde$. The set be is the smallest and is a successor.
$be'' = bde$ so $be \to bde$ is added to the basis.
From be we construct bde.

We have abe, ce, de, bde (from bd), bde (from be) and $bcde$. The logical closure of abe is updated to $abcde$ (and removed).
We have ce, de, bde (from bd), bde (from be) and $bcde$. The set ce is the smallest and is a successor.
$ce'' = ce$.
From ce we construct cd.

We have cd, de, bde (from bd), bde (from be) and $bcde$. The set cd is the smallest but is not a successor.
We have de, bde (from bd), bde (from be) and $bcde$. The set de is the smallest and is a successor.
$de'' = de$
From de we do not construct anything.

We have bde (from bd), bde (from be) and $bcde$. The set cde (from bd) is the smallest and is a successor.
$bde'' = bde$.
From bde we construct bc.

We have bc, bde (from be) and $bcde$. The set bc is the smallest but is not a successor.
We have bde (from be) and $bcde$. The set bde (from be) is the smallest but is not a successor.
We have $bcde$. The set $bcde$ is the smallest and is a successor.
$bcde'' = abcde$ so $bcde \to abcde$ is added to the basis.
From $bcde$ we do not construct anything.

In reverse lectic order, we can observe the number of sets skyrocketting, staying high and plummetting near the end.

Example of Enumeration in the Increasing Cardinality Order

And finally, the enumeration in order of increasing cardinality :

$\emptyset'' = \emptyset$.
From $\emptyset$ we construct a, b, c, d and e.

We have a, b, c, d and e. The set a is one of the smallest and is a successor.
$a'' = ab$ so $a \to ab$ is added to the basis.
From a we construct ab.

We have b, c, d, e and ab. The set b is one of the smallest and is a successor.
$b'' = b$.
From b we construct bc, bd, be.

We have c, d, e, ab, bc, bd and be. The set c is one of the smallest and is a successor.
$c'' = c$.
From c we construct cd and ce.

We have d, e, ab, bc, bd, be, cd and ce. The set d is one of the smallest and is a successor.
$d'' = d$.
From d we construct de.

We have e, ab, bc, bd, be, cd, ce and de. The set e is one of the smallest and is a successor.
$e'' = e$.
From e we do not construct anything.

We have ab, bc, bd, be, cd, ce and de. The set ab is one of the smallest and is a successor.
$ab'' = ab$.
From ab we construct abc, abd and abe.

We have bc, bd, be, cd, ce, de, abc, abd and abe. The set bc is one of the smallest and is a successor.
$bc'' = bcd$ so $bc \to bcd$ is added to the basis.
From bc we construct bcd.

We have bd, be, cd, ce, de, abc, abd, abe and bcd. The set bd is one of the smallest and is a successor.
$bd'' = bd$.
From bd we construct bc and bde.

We have bc (from bd), be, cd, ce, de, abc, abd, abe, bcd and bde. The set bc (from bd) is one of the smallest but is not a successor.
We have be, cd, ce, de, abc, abd, abe, bcd and bde. The set be is one of the smallest and is a successor.
$be'' = bde$ so $be \to bde$ is added to the basis.
From be we construct bde.

We have cd, ce, de, abc, abd, abe, bcd, bde (from bd) and bde (from be). The set cd is one of the smallest and is a successor.
$cd'' = bcd$ so $cd \to bcd$ is added to the basis.
From cd we do not construct anything.

We have ce, de, abc, abd, abe, bcd, bde (from bd) and bde (from be). The set ce is one of the smallest and is a successor.
$ce'' = ce$.
From ce we construct cd.

We have cd (from ce), de, abc, abd, abe, bcd, bde (from bd) and bde (from be). The set cd (from ce) is the smallest but is not a successor.
We have de, abc, abd, abe, bcd, bde (from bd) and bde (from be). The set de is one of the smallest and is a successor.
$de'' = de$.
From de we do not construct anything.

We have abc, abd, abe, bcd, bde (from bd) and bde (from be). The logical closure of abc is updated to $abcd$.
We have abd, abe, bcd, bde (from bd), bde (from be) and $abcd$. The set abd is one of the smallest and is a successor.
$abd'' = abcde$ so $abd \to abcde$ is added to the basis.
From abd we do not construct anything.

We have abe, bcd, bde (from bd), bde (from be) and $abcd$. The logical closure of abe is updated to $abcde$ (and removed).
We have bcd, bde (from bd), bde (from be) and $abcd$. The set bcd is one of the smallest and is a successor.
$bcd'' = bcd$
From bcd we construct $bcde$.

We have bde (from bd), bde (from be), $abcd$ and $bcde$. The set bde (from bd) is the smallest and is a successor.
$bde'' = bde$.
From bde we construct bc.

We have bc, bde (from be), $abcd$ and $bcde$. The set bc is the smallest but is not a successor.
We have bde (from be), $abcd$ and $bcde$. The set bde (from be) is the smallest but is not a successor.
We have $abcd$ and $bcde$. The logical closure of $abcd$ is updated to $abcde$ (and re-

moved).
We have *bcde*. The set *bcde* is one of the smallest and is a successor.
$bcde'' = abcde$ so $bcde \to abcde$ is added to the basis.
From *bcde* we do not construct anything.

Here, again, the number of attribute sets grows in the beginning before plummetting near the end. However, we can see that the increase is slower than in the reverse lectic order.

Conclusion on the Comparison

All these results suggest that the lectic order is the most efficient, in terms of space consumption, of the three orders considered here. It performs the best in terms of average and maximal number of sets simultaneously in memory in all cases except for extremely dense contexts which rarely appear in practice. The reverse lectic order obtains the worst results while the increasing cardinality order is in-between. However, the algorithm we used for the enumeration in increasing cardinality order has no specific order for the treatment of attribute sets of a given cardinality. Considering the results we have, we believe that treating the attribute sets of the same cardinality in lectic order would improve the results whereas using the reverse lectic order would worsen them slightly.

We believe that the efficiency of the lectic order is due to the definition of the successor relation being based on it. A successor B of a set A is closer to A in lectic order than it is in reverse lectic order, so it is treated sooner and, consequently, is kept in memory for a shorter time. This is consistent with our observations that the reverse lectic order is the least efficient order. We speculate that the space complexity of an order depends on its similarity with the lectic order. Thus, orders $\leq_o$ such that $Y \leq_o X$ and $X \leq_{lec} Y$ for a high number of sets X and Y would perform significantly less well because the sets would remain in memory longer.

3.5.3 Optimization

Our experiments were realized using a generic algorithm in order to compare the behaviour of different orders. However, the number of sets in memory could be reduced for each order.

As a general rule, when computing the successors of a set A, the sets B that are lesser than A in the enumeration order need not be taken into account. Indeed, all the pseudo-intents lesser than A being known, the set B is directly equal to $\mathcal{B}^-(B)$ and, thus, will not change and become greater than A. As such, there is no need to add and conserve it.

The complexity of checking whether the newly constructed potential successors of A are lesser than A is obviously dependent on the considered order. In the cases of the lectic and reverse lectic orders, comparing two sets can be done in the size of A which, while relatively easy to do, would increase the runtime. In the case of the increasing cardinality order, the test simply requires that we compare the cardinalities of the two sets, which can be done in constant time.

When the sets are enumerated in increasing cardinality order with the closure of every set of cardinality n known before the computation of their successors, such as Algorithm 11, potential successors of cardinality $n-1$ have the closure of all their subsets known so their logical closure will not be updated. Thus, it is possible to check whether they are successors on the spot instead of waiting, further reducing the space requirements.

Once again, we are faced with the possibility to trade runtime for space complexity and vice versa.

3.6 Logical Closure

As we have seen in Section 2.5, the logical closure of a set A under a set of implications $\mathcal{I}$ can be computed in $O(|A|^2 \times |\mathcal{I}|)$ with the naive algorithm or in $O(|\mathcal{A}| \times |\mathcal{I}|)$ using LinClosure. However, in our algorithm, we need to compute the logical closures of A under $\mathcal{B}$ without the certainty that every implication in $\mathcal{B}$ is known. For this reason, we have to be able to compute $\mathcal{B}^-(A)$ in several, separate iterations without sacrificing complexity.

The naive algorithm (Algorithm 5) can easily be run in multiple steps. Let us suppose we have an attribute set A and a set of implications $\mathcal{I}_A \subseteq \mathcal{B}$. In order to compute $\mathcal{I}_A^-(A)$, we must scan the set of implications once for A and then once for every attribute in $\mathcal{I}_A^-(A) \setminus A$. This is a total of $|\mathcal{I}_A^-(A) \setminus A| + 1$ inclusion tests in the worst case. Similarly, in order to compute $\mathcal{B}^-(A)$ from $\mathcal{I}_A^-(A)$, we need a maximum of $|\mathcal{B}^-(A) \setminus \mathcal{I}_A^-(A)| + 1$ inclusion tests. Thus, we must perform a total of $|\mathcal{I}_A^-(A) \setminus A| + |\mathcal{B}^-(A) \setminus \mathcal{I}_A^-(A)| = |\mathcal{B}^-(A) \setminus A|$ inclusion tests, which is what the naive algorithm would have required in the first place.

The same thought process can be applied to LinClosure but the initialization phase of the algorithm can be problematic. If the counters and lists are kept after the partial logical closure, it requires the same number of operations as the batch version. However, the space requirements are too high. If the initializations are done again, from scratch, everytime the logical closure is updated, then the computation time is dramatically increased. As we saw in Section 2.5, the initialization phase already consumes a significant amount of time so the use of LinClosure in this context may depend on the number of logical closures we have to compute simultaneously. That is, it may depend on the chosen order. Concerning the closure phase itself, the initial partial closure requires $|\mathcal{I}_A^-(A)| \times |\mathcal{I}_A|$ operations. After that, assuming we keep the lists and counters as they are, computing $\mathcal{B}^-(A)$ from $\mathcal{I}_A^-(A)$ requires that we update the counters of the implications in $\mathcal{B} \setminus \mathcal{I}_A$ a total of $|\mathcal{A}|$ times, then those of all the implications in $\mathcal{B}$ a total of $\mathcal{B}^-(A) \setminus \mathcal{I}_A^-(A)$ times, which sums up to $|\mathcal{B}| \times \mathcal{B}^-(A)$ operations.

Chapter 4

Application to Relational Contexts

Contents

4.1 Motivation

Our initial goal was to learn ontologies from relational data expressed in description logics. That is, we wanted to compute implications on a context in which objects are described by both attributes and relations with other objects. The problem with relational data is that, if we want to represent the relations using attributes in a formal context, the number of attributes rapidly becomes too high for regular algorithms. Even though we do not pretend to outperform methods that have been proposed over the years ([7, 59, 89]), we wanted to see whether our algorithm would work efficiently on this type of data.

In this chapter, after a brief presentation of description logics as an example of language for relational data, we propose a modification of our algorithm that computes the Duquenne-Guigues basis of formal contexts that contain a great number of attributes expressing relations between objects. We show that the adaptation to relational contexts is straighforward and that it can be used with any language as long as the relations can be expressed as attributes that are functions of sets of attributes.

4.2 Relational Data

4.2.1 Description Logics

The term *concept* used in this section refers to concepts in description logics.

Syntactically, description logics use a set of *concept names* N_C (unary predicates), a set of *role names* N_R (binary predicates) and a set of *object names* N_O (constants). *Concepts* are constructed by combining concept and role names with *constructors* such as

- $\sqcap$
- $\sqcup$
- $\neg$
- $\exists$
- $\forall$
- $\leq^n$
- $\geq^n$

The set of constructors that a description logic language uses defines both its representational power and its complexity. Indeed, adding a new constructor allows for more complex knowledge to be represented but increases the complexity of operations such as testing the subsumption. For example, in the description logic $\mathcal{ALC}$ and for any concept name A, any concepts C and D and any role name r, the following constructions are concepts :

- $\top$
- $\bot$
- A
- $C \sqcap D$
- $C \sqcup D$
- $\neg C$
- $\exists r.C$
- $\forall r.C$

Concepts constructed using role names, of the form $\exists r.C$ and $\forall r.C$, are said to have a depth of n if the concept C used in the construction has a depth of $n-1$. Concept names have a depth of 0 and are said to be *atomic concepts*.

Semantically, a description logic associates concepts to sets of instances through an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the *domain* and $.^{\mathcal{I}}$ the *interpretation function* that maps subsets of $\Delta^{\mathcal{I}}$ to concept names and subsets of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to role names. From this, the set of instances associated to constructed concepts is as follows :

- $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
- $\bot = \emptyset$
- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $\neg C = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
- $(\exists r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y,\ (x,y) \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$
- $(\forall r.D)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y,\ (x,y) \in r^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$

In order to accurately describe knowledge in a particular domain, one must specify the relevant concepts and the relations that exist between them. In description logics, relations are expressed by means of terminological axioms. For any two concepts A and B, the *terminological axiom* $A \sqsubseteq B$ means that B *subsumes* A or that the concept B is *more general* than the concept A. With the interpretation $\mathcal{I}$, we have that $A \sqsubseteq B$ if and only if $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$. When $A \sqsubseteq B$ and $B \sqsubseteq A$, we note $A \equiv B$, meaning that A is *equivalent to* B (their interpretations are the same) or that A is *defined by* B.

Assertional axioms assign object names to concepts and pairs of object names to role names. As such, given two object names $o1$ and $o2$, a concept C and a role name r, the axiom $o1 : C$ means that $o1$ belongs to the concept C and $(o1, o2) : r$ means that $o2$ fulfills the role r for $o1$.

The knowledge is contained in a *knowledge base* composed of a *TBox* and an *ABox*. The TBox contains all the terminological axioms and, thus, the concepts and their relations while the ABox contains the *assertional axioms*.

The concepts names being unary predicates and the object names constants, it is easy to see that a formal context can be derived from the domain with domain objects as objects, concept names as attributes and the relevant assertional axioms of the ABox or the interpretation function itself as the incidence relation. In order to translate the information on roles into the context, we have to also consider as attributes the various non-atomic concepts built from roles. Assertional axioms of the form $(o1, o2) : r$ in the ABox mean that $o2$ fulfills the role r for $o1$. As such, $o1$ belongs to the concepts $\exists r.C$ where C is a concept to which $o2$ belongs. Thus, the attributes "$\exists r.C$" should describe the object "$o1$". Of course, the amount of attributes in such a derived context quickly explodes as the number of concepts of depth n is exponential in the number of concepts of depth $n-1$.

An implication $A \rightarrow B$ that holds in a context induced by a domain means that every object that belongs to all the concepts (attributes) in A also belongs to all the concepts in B. As such, it means that every object that belongs to the concept $\bigsqcap_{a \in A} a$ also belongs to the concept $\bigsqcap_{b \in B} b$, which would correspond to the terminological axiom $A \sqsubseteq B$. Of course, the truth of a terminological axiom is based on the interpretation and we do not always have access to the whole domain and only know the ABox. In these cases, using the implications of the formal context derived from the ABox can help "learn" an approximation of the set of terminological axioms.

For example, let us consider the following knowledge base composed of a TBox :

- $Ball \sqcap Red \sqsubseteq Toy$
- $Boy \sqsubseteq Person$

and the following ABox :

- $o1 : Boy$
- $o2 : Ball \sqcap Red$
- $o3 : Person$
- $o4 : Chair \sqcap Big$
- $(o1, o2) : playWith$
- $(o3, o4) : own$

This TBox contains the knowledge that all red balls are toys and all boys are persons. The ABox contains four objects : a boy (who is also a person), a red ball (which is also a toy), a person and a big chair. Additionally, we know that the boy plays with the red ball and that the person owns the big chair. This knowledge about objects can be rewritten in the following way using the existential quantification on roles :

- $o1 : Boy \sqcap Person \sqcap \exists playWith.(Ball \sqcap Red \sqcap Toy)$
- $o2 : Ball \sqcap Red \sqcap Toy$
- $o3 : Person \sqcap \exists own.(Chair \sqcap Big)$
- $o4 : Chair \sqcap Big$

As we know that $\exists r.(C \sqcap D) \subseteq \exists r.C \sqcap \exists r.D$, the ABox can again be rewritten as :

- $o1 : Boy \sqcap Person \sqcap \exists playWith.(Ball \sqcap Red \sqcap Toy) \sqcap \exists playWith.(Ball \sqcap Red) \sqcap \exists playWith.(Ball \sqcap Toy) \sqcap \exists playWith.(Red \sqcap Toy) \sqcap \exists playWith.Ball \sqcap \exists playWith.Red \sqcap \exists playWith.Toy$
- $o2 : Ball \sqcap Red \sqcap Toy$
- $o3 : Person \sqcap \exists own.(Chair \sqcap Big) \sqcap \exists own.Chair \sqcap \exists own.Big$
- $o4 : Chair \sqcap Big$

Each object being described by a conjunction of concepts, it is possible to consider each of these concepts as an attribute and build a formal context with 4 objects and 17 attributes.

4.2.2 Related Works

Most works on learning terminological axioms in description logics through the computation of implications use the same principle to reduce the number of attributes. In description logics, we have $\exists r.A \sqsubseteq \exists r.B$ when $A \subseteq B$. This translates, in the induced context, as $A \to B \Rightarrow \exists r.A \to \exists r.B$. It is known that $A \to A''$ and, trivially, $A'' \to A$. As such, we have that $\exists r.A \to \exists r.A''$ and $\exists r.A'' \to \exists r.A$. This means that $\exists r.A \equiv \exists r.A''$ for every attribute set A. It is thus sufficient to consider only attributes $\exists r.X$ with X closed.

The work of Rudolph [89] is the closest to ours as it also computes (in the lectic order) the Duquenne-Guigues basis of contexts induced by knowledge bases expressed in the description logic $\mathcal{EL}$ ($\sqcap$, $\exists$). It starts with a context containing attributes corresponding to atomic concepts and computes its intents and pseudo-intents with Attribute Exploration ([51]). For every computed intent A and every role r, it constructs a new attribute $\exists r.A$ and adds it to the context. It then uses Attribute Exploration again on the new context. It goes on until it reaches some arbitrary role depth. The resulting set of pseudo-intents, computed on the last context, is the Duquenne-Guigues basis of the induced context. The main problem with this method is that the same pseudo-intents can be found multiple times. Indeed, if the set A is a pseudo-intent and contains only attributes corresponding to atomic concepts, it is obvious that A is a pseudo-intent in every context for every role depth. Even if implications that have already been found are used to avoid finding them again, the closure of these pseudo-intents must be updated each and every time, which needlessly consumes time.

In [7], Baader and Distel use the same principles to compute a basis in $\mathcal{EL}_{gfp}$. It is interesting because it replaces the classic $.''$ closure operator by the DL-centric

$.^{ii}$, which corresponds to the notion of *most specific concept*. This closure operator associates to a set of domain objects the most specific concept to which all these objects belong. This evidently corresponds to the classic $.''$ operator but it has the advantage that it can be computed using subsumption algorithms in description logics (see [9]) and thus do not require that we compute the induced context.

Other works on the problem of combining formal concept analysis and description logics include two theses by Sertkaya ([92]) and Distel ([36]) as well as the work of Borchmann ([28, 27]) concerning the handling of errors in data sets during the computation of terminological axioms.

4.3 Relational Contexts

If we want to use our algorithms on data with objects represented by both unary (attributes) and binary (relations) predicates, we have to transform the information provided by the relations into attributes in order to obtain a formal context. In order to do that, we use the notion of *relational dependency* to generalize the constructs we find in different languages, such as roles quantificators in description logics.

Definition 13 *A relational dependency is a bijection r^k that maps an attribute set A to a single attribute $r^k(A) \notin A$ such that, for any two attribute sets X and Y, $X \to Y \Rightarrow r^k(X) \to r^k(Y)$.*

These relational dependencies help us represent the dependencies between attributes in a relational context. Let us consider the example of description logics where every concept corresponds to an attribute in the derived context. The four attributes a, b, $\exists r_1.(a \sqcap b)$ and $\forall r_1.(a \sqcap b)$ are, as we have seen, such that $\{a, b\} \to \{c\} \Rightarrow \exists r_1.(a \sqcap b) \to \exists r_1.c$ and $\forall r_1.(a \sqcap b) \to \forall r_1.c$. As such, we can use the relational dependencies r_1^1 and r_1^2 to represent respectively the existential quantification $\exists$ and the universal quantification $\forall$ on the role r_1. We would then have that $r_1^1(\{a, b\}) = \exists r_1.(a \sqcap b)$ and $r_1^2(\{a, b\}) = \forall r_1.(a \sqcap b)$.

An attribute i for which there is no attribute set A and relational dependency r^k such that $i = r^k(A)$ is said to be atomic or of depth 0. An attribute j is said to be of depth n if there is an attribute set A of depth $n-1$ for which there is a relational dependency r^k such that $r^k(A) = j$. We assume there are no cycles in the dependencies.

The definition of relational dependencies allows us to obtain information on valid implications between attributes of depth n from implications which premises contain attributes of depth $n-1$. Indeed, once $A \to B$ is known, we can directly add $r^k(A) \to r^k(B)$ to the set of valid implications and use it to compute the necessary logical closures. In the case where $r^k(A) \to r^k(B)$ is in the Duquenne-Guigues basis, that is one less implication to find.

Proposition 8 *For any attribute set A and any relation dependency r^k, we have $r^k(A) \to r^k(A'')$ and $r^k(A'') \to r^k(A)$.*

Proof For any attribute set A, we know that $A \to A''$ is valid. From the definition of a relational dependency, we deduce that $r^k(A) \to r^k(A'')$ is valid for

any r^k. Similarly, the implication $A'' \rightarrow A$ is always valid because $A \subseteq A''$ so $r^k(A'') \rightarrow r^k(A)$ is valid too. □

Proposition 8 states that every attribute $r^k(A)$ is equivalent to the attribute $r^k(A'')$. As such, every object of the context described by $r^k(A)$ is also described by $r^k(A)$ and vice versa. Thus, every attribute $r^k(A)$ with A not closed is redundant.

Proposition 9 *If $\mathcal{C}$ is a relational context and $\mathcal{C}2$ is the context obtained by removing from $\mathcal{C}$ the redundant attributes, then, for every pseudo-intent P of $\mathcal{C}2$, the set $P \cup \{r^k(X) \in \mathcal{A} \mid r^k(X'') \in P\}$ is a pseudo-intent of $\mathcal{C}$.*

Proof If P is a pseudo-intent of $\mathcal{C}2$ then, by definition, P is not closed and contains the closure of all its subsets. Since $P \cup \{r^k(X) \in \mathcal{A} \mid r^k(X'') \in P\}'$ is equal to P', the set $P \cup \{r^k(X) \in \mathcal{A} \mid r^k(X'') \in P\}$ is not closed. For the same reasons, it contains the closure of all its subsets. As such, $P \cup \{r^k(X) \in \mathcal{A} \mid r^k(X'') \in P\}$ is a pseudo-intent of $\mathcal{C}$. □

We can obtain the Duquenne-Guigues basis of a relational context by computing the basis of the same context minus the redundant attributes. While the number of attributes is still high, the reduced context contains significantly less attributes.

4.4 Computing the Duquenne-Guigues Basis of Relational Contexts

Computing the basis of a context containing relational attributes can thus be done on the reduced context, using the method presented in Section 3.3 in which sets are constructed using attributes that are either atomic or of the form $r^k(A)$ where A is closed. Evidently, using non-atomic attributes $r^k(A'')$ requires knowing the closure of A. This means that sets containing $r^k(A)$ - including the singleton $\{r^k(A)\}$ - must be considered after A. Fortunately, our work is all about choosing the order in which we want to enumerate the sets.

4.4.1 Algorithm

Using our algorithm on relational contexts would be straightforward if all the attributes were known in the beginning. This is not the case. If we want to use only attributes that are atomic or constructed from intents, we must wait to know said intents before using the attributes. The problem is that if a set $r^k(A'')$ is added to the list of "interesting" attributes only after A'' has been found, the logical closure of $B \cup \{r^k(A'')\}$ has not been considered for sets B that are lesser than A in the chosen order (for which the set of successors has already been computed without knowing this attribute was relevant). The most obvious solution is to store all the sets $B \in \Phi$ and compute the logical closure of $B \cup \{r^k(A'')\}$ everytime a new A'' is computed. This would certainly require an important amount of space because of the exponential number of attribute sets in Φ. Algorithm 12 shows how to compute the Duquenne-Guigues basis of a relational context with a maximum depth of $maxDepth$.

Algorithm 12 Enumeration of Pseudo-Intents in Relational Contexts

```
1: Sets = {∅}
2: Impl = ∅
3: Att = {atomic attributes}
4: while Sets is not empty do
5:   Pick the first set A in Sets
6:   Update the logical closure of A
7:   if A has not changed then
8:     if A is a successor of the set it has been generated from then
9:       B = A''
10:      if A ≠ B then
11:        Impl = Impl ∪ {A → B}
12:      else
13:        if Depth(B) < maxDepth then
14:          Att = Att ∪ {r^k(B) | r^k is a relational dependency}
15:          for every computed intent X and relational dependency r^k do
16:            Sets = Sets ∪ Impl⁻({x ∈ X | x < r^k(B)} ∪ {r^k(B)})
17:          end for
18:        end if
19:      end if
20:      for every attribute i ∈ Att \ A greater than min(A) do
21:        Sets = Sets ∪ Impl⁻({a ∈ A | a < i} ∪ {i})
22:      end for
23:    end if
24:  else
25:    Place the new A in Sets according to the chosen order
26:  end if
27: end while
28: return Impl
```

Proposition 10 *Algorithm 12 terminates and returns the Duquenne-Guigues basis of the input context.*

Proof From an intent A we construct new attributes $r^k(A)$ such that the depth of $r^k(A)$ is greater than the depth of A. The depth being bounded and r^k being bijective, we can only add a finite number of attributes. For each attribute set B, we consider the sets $B \oplus i$ for every attribute i known when B is considered, and the sets $B \oplus$ j for every attribute j for each new attribute constructed with B. As such, each attribute set is used to construct a finite number of other attribute sets. Each new attribute set being of greater cardinality, the algorithm terminates.

Propositions 8 and 9 imply that the algorithm returns the Duquenne-Guigues basis of the context without redundant attributes, which is equivalent to the Duquenne-Guigues basis of the input context. □

It is important to note that, even though the algorithm enumerates the intents and pseudo-intents of the context without the redundant attributes, the closures themselves must be done on the initial context as it is impossible to know a priori whether an attribute is redundant.

As the number of attributes with a depth of n is equal to the number of intents of depth $n-1$, the amount of attributes increases exponentially with each level of depth. Furthermore, being required to keep the whole intent lattice in memory, the algorithm rapidly becomes unusable as soon as there are too many atomic attributes or relations.

4.4.2 Example

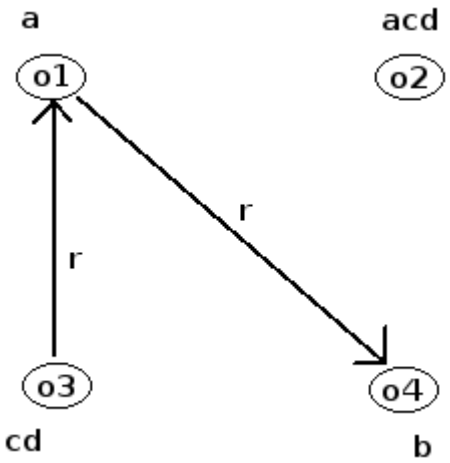


Figure 4.1: Example of relational data

Let us consider four objects $o1$, $o2$, $o3$ and $o4$, four attributes a, b, c and d and a relation (or role) r. Figure 4.1 illustrates the description of objects and their relations. There is a single relational dependency that represents the existential quantification. From the objects, we obtain the context $\mathcal{C}_R = (\{o1, o2, o3, o4\}, \mathcal{A}_R, \mathcal{I}_R)$ in which $\mathcal{A}_R$ contains every attribute of depth 0, 1 or 2. The descriptions of the objects are as follows :

- $o1$: a, $r(\emptyset)$, $r(b)$
- $o2$: a, c, d
- $o3$: c, d, $r(\emptyset)$, $r(a)$, $r(r(\emptyset))$, $r(r(b))$, $r(ar(\emptyset))$, $r(ar(b))$, $r(r(\emptyset)r(b))$, $r(ar(\emptyset)r(b))$
- $o4$: b

The algorithm starts with the set of attributes $Att = \{a, b, c, d\}$.
The closure of $\emptyset$ is $\emptyset$. It is an intent so $r(\emptyset)$ is added to the list of attributes and $Att = \{a, b, c, d, r(\emptyset)\}$.

We have a, b, c, d and $r(\emptyset)$. We choose a.
The closure of a is a. It is an intent so $r(a)$ is added to the list of attributes and $Att = \{a, b, c, d, r(\emptyset), r(a)\}$.

We have b, c, d, $r(\emptyset)$, ab, ac, ad, $ar(\emptyset)$, $r(a)$, $ar(a)$. We choose b.
The closure of b is b. It is an intent so $r(b)$ is added to the list of attributes and $Att = \{a, b, c, d, r(\emptyset), r(a), r(b)\}$.

We have c, d, $r(\emptyset)$, ab, ac, ad, $ar(\emptyset)$, $r(a)$, $ar(a)$, bc, bd, $br(\emptyset)$, $br(a)$, $r(b)$, $ar(b)$, $br(b)$. We choose c.
The closure of c is cd so we add $c \to cd$ to the list of implications.

We have d, $r(\emptyset)$, ab, ac, ad, $ar(\emptyset)$, $r(a)$, $ar(a)$, bc, bd, $br(\emptyset)$, $br(a)$, $r(b)$, $ar(b)$, $br(b)$, cd. We choose d.
The closure of d is cd so we add $d \to cd$ to the list of implications.

We have $r(\emptyset)$, ab, ac, ad, $ar(\emptyset)$, $r(a)$, $ar(a)$, bc, bd, $br(\emptyset)$, $br(a)$, $r(b)$, $ar(b)$, $br(b)$, cd. We choose $r(\emptyset)$.
The closure of $r(\emptyset)$ is $r(\emptyset)$. It is an intent so $r(r(\emptyset))$ is added to the list of attributes and $Att = \{a, b, c, d, r(\emptyset), r(a), r(b), r(r(\emptyset))\}$.

We have ab, ac, ad, $ar(\emptyset)$, $r(a)$, $ar(a)$, bc, bd, $br(\emptyset)$, $br(a)$, $r(b)$, $ar(b)$, $br(b)$, cd, $r(\emptyset)r(a)$, $r(\emptyset)r(b)$, $r(r(\emptyset))$, $ar(r(\emptyset))$, $br(r(\emptyset))$, $r(\emptyset)r(r(\emptyset))$. We choose ab.
The closure of ab is $\mathcal{A}$ so we add $ab \to \mathcal{A}$ to the list of implications.
And so on.

The algorithm ends with $Att = \{a, b, c, d, r(\emptyset), r(a), r(b), r(r(\emptyset)), r(acd), r(ar(\emptyset)r(b)), r(cd)\}$ and the following basis of implications :

- $c \to cd$
- $d \to cd$
- $ab \to \mathcal{A}$
- $ar(\emptyset) \to ar(\emptyset)r(b)$
- $bcd \to \mathcal{A}$
- $br(\emptyset) \to \mathcal{A}$
- $r(\emptyset)r(a) \to \{o3\}'$
- $r(\emptyset)r(b) \to ar(\emptyset)r(b)$
- $r(\emptyset)r(r(\emptyset)) \to \{o3\}'$
- $cdr(\emptyset) \to \{o3\}'$
- $r(\emptyset)r(cd) \to \mathcal{A}$

The implications can be transposed in the description logic syntax, replacing $r(A)$ by $\exists r.A$:

- $c \sqsubseteq d$
- $d \sqsubseteq c$

- $a \sqcap b \sqsubseteq \bot$
- $a \sqcap \exists r.\top \sqsubseteq \exists r.b$
- $b \sqcap c \sqcap d \sqsubseteq \bot$
- $b \sqcap \exists r.\top \sqsubseteq \bot$
- $\exists r.\top \sqcap \exists r.a \sqsubseteq \bigsqcap\{o3\}'$
- $\exists r.\top \sqcap \exists r.b \sqsubseteq a$
- $\exists r.\top \sqcap \exists r.(\exists r.\top) \sqsubseteq \bigsqcap\{o3\}'$
- $c \sqcap d \sqcap \exists r.\top \sqsubseteq \bigsqcap\{o3\}'$
- $\exists r.\top \sqcap \exists r.(cd) \sqsubseteq \bot$

4.5 Scope of the Application

Our algorithm can be used on any formal contexts in which attributes can be represented by relational dependencies. Some information on relations between objects can easily be represented by such attributes.

- The knowledge "it is in a relation r with an object that is described by X", noted $\exists r.X$ in description logics, can be represented by the set of attributes $\{r^1(Y) \mid Y \subseteq X\}$. If every object that is A is also B ($A \to B$), then something that is in a relation r with an object that is A is also in a relation r with something that is B. Thus, it corresponds to the framework of relational dependencies.

- The knowledge "every object it is in a relation r with is described by X", noted $\forall r.X$ in description logics, can be represented by the set of attributes $\{r^2(x) \mid x \in X\}$. Indeed, if knowing that every object which something is in a relation r with is described by the attributes a and b means that it is also described by $\{a\} \cup \{b\}$. Consequently, we can consider only individual attributes independently without loss of information, which reduces the number of attributes. As before, if $a \to b$, then being in a relation r with only objects that are a implies being in a relation r with only objects that are b and our algorithm can be applied directly.

- The knowledge "it is in a relation r with **more** than n objects that are described by X" can be represented by the set of attributes $\{r^3(Y) \mid Y \subseteq X\}$. Indeed, if more than n objects are described by X, then even more are described by the subsets of X. Once again, if $A \to B$, then being in a relation r with more than n objects described by A implies being in a relation r with more than n objects described by B and our algorithm can be used.

- The knowledge "it is in a relation r with **less** than n objects that are described by X" is harder to represent because there can be more than n objects described by the subsets of X. Moreover, if $A \to B$, we **cannot** say that

being in a relation r with less than n objects described by A implies being in a relation r with less than n objects described by B. Consequently, this particular knowledge cannot be represented with relational dependencies and our algorithm cannot treat it. There is always the possibility to consider all the possible attributes for this particular knowledge as atomic attributes and use the algorithm anyway, but the size of the attribute set would explode.

- The knowledge "it is **not** in a relation r with any object described by X" similarly cannot be treated by our algorithm because if $A \to B$ then not being in a relation r with an object described by A does not necessarily imply not being in a relation r with an object described by B.

In conclusion, it seems that only knowledge of the form "being in relation r with **more than** n objects that are X" can be represented efficiently using relational dependencies.

Chapter 5

Computing a Basis for Association Rules

Contents

5.1 Motivation

Implications, and the whole of formal concept analysis, are closely related to data mining and databases theory. In databases, the notions of functional dependency and minimal keys are tied to implications. In data mining, the field of association rules mining is concerned with finding sets of patterns of which implications are subsets. In fact, formal concept analysis has long been used as a mathematical framework to help develop algorithms in data mining. It is therefore natural that we took an interest in association rules.

In this chapter, after a brief overview of association rules, we show that our algorithm can be modified to compute a basis for all the association rules in a formal context without an increase in the worst-case complexity. More specifically, it can obtain a basis for association rules by computing a basis for uncertain implications, the Luxenburger basis, in addition to the Duquenne-Guigues basis.

5.2 Association Rules

Association rules exist in the same framework as implications, i.e. a context with a set of objects described by attributes. The *support* of an attribute set A is the number of objects described by A. In other words, $supp(A) = |A'|$. An attribute set is said to be *frequent* if its support is above a certain threshold. An association rule is an expression $A \rightarrow_{s,c} B$ in which s is the support of A and c is the confidence of the rule, defined as $\frac{supp(A \cup B)}{supp(A)}$. The confidence of the rule expresses the likelihood to have the attribute set B describe an object described by A. Naturally, finding association rules with a high confidence and support has been an object of interest in the data mining community for many years.

	a	b	c	d	e
o1	×	×			
o2		×		×	×
o3		×	×	×	
o4			×		×
o5				×	×

Figure 5.1: Context 1

Introduced by Agrawal (see [1]) along the notion of association rule, APRIORI is the first algorithm to compute frequent association rules. As illustrated in Algorithm 13, it uses a bottom-up approach in which attribute sets are constructed from their subsets by adding a single attribute. Using the property that the support of an attribute set is lesser than the support of its subsets, the algorithm constructs new sets only from frequent ones. Effectively enumerating all the frequent elements (and some non-frequent) of the powerset of $\mathcal{A}$, APRIORI is inefficient but historically significant.

Algorithm 13 APRIORI

```
1: F_0 = {∅}
2: k = 1
3: while F_{k-1} ≠ ∅ do
4:    F_k = ∅
5:    for every set S in F_{k-1} do
6:       for every attribute i not in S do
7:          if S ∪ {i} is frequent then
8:             F_k = F_k ∪ {S ∪ {i}}
9:          end if
10:      end for
11:   end for
12:   k = k + 1
13: end while
14: Return ⋃_{j=1..k} F_j
```

As the support of the rules are the support of the premises, works on the computation of association rules are mainly interested in finding frequent attribute sets first, then constructing the rules from them later. Early on, algorithms tended to enumerate most, if not all, of the frequent sets. Later, it has been found that, since any two attribute sets A and B such that $A'' = B''$ have the same extent and thus the same support, it is sufficient to know only closed sets. Therefore, computing the frequent part of the concept lattice of a context is enough to be able to retrieve all the frequent association rules.

We saw in Section 2.3 that several algorithms can be used to compute the concepts set and most of them can be modified to only compute its frequent part. However, there is one algorithm, TITANIC (see [97]), that has been proposed specifically for the problem of enumerating only frequent closed sets. It follows the same principles as APRIORI but, instead of considering every frequent set as a candidate, it focuses on so-called *key sets*, i.e. minimal generators of sets under $.''$. In addition to pruning sets that are not frequent, it prunes sets that are not keys. Keys can easily be recognized as their support is different from the support of all their lower covers in the powerset of $\mathcal{A}$. Once all the key sets are found, closed sets can be retrieved by intersecting all the keys with the same support.

Of course, the number of association rules can be very high and tuning the thresholds for the support and confidence helps the pruning but, as it is insufficient, bases for association rules have been found and studied.

Association rules with a confidence of 1, sometimes called *global implications*, are the implications studied throughout our work. As we have seen, the Duquenne-Guigues basis plays the role of smallest set of implication that allows for all the others to be found. Association rules with a confidence $c < 1$, called *partial implications*, have their own basis called the *Luxenburger basis* and introduced in [77]. It can be shown that $A \rightarrow_{s,c} B$ if and only if $A \rightarrow_{s,c} A \cup B$ and $A'' \rightarrow_{s,c} B''$. Hence, we can restrict ourselves to associations rules in which A and B are frequent intents and $A \subset B$.

Theorem 1 *(From [77]) Let $A, B, C \subseteq \mathcal{A}$ be intents with $A \subseteq B \subseteq D$. Then $A \rightarrow_{s,c} B$ and $B \rightarrow_{s',c'} D$ implies $A \rightarrow_{s,c \times c'} D$.*

This result implies that association rules can be derived by transitivity. Consequently, instead of all the possible association rules between intents, we only have to consider rules of the form $A \rightarrow_{s,c} B$ such that there is no intent C such that $A \subseteq C \subseteq B$. In other words, only the covering graph of the lattice of intents is needed. Moreover, Luxenburger showed that, as long as a cycle exists in the (undirected) graph in which the intents are vertex and partial implication edges, we could remove a rule and still be able to derive all the partial implications. This leads us to the following definition :

Definition 14 *The Luxenburger basis of a formal context $\mathcal{C}$ is a set $\mathcal{L}_B = \{A \rightarrow_{s,c} B\}$ of partial implications such that B is an upper cover of A in the intents lattice of $\mathcal{C}$, the set $\{(A, B) \mid A \rightarrow_{s,c} B \in \mathcal{L}_B\}$ of edges forms a spanning tree of the covering graph of the intents lattice of $\mathcal{C}$ and a single partial implication has the whole set of attributes as its conclusion.*

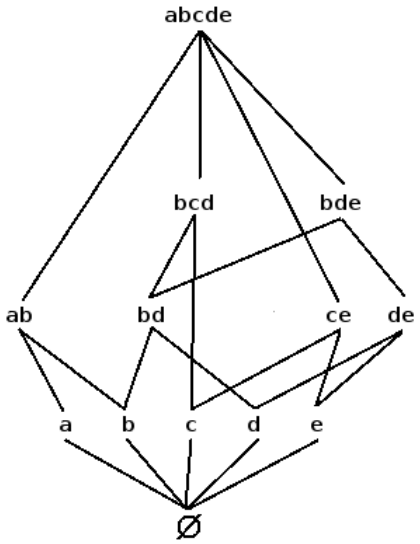


Figure 5.2: Covering graph of the lattice of intents of Context 1

Multiple spanning trees exist for an intents lattice so the Luxenburger basis of a context is not unique. Figures 5.2, 5.3 and 5.4 illustrate the covering graph of the intents lattice of our running example (as shown once again in Figure 5.1) and two possible spanning trees, respectively. The Luxenburger basis corresponding to the **first** tree is :

- $\emptyset \rightarrow_{5,0.6} b$
- $\emptyset \rightarrow_{5,0.4} c$
- $\emptyset \rightarrow_{5,0.6} d$
- $\emptyset \rightarrow_{5,0.6} e$

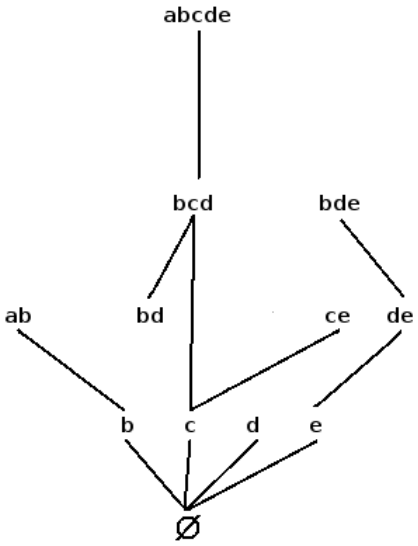


Figure 5.3: A spanning tree of the covering graph of the lattice of intents of Context 1

- $b \rightarrow_{3,0.33} ab$
- $c \rightarrow_{2,0.5} bcd$
- $c \rightarrow_{2,0.5} ce$
- $e \rightarrow_{3,0.66} de$
- $bd \rightarrow_{2,0.5} bcd$
- $de \rightarrow_{2,0.5} bde$
- $bcd \rightarrow_{1,0} abcde$

The Luxenburger basis corresponding to the **second** tree is :

- $\emptyset \rightarrow_{5,0.4} c$
- $\emptyset \rightarrow_{5,0.6} d$
- $\emptyset \rightarrow_{5,0.6} e$
- $b \rightarrow_{3,0.33} ab$
- $b \rightarrow_{3,0.66} bd$
- $d \rightarrow_{3,0.66} bd$
- $d \rightarrow_{3,0.66} de$
- $e \rightarrow_{3,0.33} ce$
- $bd \rightarrow_{2,0.5} bcd$

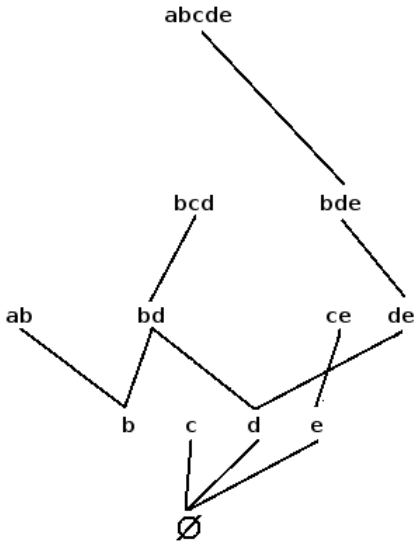


Figure 5.4: A second spanning tree of the covering graph of the lattice of intents of Context 1

- $de \rightarrow_{2,0.5} bde$
- $bde \rightarrow_{1,0} abcde$

Deriving the confidence of associations rules of the form $A \rightarrow B$ (where both A and B are intents) is easy. It suffices to find a path in the spanning tree that leads from A to B. If the path is $\{X_1, ..., X_n\}$, then the confidence of the rule $X_1 \rightarrow_{s,c} X_n$ is $\Pi_{i=1...n-1} \frac{supp(X_i)}{supp(X_{i+1})}$. This is effectively multiplying the confidences of the rules on the path while reversing those of the rules we go contrary to. For example, let us suppose we want to know the confidence of $c \rightarrow_{s,c} bcd$ in our Context 1. With the first Luxenburger basis, the rule $c \rightarrow_{2,0.5} bcd$ exists so the confidence is known as 0.5. With the second Luxenburger basis, the shortest path is $\{c, \emptyset, d, bd, bcd\}$. The confidence of the rule is thus $(\frac{5}{2}) \times (\frac{3}{5}) \times (\frac{2}{3}) \times (\frac{1}{2}) = 0.5$. Since every intent is in a rule, and since the confidence of a rule between two intents can be computed and every other rule can be derived from a rule between intents, the Luxenburger basis is enough to derive all the association rules with a confidence of less than 1.

The Luxenburger basis and the Duquenne-Guigues basis form together a basis for association rules. If one is interested only in **frequent** association rules, as is often the case, it suffices to keep only the frequent implications and partial implications to obtain a base.

5.3 (Also) Computing the Luxenburger basis

We now want to show that the algorithms we proposed are able to compute the Luxenburger basis along with the Duquenne-Guigues basis with only a slight modification.

5.3.1 Successor Relation between Intents

We saw in Section 3.2 that the successor relation, presented in that same section, defines a spanning tree of the covering graph of the lattice Φ of attribute sets closed under $\mathcal{B}^-$. In the successor relation, every element of Φ has a unique predecessor, its lectically greatest subset in the lattice. We have also seen that the lattice Φ contains every intent and pseudo-intent and that a pseudo-intent can have a maximum of one successor. This means that every intent in Φ has an intent as predecessor, except for some essential intents.

Definition 15 *(**Successor relation between intents**) For any two attribute sets $A, B \in \Phi$, B is a successor of A (noted $A \rightsquigarrow_I B$) if and only if A is the lectically greatest strict subset of B in Φ that is an intent.*

The relation $\rightsquigarrow_I$ is similar to $\rightsquigarrow$. Every element of Φ has a unique predecessor, except for $\emptyset$ and $\emptyset''$, and between 0 and $|\mathcal{A}|$ successors, except for $\mathcal{A}$ that never has a successor. The difference is that the predecessor of a set is always an intent.

Proposition 11 *For any $A \in \Phi$, there is a unique finite sequence $A_1, ..., A_n$ of elements of intents such that $\emptyset'' \rightsquigarrow_I A_1 \rightsquigarrow_I ... \rightsquigarrow_I A_n \rightsquigarrow_I A$.*

Proof Every non-empty set in the lattice Φ contains $\emptyset''$ and has a strict subset that is an intent as a predecessor so such a sequence exists. The number of attributes is finite so the sequence itself is finite. The predecessor of an attribute set is unique so the sequence is unique. □

Corollary 2 *The successor relation $\rightsquigarrow_I$ defines a spanning tree of the covering graph of the intents lattice*

Since $\rightsquigarrow_I$ defines a spanning tree of the covering graph of the intents lattice and $\mathcal{A}$ has a single predecessor, given Definition 14, the set of all $A \rightarrow_{s,c} B$ such that $A \rightsquigarrow_I B$, A and B are intents, $s = |A'|$ and $c = \frac{|B'|}{|A'|}$ is the Luxenburger basis of the context. Figure 5.5 shows both the spanning trees of Φ and the intents lattice induced by $\rightsquigarrow_I$ in our running example.

Proposition 12 *For any intent A, if $A \rightsquigarrow B$ then $A \rightsquigarrow_I B$.*

The differences between $\rightsquigarrow$ and $\rightsquigarrow_I$ are for the essential intents B such that $A \rightsquigarrow B$ with A being a pseudo-intent. If we want to compute the Luxenburger basis with our enumeration algorithms, we have to find the predecessor (for $\rightsquigarrow_I$) of those essential intents.

Proposition 13 *Given two intents A and B and a pseudo-intent P such that $A \rightsquigarrow P \rightsquigarrow B$, the intent C such that $C \rightsquigarrow_I B$ is either A or a superset of A.*

Proof Let us suppose that $A \rightsquigarrow P \rightsquigarrow B$ and that there is an intent $C \subseteq B$ that is lectically maximal and not a superset of A. From the supposed order, we deduce that $\min(P \Delta A) \in P$ and $\min(C \Delta A) \in C$. However, we know that $A \vee C = B$ because otherwise C would not be lectically maximal. As such, we have $B \setminus A \subseteq C$. More precisely, we have that $\min(P \Delta A) = \min(C \Delta A) \in P$. By the definition of

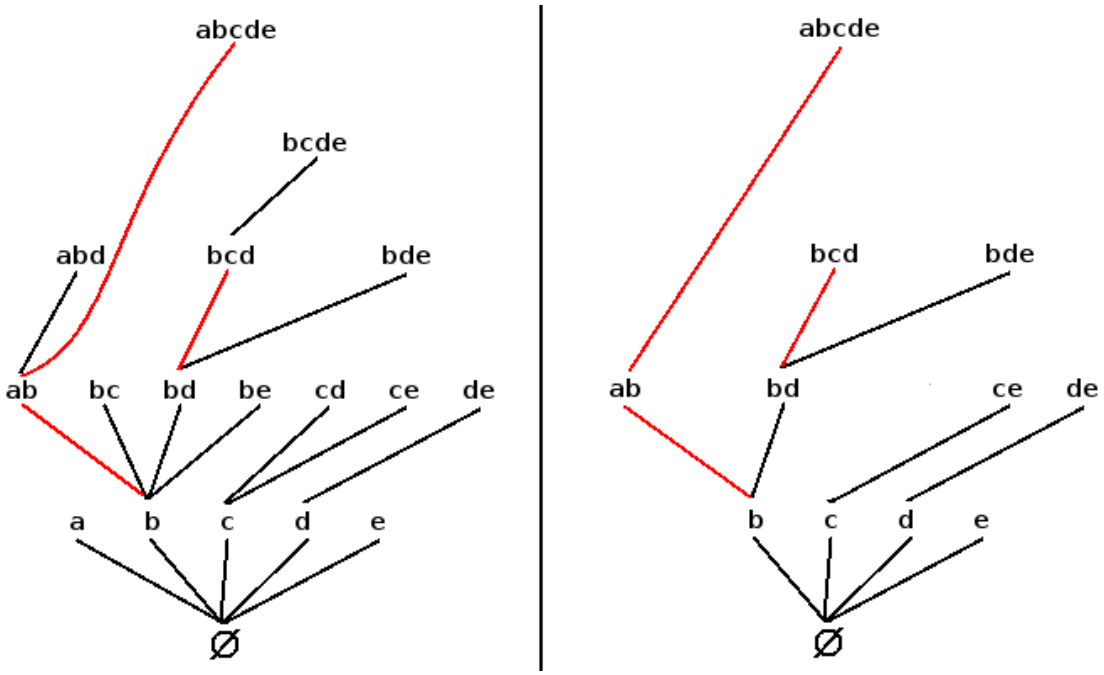


Figure 5.5: Spanning tree of Φ and of the intents lattice induced by $\rightsquigarrow_I$ in Context 1 (differences with $\rightsquigarrow$ are in red)

the successor relation, we know that $A \oplus \min(P \Delta A) = P$ so C is a superset of P, which contradicts our hypothesis. Thus, the lectically greatest subset of B that is an intent is a superset of A. $\square$

For any two intents A and B such that $A \rightsquigarrow P \rightsquigarrow B$, PREDECESSOR (Algorithm 14) computes the predecessor of B for the relation $\rightsquigarrow_I$.

Algorithm 14 Predecessor(B)

```
1: INPUT A et B such that A ⇝ P ⇝ B
2: C = A
3: for every attribute i in B \ A in increasing order do
4:    if B(C ∪ {i}) ≠ B then
5:       C = C ∪ {i}
6:    end if
7: end for
8: return C
```

Proposition 14 *Algorithm 14 terminates and computes the lectically greatest intent that is a subset of B''*

Proof There is a finite number of attributes and the loop goes through them in linear order so the algorithm terminates. The attribute set it returns, C, is either A or closed under $\mathcal{B}(.)$, so it is an intent. It is the logical closure of a subset of B and it is not equal to B so we have $C \subset B$. Let us suppose that there is an intent $D \subset B$ lectically greater than C with $i = \min(C \Delta D)$. The attribute i is such that $\mathcal{B}(X \cup \{i\}) \subset B$ for any $A \subseteq X \subseteq D$ so the algorithm would have added it to C. Hence, we have $C = D$.

Thus, the algorithm returns C, the lectically greatest intent that is a subset of B. □

Algorithm 14, PREDECESSOR, has to compute a maximum of $|\mathcal{A}|$ logical closures so the algorithm is in $O(|\mathcal{A}|^2 \times |\mathcal{B}|)$, which is the same complexity as SUCCESSORS.

5.3.2 Algorithm

Now that we know how to obtain the predecessors of essential intents for $\rightsquigarrow_I$, we can construct the spanning tree of the intent lattice. Proposition 12 says that we can start from the spanning tree induced by $\rightsquigarrow$ and then compute the new predecessors of the essential intents that are successors of pseudo-intents. In order to do that, we can use Algorithm 10 and apply Algorithm 14 everytime an intent is the successor of a pseudo-intent. Algorithm 15 is a simple modification of Algorithm 10 that computes the spanning tree of the intents lattice and, thus, the Luxenburger basis along with the Duquenne-Guigues basis.

Proposition 15 *Algorithm 15 terminates and returns the Duquenne-Guigues basis and Luxenburger basis.*

Proof The algorithm terminates and returns the Duquenne-Guigues basis for the same reasons as Algorithm 10. If we have $A \rightsquigarrow B$ with both A and B being intents, Proposition 12 states that $A \rightsquigarrow_I B$ and, as such, $A \rightarrow_{s,c} B$ is in the Luxenburger basis. If $A \rightsquigarrow B$ with A being a pseudo-intent, Proposition 14 states that the set C resulting from the application of the PREDECESSOR algorithm to the set B is such that $C \rightsquigarrow_I B$ and, as such, $C \rightarrow_{s,c} B$ is in the Luxenburger basis. Therefore, the algorithm returns a subset of the Luxenburger basis. Every intent B is enumerated so every $A \rightarrow_{s,c} B$ is found. Hence, the algorithm returns a superset of the Luxenburger basis and, as such, the Luxenburger basis itself. □

The algorithm enumerates every intent and pseudo-intent. While we need to compute the successors of each set, we have seen in Section 3.2 that only the intents required any additional computation. As such, SUCCESSORS is applied once to every intent. In order to compute the Luxenburger basis, we must compute the predecessor under $\rightsquigarrow_I$ of the essential intents that are successors of pseudo-intents. There cannot be more essential intents than pseudo-intents so the number of sets we have to apply either SUCCESSORS or PREDECESSOR to is less than $|\Phi|$. PREDECESSOR and SUCCESSORS having the same worst-case complexity, Algorithm 15 is in $O(|\Phi| \times |\mathcal{A}|^2 \times |\mathcal{B}|)$.

5.3.3 Example

In our running example, Algorithm 15 runs as follows :

$\emptyset'' = \emptyset$.
From $\emptyset$ we construct a, b, c, d and e.

We have a, b, c, d and e. The set a is one of the smallest and is a successor.
$a'' = ab$ so $a \rightarrow ab$ is added to the **Duquenne-Guigues** basis.
From a we construct ab.

Algorithm 15 Enumeration of Pseudo-Intents and Partial Implications

```
1: Sets = {∅}
2: Impl = ∅
3: Lux = ∅
4: while Sets is not empty do
5:   Pick the first set A in Sets
6:   Update the logical closure of A
7:   if A has not changed then
8:     if A is a successor of the set it has been generated from then
9:       B = A''
10:      if A ≠ B then
11:        Impl = Impl ∪ {A → B}
12:      else
13:        if A has been constructed from an intent C then
14:          sup = |A'|
15:          cf = sup / |C'|
16:          Lux = Lux ∪ C →_{sup,cf} A
17:        else
18:          D = Predecessor(A)
19:          sup = |A'|
20:          cf = sup / |D'|
21:          Lux = Lux ∪ D →_{sup,cf} A
22:        end if
23:      end if
24:      for every attribute i ∉ A greater than min(A) do
25:        Sets = Sets ∪ Impl⁻({a ∈ A || a < i} ∪ {i})
26:      end for
27:    end if
28:  else
29:    Place the new A in Sets according to the chosen order
30:  end if
31: end while
32: return Impl, Lux
```

We have b, c, d, e and ab. The set b is one of the smallest and is a successor.
$b'' = b$ and b has been constructed from $\emptyset$ so $\emptyset \rightarrow_{5,0.6} b$ is added to the **Luxenburger** basis.
From b we construct bc, bd, be.

We have c, d, e, ab, bc, bd and be. The set c is one of the smallest and is a successor.
$c'' = c$ and c has been constructed from $\emptyset$ so $\emptyset \rightarrow_{5,0.4} c$ is added to the **Luxenburger** basis.
From c we construct cd and ce.

We have d, e, ab, bc, bd, be, cd and ce. The set d is one of the smallest and is a successor.
$d'' = d$ and d has been constructed from $\emptyset$ so $\emptyset \rightarrow_{5,0.6} d$ is added to the **Luxen-**

burger basis.
From d we construct de.

We have e, ab, bc, bd, be, cd, ce and de. The set e is one of the smallest and is a successor.
$e'' = e$ and e has been constructed from $\emptyset$ so $\emptyset \rightarrow_{5,0.6} e$ is added to the **Luxenburger** basis.
From e we do not construct anything.

We have ab, bc, bd, be, cd, ce and de. The set ab is one of the smallest and is a successor.
$ab'' = ab$ and ab is an essential intent that has been constructed from a. The PREDECESSOR algorithm finds that $b \rightsquigarrow_I ab$ so $b \rightarrow_{3,0.33} ab$ is added to the **Luxenburger** basis.
From ab we construct abc, abd and abe.

We have bc, bd, be, cd, ce, de, abc, abd and abe. The set bc is one of the smallest and is a successor.
$bc'' = bcd$ so $bc \rightarrow bcd$ is added to the **Duquenne-Guigues** basis.
From bc we construct bcd.

We have bd, be, cd, ce, de, abc, abd, abe and bcd. The set bd is one of the smallest and is a successor.
$bd'' = bd$ and bd has been constructed from b so $b \rightarrow_{3,0.66} bd$ is added to the **Luxenburger** basis.
From bd we construct bc and bde.

We have bc (from bd), be, cd, ce, de, abc, abd, abe, bcd and bde. The set bc (from bd) is one of the smallest but is not a successor.
We have be, cd, ce, de, abc, abd, abe, bcd and bde. The set be is one of the smallest and is a successor.
$be'' = bde$ so $be \rightarrow bde$ is added to the **Duquenne-Guigues** basis.
From be we construct bde.

We have cd, ce, de, abc, abd, abe, bcd, bde (from bd) and bde (from be). The set cd is one of the smallest and is a successor.
$cd'' = bcd$ so $cd \rightarrow bcd$ is added to the **Duquenne-Guigues** basis.
From cd we do not construct anything.

We have ce, de, abc, abd, abe, bcd, bde (from bd) and bde (from be). The set ce is one of the smallest and is a successor.
$ce'' = ce$ and ce has been constructed from c so $c \rightarrow_{2,0.5} ce$ is added to the **Luxenburger** basis.
From ce we construct cd.

We have cd (from ce), de, abc, abd, abe, bcd, bde (from bd) and bde (from be). The set cd (from ce) is the smallest but is not a successor.
We have de, abc, abd, abe, bcd, bde (from bd) and bde (from be). The set de is one

of the smallest and is a successor.
$de'' = de$ and de has been constructed from d so $d \rightarrow_{3,0.66} de$ is added to the **Luxenburger** basis.
From de we do not construct anything.

We have abc, abd, abe, bcd, bde (from bd) and bde (from be). The logical closure of abc is updated to $abcd$.
We have abd, abe, bcd, bde (from bd), bde (from be) and $abcd$ (from ab). The set abd is one of the smallest and is a successor.
$abd'' = abcde$ so $abd \rightarrow abcde$ is added to the **Duquenne-Guigues** basis.
From abd we construct $abcde$.

We have abe, bcd, bde (from bd), bde (from be), $abcd$ and $abcde$. The logical closure of abe is updated to $abcde$ (and removed).
We have bcd, bde (from bd), bde (from be), $abcd$ and $abcde$. The set bcd is one of the smallest and is a successor.
$bcd'' = bcd$ and bcd is an essential intent that has been constructed from a. The PREDECESSOR algorithm finds that $bd \rightsquigarrow_I bcd$ so $bd \rightarrow_{2,0.5} bcd$ is added to the **Luxenburger** basis.
From bcd we construct $bcde$.

We have bde (from bd), bde (from be), $abcd$, $bcde$ and $abcde$. The set bde (from bd) is the smallest and is a successor.
$bde'' = bde$ and bde has been constructed from bd so $bd \rightarrow_{2,0.6} bde$ is added to the **Luxenburger** basis.
From bde we construct bc.

We have bc, bde (from be), $abcd$, $bcde$ and $abcde$. The set bc is the smallest but is not a successor.
We have bde (from be), $abcd$, $bcde$ and $abcde$. The set bde (from be) is the smallest but is not a successor.
We have $abcd$, $bcde$ and $abcde$. The logical closure of $abcd$ is updated to $abcde$ (and removed).
We have $bcde$ and $abcde$ (from abd). The set $bcde$ is one of the smallest and is a successor.
$bcde'' = abcde$ so $bcde \rightarrow abcde$ is added to the **Duquenne-Guigues** basis.
From $bcde$ we do not construct anything.

We have $abcde$. The set $abcde$ is one of the smallest and is a successor. $abcde = abcde$ and $abcde$ is an essential intent that has been constructed from abd. The PREDECESSOR algorithm finds that $ab \rightsquigarrow_I abcde$ so $ab \rightarrow_{1,0} abcde$ is added to the **Luxenburger** basis.

The algorithm ends, having computed the following Duquenne-Guigues basis

- $a \rightarrow ab$
- $bc \rightarrow bcd$

- $cd \rightarrow bcd$
- $be \rightarrow bde$
- $abd \rightarrow abcde$
- $bcde \rightarrow abcde$

along with the Luxenburger basis

- $\emptyset \rightarrow_{5,0.6} b$
- $\emptyset \rightarrow_{5,0.4} c$
- $\emptyset \rightarrow_{5,0.6} d$
- $\emptyset \rightarrow_{5,0.6} e$
- $b \rightarrow_{3,0.33} ab$
- $b \rightarrow_{3,0.66} bd$
- $c \rightarrow_{2,0.5} ce$
- $d \rightarrow_{3,0.66} de$
- $bd \rightarrow_{2,0.5} bcd$
- $bd \rightarrow_{2,0.6} bde$
- $ab \rightarrow_{1,0} abcde$

in the chosen order.

5.4 Discussion

The algorithm that we propose can compute both the Luxenburger and Duquenne-Guigues bases in a single run. Computing these two bases in order to obtain a minimal set of association rules using formal concept analysis is not a new idea as it has been discussed as early as 2001 (see [96]). In order to compute the bases, three steps are usually performed :

- The concept lattice is computed (or the iceberg concept lattice if the support has to be considered)
- The Duquenne-Guigues basis is computed from the concept lattice
- The Luxenburger basis is computed from the concept lattice

Our algorithm, however, computes everything in a single step. To the best of our knowledge, this has not been proposed before. Of course, a similar result could be obtained by modifying Next Closure to make it compute the predecessor of each intent. The most obvious method would be to apply Successor to intents that have not been generated by one of their subsets. But this would be slightly less efficient than our algorithm because Predecessor would be applied more often, resulting in a significant increase in the number of logical closures. More efficient methods to construct a spanning tree of the intents lattice during the execution of Next Closure could surely be thought of.

The properties of algorithm presented in Section 3.3 still hold. Most importantly, the order in which implications and partial implications are enumerated can still be chosen. Since Successor and Predecessor are polynomial and every element of Φ results in an element of the basis of associal rules, the algorithm has a polynomial delay.

Chapter 6

Future Work and Conclusion

Contents

6.1 Future Work

In our work, we supposed that the order in which the attribute sets are enumerated extends the inclusion order. Given that pseudo-intents are recursively defined, it seems natural to want to know all the subsets of a set before deciding whether it is a pseudo-intent. However, being able to efficiently recognize a pseudo-intent without knowing all its subsets would allow for different orders to be used and may help find algorithms with a better delay. While we do not claim to have solved the problem, we believe that we have started working on promising ideas.

As mentioned in Section 2.4.5, recognizing a pseudo-intent in the general case is coNP-complete. However, this general case means being able to tell whether a given set is a pseudo-intent with only a context and the set itself as inputs. It would be interesting to know if it is easier to tell whether a set is a pseudo-intent when we have additional information. Indeed, an enumeration being a process, some information has already been found when we are confronted with the problem of recognizing a pseudo-intent.

Here, we are interested in the problem of recognizing a pseudo-intent when we already know a non-empty subset $\mathcal{I}$ of the Duquenne-Guigues basis of the context. This case is the most common as only the first pseudo-intent is found in the absence of other information. Let us call $\Phi_{\mathcal{I}}$ the lattice of attribute sets closed under $\mathcal{I}(.)$. When $\mathcal{I} = \mathcal{B}$, the lattice $\Phi_{\mathcal{I}}$ is the lattice of intents.

Proposition 16 *Let A and B be two elements of $\Phi_{\mathcal{I}}$. The set B is a lower cover of A if and only if $B = A \setminus C$ with C minimal such that $\forall G \in Gen_{\mathcal{I}}(A)$, $G \cap C \neq \emptyset$.*

Proof Let $C \subseteq \mathcal{A}$ be minimal such that $\forall G \in Gen(A)$, $G \cap C \neq \emptyset$. For any attribute $i \in A \setminus C$, the set $(A \setminus C) \cup \{i\} = A \setminus (C \setminus \{i\})$ contains a minimal generator of A because of the minimality of C. Therefore, any B between $(A \setminus C)$ and A is such that $\mathcal{I}(B) = A$ and cannot be in $\Phi_{\mathcal{I}}$. Hence, $A \setminus C$ is a lower cover of A.

Let B be a lower cover of A in $\Phi_{\mathcal{I}}$. By definition, $\mathcal{I}(B \cup \{i\}) = A$ for any attribute $i \in A \setminus B$ so there is a subset C of A such that $i \in C$ and $(C \setminus \{i\}) \subseteq B$ that is a minimal generator of A. □

This proposition states that the lower covers of A in $\Phi_{\mathcal{I}}$ can be obtained from the minimal generators of A, which can themselves be obtained from A and $\mathcal{I}$.

Proposition 17 *An attribute set $A \in \Phi_{\mathcal{I}}$ is a pseudo-intent if it is not closed and all its lower covers are closed.*

Proof Let us suppose that A is not closed and that all its lower covers are closed. Let X be any subset of A and B a lower cover of A. If $B \subset X$, then $\mathcal{I}(X) = A$ and $X'' = A''$. If $X \subseteq B$ and $X'' \not\subseteq B$, then B cannot be closed and we have a contradiction. Therefore, $X'' \subseteq B$. Since the closure of every subset of A is either A'' or contained in A, the set A is a pseudo-intent. □

This proposition states that some pseudo-intents can be recognized in $\Phi_{\mathcal{I}}$ only by looking at their lower cover.

Definition 16 *The set of pseudo-intents recognizable from $\mathcal{I}$, noted $Rec(\mathcal{I})$, is composed of the elements of $\Phi_{\mathcal{I}}$ which lower covers are all closed.*

Proposition 18 $\forall \mathcal{I} \subset \mathcal{B}$, $Rec(\mathcal{I}) \neq \emptyset$

Proof Let P be minimal among pseudo-intents that are not premises of implications in $\mathcal{I}$. If there is a lower cover X of P in $\Phi_{\mathcal{I}}$ that is not closed, then there is a set A such that $A \to A''$ holds in the context. If $A'' \subseteq P$, then P is not minimal. If $A'' \not\subseteq P$, then P is not a pseudo-intent. Both cases lead to contradictions so all the lower covers of P are closed and P is recognizable from $\mathcal{I}$. Given that $\mathcal{I} \subset \mathcal{B}$, the set of pseudo-intents that are not a premise of $\mathcal{I}$ is not empty, so it has minimal elements. Hence, the set of pseudo-intents that are recognizable from $\mathcal{I}$ is not empty if $\mathcal{I} \neq \mathcal{B}$. □

At any step during the enumeration of pseudo-intents, there is a non-empty set of pseudo-intents that are recognizable from the already acquired implications. Let us suppose the following set of implications is the Duquenne-Guigues basis of some context $\mathcal{C} = (\mathcal{O}, \{a, c, b, d, e\}, \mathcal{R})$:

- $bc \to bcd$
- $cd \to bcd$
- $ae \to abde$
- $be \to bde$
- $bcde \to abcde$

Now, let us call $\mathcal{I}$ the set with the three first implications, $\{bc \to bcd, cd \to bcd, ae \to abde\}$. Figure 6.1 shows the $\Phi_{\mathcal{I}}$ lattice. Since we know the Duquenne-Guigues basis, we can see that the following sets, in particular, are closed : b, e, bcd, bde. This implies that $Rec(\mathcal{I}) = \{be, bcde\}$. If we were to know an algorithm that uses Proposition 17 to recognize a pseudo-intent, it would be possible to find $bcde$ before be, thus bypassing the inclusion order, because $bc \to bcd$ and $cd \to bcd$ give us enough information on $bcde$ and its lower covers. Similarly, it would be possible to find $bcde$ before bc if we knew of the pseudo-intents be and cd but $bcde$ would not be recognizable from only bc and be. This illustrates that the order of enumeration would play a much greater role in this kind of algorithm.

While we think that finding algorithms that can recognize pseudo-intents from subsets of the Duquenne-Guigues basis is a promising idea, using Proposition 17 directly would not be efficient as the number of lower covers of a set in $\Phi_{\mathcal{I}}$ can be exponential in the size of $\mathcal{I}$. But we intend to investigate this lead more thoroughly in the future.

6.2 Conclusion

Initially centered around the use of formal concept analysis to learn terminological axioms in description logics, our work ventured into the construction of the Duquenne-Guigues basis in the general case. We started by developing an algorithm that enumerates the elements of the search space Φ breadth-first, which corresponds to the increasing cardinality order presented in Section 3.3.2 once the constraint of

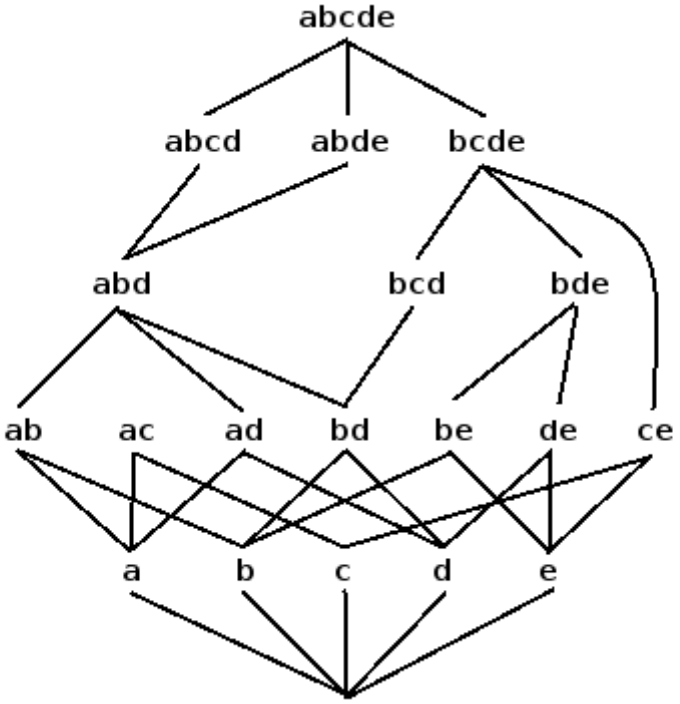


Figure 6.1: $\Phi_{\{bc \to bcd, cd \to bcd, ae \to abde\}}$

respecting the inclusion is added. From there, we saw that we could generalize the algorithm to allow an enumeration in any order, as long as it extends the inclusion order. Because of this generality and our desire to decrease the number of operations, we chose to have attribute sets constructed only from one of their subsets, which differs from algorithms the likes of Next Closure. While it would have been possible to construct sets from any other set by using another canonicity test, we felt that the inclusion relation allowed us to transmit the result of computations done on sets to their successors, thereby reducing the amount of redundant operations.

We empirically compared our algorithms with Next Closure on randomly generated contexts using the number of logical closures performed as a metric. We chose this metric instead of the runtime because the number of operations is independent of the platform, language, data structures and non-optimal coding. Results showed that our algorithms performed slightly less logical closures on sparse contexts in which objects were described by less than 33% of the available attributes on average. On denser contexts, Next Closure appeared to outperform our algorithm on the chosen metric. We then empirically studied the effect of the order of enumeration on the space complexity of our algorithms. We enumerated the pseudo-intents of randomly generated contexts using the lectic order, the reverse lectic order and the increasing cardinality order and considered the evolution of the number of attribute sets simultaneously stored in memory during the course of the algorithm. Results showed clear differences in the memory consumptions of the three orders, with the lectic order appearing to be the most efficient. We conjectured that the memory consumption of an order is function of its distance to the lectic order because of the canonicity test being based on it. However, the tests used the general version of the

algorithm and optimizations can be thought for each order, which could change the results. Further research has thus to be done on the subject.

Using our general-case algorithm, we returned to our initial problem with relational data in general and description logics in particular. We showed that the flexibility with the orders allowed us to apply our method to relational contexts in the same fashion as Next Closure. Therefore, all our results on traditional contexts also apply to relational contexts. We discussed the type of relational information that could be represented using our method and found that all knowledge of the form "being in relation with at least n objects that are..." could be treated.

Finally, we applied our algorithm to the domain of association rules mining. Implications being certain association rules, this seemed natural. We showed that, by only slightly modifying our algorithm, we were able to compute the Luxenburger basis of a context alongside the Duquenne-Guigues basis. As such, we showed that our algorithm can directly compute a basis of association rules.

Concerning the complexity of enumerating pseudo-intents, we feel that the lack of results on non-lectic orders allows for the possibility of a polynomial-delay algorithm. Next Closure only enumerates in the lectic order and attribute-incremental and divide-and-conquer approaches do not allow for the order to be finely tuned. Because of this, we believe that our work is a step in the right direction as it provides an algorithm that enumerates in any order that respects the inclusion relation. While we do not claim to have a polynomial delay, we think that the further study of the different orders is promising.

Bibliography

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Bocca et al. [24], pages 487–499.

[2] S. Andrews. In-close, a fast algorithm for computing formal concepts. In *International Conference on Conceptual Structures (ICCS)*, January 2009. Final version of paper accepted (via peer review) for the International Conference on Conceptual Structures (ICCS) 2009, Moscow.

[3] Franz Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In Gottlob and Walsh [55], pages 319–324.

[4] Franz Baader, Gerhard Brewka, and Thomas Eiter, editors. *KI 2001: Advances in Artificial Intelligence, Joint German/Austrian Conference on AI, Vienna, Austria, September 19-21, 2001, Proceedings*, volume 2174 of *Lecture Notes in Computer Science*. Springer, 2001.

[5] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[6] Franz Baader and Felix Distel. A finite basis for the set of $\mathcal{EL}$-implications holding in a finite model. In Medina and Obiedkov [79], pages 46–61.

[7] Franz Baader and Felix Distel. Exploring finite models in the description logic $\mathcal{EL}_{gfp}$. In Ferré and Rudolph [45], pages 146–161.

[8] Franz Baader, Bernhard Ganter, Baris Sertkaya, and Ulrike Sattler. Completing description logic knowledge bases using formal concept analysis. In Veloso [102], pages 230–235.

[9] Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumers in description logics with existential restrictions. In Dean [34], pages 96–103.

[10] Franz Baader and Barbara Morawska. Unification in the description logic $\mathcal{EL}$. In Treinen [100], pages 350–364.

[11] Franz Baader and Baris Sertkaya. Applying formal concept analysis to description logics. In Eklund [41], pages 261–286.

[12] Franz Baader and Barış Sertkaya. Usability issues in description logic knowledge base completion. In *Formal Concept Analysis*, pages 1–21. Springer, 2009.

[13] Mikhail A Babin and Sergei O Kuznetsov. On links between concept lattices and related complexity problems. In *Formal Concept Analysis*, pages 138–144. Springer, 2010.

[14] Mikhail A. Babin and Sergei O. Kuznetsov. Recognizing pseudo-intents is conp-complete. In Kryszkiewicz and Obiedkov [65], pages 294–301.

[15] Michael Bain. Inductive construction of ontologies from formal concept analysis. In Gedeon and Fung [52], pages 88–99.

[16] Jaume Baixeries, Laszlo Szathmary, Petko Valtchev, and Robert Godin. Yet a faster algorithm for building the hasse diagram of a concept lattice. In *Formal Concept Analysis*, pages 162–177. Springer, 2009.

[17] Konstantin Bazhanov and Sergei A. Obiedkov. Comparing performance of algorithms for generating the Duquenne-Guigues basis. In Napoli and Vychodil [82], pages 43–57.

[18] Catriel Beeri and Philip A Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Transactions on Database Systems (TODS)*, 4(1):30–59, 1979.

[19] Sadok Ben Yahia and Engelbert Mephu Nguifo. Approches d'extraction de règles d'association basées sur la correspondance de Galois. *Ingénierie des systèmes d'information*, 9(3-4):23–55, 2004.

[20] V. Richard Benjamins, Mathieu d'Aquin, and Andrew Gordon, editors. *Proceedings of the 7th International Conference on Knowledge Capture, K-CAP 2013, Banff, Canada, June 23-26, 2013*. ACM, 2013.

[21] Karell Bertet and Bernard Monjardet. The multiple facets of the canonical direct unit implicational basis. *Theoretical Computer Science*, 411(22):2155–2166, 2010.

[22] Karell Bertet and Mirabelle Nebut. Efficient algorithms on the Moore family associated to an implicational system. *Discrete Mathematics & Theoretical Computer Science*, 6(2):315–338, 2004.

[23] Garrett Birkhoff. *Lattice theory*, volume 25. American Mathematical Society New York, 1948.

[24] Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors. *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*. Morgan Kaufmann, 1994.

[25] Daniel Borchmann. A general form of attribute exploration. *CoRR*, abs/1202.4824, 2012.

[26] Daniel Borchmann. A generalized next-closure algorithm - enumerating semilattice elements from a generating set. In Szathmary and Priss [98], pages 9–20.

[27] Daniel Borchmann. Axiomatizing $\mathcal{EL}^{\perp}$-expressible terminological knowledge from erroneous data. In Benjamins et al. [20], pages 1–8.

[28] Daniel Borchmann. Towards an error-tolerant construction of\ mathcal {EL}ˆ\ bot-ontologies from data using formal concept analysis. In *Formal Concept Analysis*, pages 60–75. Springer, 2013.

[29] Daniel Borchmann and Felix Distel. Mining of el-gcis. In Spiliopoulou et al. [94], pages 1083–1090.

[30] Loïc Cerf, Jérémy Besson, Céline Robardet, and Jean-François Boulicaut. Closed patterns meet n-ary relations. *TKDD*, 3(1), 2009.

[31] Vicky Choi. Faster algorithms for constructing a concept (galois) lattice. *CoRR*, abs/cs/0602069, 2006.

[32] William W Cohen and Haym Hirsh. The learnability of description logics with equality constraints. *Machine Learning*, 17(2-3):169–199, 1994.

[33] Luc De Raedt. The logic of learning. In *Formal Concept Analysis*, pages 57–57. Springer, 2009.

[34] Thomas Dean, editor. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*. Morgan Kaufmann, 1999.

[35] Felix Distel. Model-based most specific concepts in some inexpressive description logics. In Grau et al. [56].

[36] Felix Distel. *Learning description logic knowledge bases from data using methods from formal concept analysis*. PhD thesis, 2011.

[37] Felix Distel. Some complexity results about essential closed sets. In *Formal Concept Analysis*, pages 81–92. Springer, 2011.

[38] Felix Distel and Barış Sertkaya. On the complexity of enumerating pseudo-intents. *Discrete Applied Mathematics*, 159(6):450–466, 2011.

[39] Ding-Zhu Du and Ming Li, editors. *Computing and Combinatorics, First Annual International Conference, COCOON '95, Xi'an, China, August 24-26, 1995, Proceedings*, volume 959 of *Lecture Notes in Computer Science*. Springer, 1995.

[40] Vincent Duquenne. The core of finite lattices. *Discrete Mathematics*, 88(2):133–147, 1991.

[41] Peter W. Eklund, editor. *Concept Lattices, Second International Conference on Formal Concept Analysis, ICFCA 2004, Sydney, Australia, February 23-26, 2004, Proceedings*, volume 2961 of *Lecture Notes in Computer Science*. Springer, 2004.

[42] Peter W. Eklund, Jean Diatta, and Michel Liquiere, editors. *Proceedings of the Fifth International Conference on Concept Lattices and Their Applications, CLA 2007, Montpellier, France, October 24-26, 2007*, volume 331 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[43] Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. Dl-foil concept learning in description logics. In Zelezný and Lavrac [108], pages 107–121.

[44] Sébastien Ferré and Olivier Ridoux. A logical generalization of formal concept analysis. In Ganter and Mineau [48], pages 371–384.

[45] Sébastien Ferré and Sebastian Rudolph, editors. *Formal Concept Analysis, 7th International Conference, ICFCA 2009, Darmstadt, Germany, May 21-24, 2009, Proceedings*, volume 5548 of *Lecture Notes in Computer Science*. Springer, 2009.

[46] Jean-Gabriel Ganascia. Charade: A rule system learning system. In McDermott [78], pages 345–347.

[47] Bernhard Ganter. Two basic algorithms in concept analysis. In Kwuida and Sertkaya [73], pages 312–340.

[48] Bernhard Ganter and Guy W. Mineau, editors. *Conceptual Structures: Logical, Linguistic, and Computational Issues, 8th International Conference on Conceptual Structures, ICCS 2000, Darmstadt, Germany, August 14-18, 2000, Proceedings*, volume 1867 of *Lecture Notes in Computer Science*. Springer, 2000.

[49] Bernhard Ganter and Klaus Reuter. Finding all closed sets: A general approach. *Order*, 8(3):283–290, 1991.

[50] Bernhard Ganter, Gerd Stumme, and Rudolf Wille, editors. *Formal Concept Analysis, Foundations and Applications*, volume 3626 of *Lecture Notes in Computer Science*. Springer, 2005.

[51] Bernhard Ganter, Rudolf Wille, and Cornelia Franzke. *Formal concept analysis: mathematical foundations*. Springer-Verlag New York, Inc., 1997.

[52] Tamás D. Gedeon and Lance Chun Che Fung, editors. *AI 2003: Advances in Artificial Intelligence, 16th Australian Conference on Artificial Intelligence, Perth, Australia, December 3-5, 2003, Proceedings*, volume 2903 of *Lecture Notes in Computer Science*. Springer, 2003.

[53] Alain Gély, Raoul Medina, and Lhouari Nourine. About the enumeration algorithms of closed sets. In *Formal Concept Analysis*, pages 1–16. Springer, 2010.

[54] Alain Gély, Raoul Medina, Lhouari Nourine, and Yoan Renaud. Uncovering and reducing hidden combinatorics in Guigues-Duquenne bases. In *Formal Concept Analysis*, pages 235–248. Springer, 2005.

[55] Georg Gottlob and Toby Walsh, editors. *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*. Morgan Kaufmann, 2003.

[56] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors. *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[57] Jean-Louis Guigues and Vincent Duquenne. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences humaines*, 95:5–18, 1986.

[58] Michel Habib, Raoul Medina, Lhouari Nourine, and George Steiner. Efficient algorithms on distributive lattices. *Discrete Applied Mathematics*, 110(2):169–187, 2001.

[59] Mohamed Rouane Hacene, Marianne Huchard, Amedeo Napoli, and Petko Valtchev. A proposal for combining formal concept analysis and description logics for mining relational data. In Kuznetsov and Schmidt [71], pages 51–65.

[60] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.

[61] Miki Hermann and Baris Sertkaya. On the complexity of computing generators of closed sets. In Medina and Obiedkov [79], pages 158–168.

[62] Pascal Hitzler, Thomas Roth-Berghofer, and Sebastian Rudolph, editors. *Foundations of Artificial Intelligence FAInt 2007, Osnabrück, Germany, September 10, 2007*, volume 277 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[63] Bernhard Hollunder, Werner Nutt, and Manfred Schmidt-Schauß. Subsumption algorithms for concept description languages. In *ECAI*, pages 348–353, 1990.

[64] Petr Krajca, Jan Outrata, and Vilém Vychodil. Advances in algorithms based on cbo. In Kryszkiewicz and Obiedkov [65], pages 325–337.

[65] Marzena Kryszkiewicz and Sergei A. Obiedkov, editors. *Proceedings of the 7th International Conference on Concept Lattices and Their Applications, Sevilla, Spain, October 19-21, 2010*, volume 672 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[66] Sergei O Kuznetsov. Learning of simple conceptual graphs from positive and negative examples. In *Principles of Data Mining and Knowledge Discovery*, pages 384–391. Springer, 1999.

[67] Sergei O Kuznetsov. Machine learning and formal concept analysis. In *Concept Lattices*, pages 287–312. Springer, 2004.

[68] Sergei O Kuznetsov. On the intractability of computing the Duquenne-Guigues basis. *J. UCS*, 10(8):927–933, 2004.

[69] Sergei O Kuznetsov and Sergei Obiedkov. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental and Theoretical Artificial Intelligence*, 14:189–216, 2002.

[70] Sergei O. Kuznetsov and Sergei A. Obiedkov. Counting pseudo-intents and #p-completeness. In Missaoui and Schmid [80], pages 306–308.

[71] Sergei O. Kuznetsov and Stefan Schmidt, editors. *Formal Concept Analysis, 5th International Conference, ICFCA 2007, Clermont-Ferrand, France, February 12-16, 2007, Proceedings*, volume 4390 of *Lecture Notes in Computer Science*. Springer, 2007.

[72] Léonard Kwuida. When is a concept algebra boolean? In *Concept Lattices*, pages 142–155. Springer, 2004.

[73] Léonard Kwuida and Baris Sertkaya, editors. *Formal Concept Analysis, 8th International Conference, ICFCA 2010, Agadir, Morocco, March 15-18, 2010. Proceedings*, volume 5986 of *Lecture Notes in Computer Science*. Springer, 2010.

[74] Lotfi Lakhal and Gerd Stumme. Efficient mining of association rules based on formal concept analysis. In Ganter et al. [50], pages 180–195.

[75] Patrick Lambrix and Jalal Maleki. Learning composite concepts in description logics: A first step. In *Foundations of Intelligent Systems*, pages 68–77. Springer, 1996.

[76] Man Li, Xiao-Yong Du, and Shan Wang. Learning ontology from relational database. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 6, pages 3410–3415. IEEE, 2005.

[77] Michael Luxenburger. Implications partielles dans un contexte. *Mathématiques, informatique et sciences humaines*, (113):35–55, 1991.

[78] John P. McDermott, editor. *Proceedings of the 10th International Joint Conference on Artificial Intelligence. Milan, Italy, August 1987*. Morgan Kaufmann, 1987.

[79] Raoul Medina and Sergei A. Obiedkov, editors. *Formal Concept Analysis, 6th International Conference, ICFCA 2008, Montreal, Canada, February 25-28, 2008, Proceedings*, volume 4933 of *Lecture Notes in Computer Science*. Springer, 2008.

[80] Rokia Missaoui and Jürg Schmid, editors. *Formal Concept Analysis, 4th International Conference, ICFCA 2006, Dresden, Germany, February 13-17, 2006, Proceedings*, volume 3874 of *Lecture Notes in Computer Science*. Springer, 2006.

[81] Bernard Monjardet. Arrowian characterizations of latticial federation consensus functions. *Mathematical Social Sciences*, 20(1):51–71, 1990.

[82] Amedeo Napoli and Vilém Vychodil, editors. *Proceedings of The Eighth International Conference on Concept Lattices and Their Applications, Nancy, France, October 17-20, 2011*, volume 959 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

[83] Kamal Nehmé, Petko Valtchev, Mohamed H Rouane, and Robert Godin. On computing the minimal generator family for concept lattices and icebergs. In *Formal Concept Analysis*, pages 192–207. Springer, 2005.

[84] Sergei Obiedkov and Vincent Duquenne. Attribute-incremental construction of the canonical implication basis. *Annals of Mathematics and Artificial Intelligence*, 49(1-4):77–99, 2007.

[85] Nicolas Pasquier. Mining association rules using frequent closed itemsets. *Encyclopedia of Data Warehousing and Mining*, 2005.

[86] Nicolas Pasquier, Rafik Taouil, Yves Bastide, Gerd Stumme, and Lotfi Lakhal. Generating a condensed representation for association rules. *Journal of Intelligent Information Systems*, 24(1):29–60, 2005.

[87] Sebastian Rudolph. Exploring relational structures via $\mathcal{FLE}$. In Wolff et al. [106], pages 196–212.

[88] Sebastian Rudolph. *Relational exploration: combining description logics and formal concept analysis for knowledge specification.* PhD thesis, 2006.

[89] Sebastian Rudolph. Relational exploration - reconciling Plato and Aristotle. In Hitzler et al. [62].

[90] Sebastian Rudolph. Some notes on pseudo-closed sets. In *Formal Concept Analysis*, pages 151–165. Springer, 2007.

[91] Anja Rusch and Rudolf Wille. *Knowledge spaces and formal concept analysis.* Springer, 1996.

[92] Baris Sertkaya. *Formal concept analysis methods for description logics.* PhD thesis, 2007.

[93] Nikolay V. Shilov and Sang-Yong Han. A proposal of description logic on concept lattices. In Eklund et al. [42].

[94] Myra Spiliopoulou, Haixun Wang, Diane J. Cook, Jian Pei, Wei Wang, Osmar R. Zaïane, and Xindong Wu, editors. *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on, Vancouver, BC, Canada, December 11, 2011.* IEEE, 2011.

[95] Gerd Stumme. Attribute exploration with background implications and exceptions. In H.-H. Bock and W. Polasek, editors, *Data Analysis and Information Systems. Statistical and Conceptual approaches. Proc. GfKl'95. Studies in Classification, Data Analysis, and Knowledge Organization 7*, pages 457–469, Heidelberg, 1996. Springer.

[96] Gerd Stumme, Rafik Taouil, Yves Bastide, Nicolas Pasquier, and Lotfi Lakhal. Intelligent structuring and reducing of association rules with formal concept analysis. In Baader et al. [4], pages 335–350.

[97] Gerd Stumme, Rafik Taouil, Yves Bastide, Nicolas Pasquier, and Lotfi Lakhal. Computing iceberg concept lattices with TITANIC. *Data & knowledge engineering*, 42(2):189–222, 2002.

[98] Laszlo Szathmary and Uta Priss, editors. *Proceedings of The Ninth International Conference on Concept Lattices and Their Applications, Fuengirola (Málaga), Spain, October 11-14, 2012*, volume 972 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[99] Rafik Taouil, Yves Bastide, et al. Computing proper implications. In *9th International Conference on Conceptual Structures: Broadening the Base-ICCS'2001*, 2001.

[100] Ralf Treinen, editor. *Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings*, volume 5595 of *Lecture Notes in Computer Science*. Springer, 2009.

[101] Petko Valtchev and Vincent Duquenne. On the merge of factor canonical bases. In Medina and Obiedkov [79], pages 182–198.

[102] Manuela M. Veloso, editor. *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2007.

[103] Marcel Wild. Computations with finite closure systems and implications. In Du and Li [39], pages 111–120.

[104] Rudolf Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In *Ordered Sets*, pages 445–470. Springer, 1982.

[105] Rudolf Wille. Mathematics presenting, reflecting, judging. In *Formal Concept Analysis*, pages 17–33. Springer, 2010.

[106] Karl Erich Wolff, Heather D. Pfeiffer, and Harry S. Delugach, editors. *Conceptual Structures at Work: 12th International Conference on Conceptual Structures, ICCS 2004, Huntsville, AL, USA, July 19-23, 2004. Proceedings*, volume 3127 of *Lecture Notes in Computer Science*. Springer, 2004.

[107] Mohammed Javeed Zaki and Mitsunori Ogihara. Theoretical foundations of association rules. In *3rd ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 71–78. Citeseer, 1998.

[108] Filip Železný and Nada Lavrac, editors. *Inductive Logic Programming, 18th International Conference, ILP 2008, Prague, Czech Republic, September 10-12, 2008, Proceedings*, volume 5194 of *Lecture Notes in Computer Science*. Springer, 2008.

Index