Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Méthodes symboliques de fouille de données avec la plate-forme Coron

THÈSE

présentée et soutenue publiquement le 24 novembre 2006

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Laszlo SZATHMARY

Composition du jury

Président :	Claude GODART	Professeur, UHP Nancy 1, France
Rapporteurs :	Bruno CRÉMILLEUX	Professeur, Université de Caen, France
	Sergei O. KUZNETSOV	Professor, Higher School of Economics, Moscow, Russia
Examinateurs :	Katalin BOGNÁR	Associate professor, University of Debrecen, Hungary
	Marzena KRYSZKIEWICZ	Associate professor, Warsaw Univ. of Technology, Poland
	Amedeo NAPOLI	Directeur de recherche CNRS, UHP Nancy 1, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Remerciements

En premier lieu, je remercie vivement Amedeo Napoli qui m'a proposé d'effectuer une thèse sous sa direction. Travailler avec Amedeo m'a permis d'apprendre énormément, sur le plan scientifique, sur le plan humain, ainsi que sur le métier de chercheur.

Je tiens à remercier les personnes qui m'ont fait l'honneur de participer à mon jury de thèse et de s'intéresser à ce travail. Merci à Claude Godart d'avoir présidé ce jury. Un grand merci à Bruno Crémilleux et Sergei O. Kuznetsov d'avoir accepté d'être rapporteur et d'avoir évalué mon travail avec autant d'intérêt et d'enthousiasme. Je remercie également beaucoup les examinateurs qui ont composé mon jury : Katalin Bognár et Marzena Kryszkiewicz. Merci pour l'attention avec laquelle elles ont lu et évalué ce mémoire, pour leurs encouragements ainsi que pour les remarques et critiques constructives qu'elles m'ont adressées.

Je souhaite remercier tous les membres de l'équipe Orpailleur pour leur acceuil, leur soutien, leurs conseils et pour tout ce qui fait de cette équipe un lieu où il est très agréable et profitable de travailler. Merci en particulier à Sandy Maumus, avec qui c'était un vrai plaisir de travailler sur notre projet commun. Merci également à mes camarades de bureau, Mathieu d'Aquin et Sylvain Tenier, pour m'avoir supporté pendant tout ce temps. Merci à mes stagiaires, Thomas Bouton et Pierre Petronin, pour leur contribution à mon travail.

Antoinette Courrier et Nadine Beurné m'ont apporté une aide très efficace et très sympathique.

Merci aux autres thésards du LORIA avec qui j'ai partagé tant de discussions, tant de cafés et tant de baby-foot (Mohamed, Fred, Raghav, Daniel, Rokia, Yves, Nizar, Adrien, Fadi, Hatem, Szilárd, Ustun, Rémi, Benjamin, ...). Merci aussi aux autres copains de Nancy : Ola, Jean-Daniel, Alan, Sébastien, ...

Je voudrais remercier aussi Petko Valtchev de l'Université du Québec à Montréal pour sa très enrichissante collaboration aux travaux de l'équipe.

Je souhaite également remercier madame Demangel et madame Hans au SIUAP pour m'avoir introduit dans le monde du rock & roll et de la salsa. Grâce à la danse, le temps de la rédaction est passé beaucoup plus facilement. Merci à tout le monde au SIUAP, surtout à Sophie et Anne-Laure.

Je tiens également à remercier ma famille et tout particulièrement ma mère pour m'avoir toujours fait confiance et toujours soutenu dans mes études.

Je remercie aussi tous ceux que j'ai oublié de remercier !

Finalement, je voudrais remercier le travail des développeurs des logiciels suivants : Debian GNU/Linux, K Desktop Environment (KDE), vim, LaTeX, gv, xpdf, GALICIA, Java, Eclipse, Microsoft Windows XP, Microsoft Visio 2000, ...

László Szathmáry

For my mother.
For my grandparents.

Table of Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**

1.1 Knowledge Discovery in Databases 1

1.2 Data Mining – The Central Step of KDD 3

1.3 Methods for KDD 3

1.3.1 An Introducting Example 3

1.3.2 Data Mining Methods 4

1.4 Overview of the Thesis 5

1.4.1 Frequent Itemset Search 5

1.4.2 Frequent Association Rules 6

1.4.3 Rare Itemsets and Rare Association Rules 6

1.4.4 The Coron Toolkit 6

1.5 Thesis Outline 7

2 Symbolic Methods for Knowledge Discovery **9**

2.1 Lattice-Based Classification 9

2.1.1 Classical Notation of Formal Concept Analysis 13

2.2 Frequent Itemset Search and Association Rule Extraction 15

2.2.1 Frequent Itemset Search 15

2.2.2 Association Rule Extraction 16

2.3 Applications 18

2.3.1 Mining Chemical Reaction Database 18

2.3.2 An Experiment in Biology 22

2.3.3 An Introduction to Web Mining 24

2.4 Discussion 25

2.5 Conclusion 26

3 Frequent Itemset Search **29**
3.1 Basic Concepts . . . 29
3.2 Classification of Itemset Mining Algorithms . . . 30
3.2.1 Levelwise Algorithms . . . 30
3.2.2 Vertical Algorithms . . . 33
3.2.3 Hybrid Algorithms . . . 35
3.2.4 Other Algorithms . . . 35
3.3 Detailed Description of Selected Algorithms . . . 36
3.3.1 Zart . . . 37
3.3.2 Eclat-Z . . . 51
3.3.3 Charm-MFI . . . 58

4 Frequent Association Rules **61**
4.1 All Association Rules . . . 61
4.2 Closed Association Rules . . . 64
4.3 Family of "Minimal Non-Redundant Association Rules" . . . 68
4.3.1 Why MNR Rules are "Minimal" and "Non-Redundant" . . . 69
4.3.2 Deducing Valid Association Rules from MNR Rules . . . 72
4.4 Other Statistical Measures . . . 72
4.4.1 Interestingness Measures . . . 72
4.4.2 Combining Quality Measures . . . 74
4.4.3 Example . . . 75
4.5 Efficient Support Derivation of Frequent Itemsets . . . 75
4.6 Experimental Results . . . 77
4.6.1 Number of Rules . . . 77
4.6.2 Execution Times of Rule Generation . . . 78
4.7 Related Work . . . 80
4.8 Conclusion . . . 80

5 Rare Itemsets and Rare Association Rules **81**
5.1 Rare Itemsets . . . 81
5.1.1 Contribution and Motivations . . . 82
5.1.2 Basic Concepts . . . 82
5.1.3 A Lattice-Based Approach for Itemset Enumeration . . . 83
5.1.4 Finding Rare Itemsets . . . 85
5.2 Border Study . . . 88
5.2.1 Minimal Rare Generators . . . 90

5.3 Rare Association Rules 93
5.3.1 Introduction 93
5.3.2 Basic Concepts 93
5.3.3 Related Work 93
5.3.4 Contribution and Motivations 95
5.3.5 Finding MRG Association Rules 96
5.3.6 Generating MRG Rules 98
5.3.7 Calculating Other Interestingness Measures 99
5.3.8 A Methodology for Finding Rare Association Rules 100
5.3.9 A Classification of Association Rules by the Frequency of the Antecedent and Consequent 100
5.3.10 Experimental Results 101
5.3.11 Conclusion 104

6 A Data Mining Methodology and the Coron Toolkit 105
6.1 An Overview of the Coron System 105
6.2 A Global Data Mining Methodology 107
6.2.1 Definition of the Study Framework 108
6.2.2 Iterative Step 108
6.2.3 Using the Methodology with Rare Association Rules 109
6.3 Experiments on a Real-Life Biological Database 109
6.3.1 Introduction to Mining Biological Data 109
6.3.2 Studied Population: the STANISLAS Cohort 110
6.3.3 Experiments and Results 111
6.4 Other Uses of the Coron System 114
6.4.1 The Cabamaka System 114
6.4.2 The Galicia Platform 119

7 Conclusion and Perspectives 121
7.1 Conclusion 121
7.1.1 Algorithms for Finding MNR Rules 121
7.1.2 Closed Association Rules 122
7.1.3 Rare Itemsets and Rare Association Rules 122
7.1.4 The Coron Toolkit 123
7.2 Mid- and Long-Term Perspectives 124

8 Summaries in French and in Hungarian **127**
8.1 Résumé étendu 127
8.1.1 Introduction 127
8.1.2 Une méthodologie pour la fouille et la boîte à outils Coron 134
8.1.3 L'extraction de motifs rares 143
8.1.4 Conclusion et perspectives 151
8.2 Rezümé 156
8.2.1 Konklúzió 156
8.2.2 MNR szabályokat kereső algoritmusok 156
8.2.3 Zárt asszociációs szabályok 156
8.2.4 Ritka minták és ritka asszociációs szabályok 157
8.2.5 A Coron rendszer 158

A Test Environment **159**

B Classical Itemset Mining Algorithms **161**
B.1 Apriori 161
B.2 Apriori-Close 164
B.3 Eclat 167
B.4 Charm 172

C The Trie Data Structure **179**
C.1 The Trie Data Structure 179
C.2 Subset Function with a Trie 180
C.3 Superset Function with a Trie 182

D Horizontal and Vertical Data Layouts **187**

E Efficient Support Count of 2-itemsets **189**

F Comparison of Generators Representation and Frequent Closed Itemset Representation **191**

G The Coron Toolkit **193**
G.1 Coron-base 193
G.1.1 Command-line Interface 193
G.1.2 Available Algorithms 196
G.1.3 Graphical User Interface 200
G.2 AssRuleX 204

G.2.1 Command-line Interface 204
G.2.2 Graphical User Interface 208
G.3 Pre-processing: Data Preparation and Filtering 210
G.3.1 Filter-DB 210
G.3.2 Filter-Compl 213
G.4 Post-processing of the Extracted Rules 214
G.4.1 Filter-Rules 214
G.4.2 Rule-Coloring 215
G.4.3 RuleMiner: a GUI for Post-processing Modules 216
G.5 Other Utilities 218
G.5.1 Analyze-Itemsets 218
G.5.2 Analyze-Rules 219
G.5.3 getInterval 220
G.5.4 rules2sql 221
G.6 LeCo 223
G.7 Using the Coron API 226

Bibliography **229**

Glossary **241**

Index **243**

List of Figures

1.1 The KDD loop: from rough data to knowledge units. The overall objective of the KDD process is to select, prepare and extract knowledge units from different sources, and then to represent the extracted knowledge units in adequate knowledge structures. 2

2.1 The lattice representing the power set of the set {a,b,c,d,e}. 10
2.2 The lattice representing a part of the divisibility relation in ℕ. 10
2.3 The Galois lattice associated to the formal context of Figure 2.1. 12
2.4 The general schema of a synthesis problem. 18
2.5 Skeleton and functional groups of a target molecule. 19
2.6 The structural information on a reaction with the associated atom-to-atom mapping (reaction #13426 in the JSM-2002 database). 19
2.7 The representation of the reaction #13426 in the JSM-2002 database at the block level. 20

3.1 Powerset lattice of database D (Table 3.1). 31
3.2 IT-tree: Itemset-Tidset search tree of dataset D (Table 3.1). 34
3.3 Result of *Pascal* on D (Table 3.1) with $min\_supp = 2$ (40%). 38
3.4 Result of *Zart* on D (Table 3.1) with $min\_supp = 2$ (40%). Note that information on the subsumption relation (see Def. 3.5) is provided by the trie data structure. 38
3.5 Response times of *Zart* for T20I6D100K. 50
3.6 Response times of *Zart* for C20D10K. 50
3.7 Response times of *Zart* for MUSHROOMS. 50
3.8 The PosIndex structure of *Eclat-Z*. Timeline (left) and final state (right). 53
3.9 Contents of the file with the FIs. File positions are also indicated. 53

4.1 Position of Closed Rules. 66
4.2 Non-optimized and optimized support derivation of frequent itemsets. 76

5.1 Powerset lattice of dataset D (Table 3.1). 83
5.2 Rare equivalence classes found by *BtB* at different min_supp values. 98

6.1 Life-cycle of the mining methodology. 107

8.1 La boucle d'ECBD : des données brutes aux unités de connaissance. L'objectif du processus d'ECBD est de sélectionner, de préparer et d'extraire des unités de connaissances depuis différents sources, puis de représenter les unités de connaissances dans des structures appropriées. 128
8.2 Cycle de vie de la méthodologie de fouille symbolique de données. 135

8.3 Treillis des parties de la base de données D (voir Tableau 8.3). 146

B.1 Response times of *Apriori-Close* for T20I6D100K. 166
B.2 Response times of *Apriori-Close* for C20D10K. 166
B.3 Response times of *Apriori-Close* for MUSHROOMS. 166
B.4 Execution of *Eclat* on dataset D with $min\_supp = 3$ (60%). 169
B.5 Response times of *Eclat* for T20I6D100K. 170
B.6 Response times of *Eclat* for C20D10K. 171
B.7 Response times of *Eclat* for MUSHROOMS. 171
B.8 Execution of *Charm* on dataset D' with $min\_supp = 3$ (50%). 174
B.9 Hash table for the IT-tree in Figure B.8. 175
B.10 Response times of *Charm* for T20I6D100K. 176
B.11 Response times of *Charm* for C20D10K. 177
B.12 Response times of *Charm* for MUSHROOMS. 177

C.1 Trie of dictionary words. 179
C.2 Trie of itemsets. 180
C.3 Subset function with a trie. 181
C.4 Superset function with a trie. 185

D.1 Horizontal and vertical layouts of dataset D (Table 3.1). 187
D.2 Horizontal to vertical database transformation. 188
D.3 Vertical to horizontal database transformation. 188

E.1 Initialized upper triangular matrix for counting the support of 2-itemsets. 189
E.2 Support count of 2-itemsets of dataset D (Table 3.1) with an upper triangular matrix. 190

G.1 Step 0 – Welcome screen. 200
G.2 Step 1 – Choosing the input file. 201
G.3 Step 2 – Choosing the output file. 201
G.4 Step 3 – Defining the minimum support. 202
G.5 Step 4 – Choosing the mining algorithm. 202
G.6 Step 5 – Statistics of the user's choice. 203
G.7 Step 6 – Displaying the result. 203
G.8 Step 3 – Defining the minimum support and minimum confidence thresholds. . . 208
G.9 Step 4 – Choosing the mining algorithm and rules to extract. 209
G.10 Step 6 – Displaying the result. 209
G.11 Horizontal filtering. 212
G.12 Vertical filtering. 212
G.13 Graphical frontend for Filter-Rules. 216
G.14 Graphical frontend for Rule-Coloring. 217
G.15 Filtering rules by support and/or confidence. 217

List of Tables

1.1 An example of a binary matrix representing transactions between customers (C) and products (P). 4

2.1 An example of formal context. 11
2.2 The original data are prepared for the mining task: the binary transformation of the data can be done without taking into account the atom mapping, i.e. one single line in the binary table, or by taking into account the atom mapping, i.e. two lines in the table. 21

3.1 A toy dataset (D) for the examples. 31
3.2 Tables used in *Zart*. 43
3.3 Fields of the tables of *Zart*. 43
3.4 Execution of *Zart* on dataset D with $min\_supp = 2$ (40%). 46
3.5 Output of *Zart*. 47
3.6 Characteristics of databases used for *Zart*. 48
3.7 Response times of *Zart* and other statistics. 49
3.8 Order of frequent itemsets produced by *Eclat*. 52
3.9 Execution of *Eclat-Z* on dataset D with $min\_supp = 2$ (40%). 56
3.10 Response times of *Eclat-Z*. 57
3.11 Execution of *Charm-MFI* on dataset D with $min\_supp = 3$ (60%). 59
3.12 Response times of *Charm-MFI*. 60

4.1 Different sets of association rules extracted from dataset D (Table 3.1) by $min\_supp = 3$ (60%) and $min\_conf = 0.5$ (50%). 64
4.2 Characteristics of the different quality measures. 75
4.3 Generating $\mathcal{CR}$ without (−) and with (+) cache. 78
4.4 Generating $\mathcal{MNR}$ without (−) and with (+) cache. 78
4.5 Comparing sizes of different sets of association rules. 79
4.6 Execution times of rule generation. 79

5.1 Execution of the *Apriori-Rare* algorithm with $min\_supp = 3$ ($max\_supp = 2$). . 87
5.2 Minimal rare itemsets found in dataset D. 87
5.3 Execution of the *Arima* algorithm with $min\_supp = 3$ ($max\_supp = 2$). 89
5.4 Border sizes. 92
5.5 Rare equivalence classes found by *BtB* at $min\_supp = 4$. 98
5.6 Exact MRG rules found in dataset D (Table 3.1) by $min\_supp = 4$. 99
5.7 Approximate MRG rules found in dataset D (Table 3.1) by $min\_supp = 4$. 99
5.8 FR rules in dataset D (Table 3.1) by $min\_supp = 4$. 101

5.9 Distribution of MRG rules. 102
5.10 Steps taken to find MRG association rules. 103

6.1 Extracted association rules that are interesting from the expert's point of view. LDL-C: concentration in low-density cholesterol; BMI: body mass index; OC: oral contraceptive intake; MS: metabolic syndrome. 115

8.1 Un exemple de matrice binaire représentant des transactions entre des clients (C) et des produits (P). 130
8.2 Règles d'association extraites présentant un intérêt du point de vue de l'expert. . 141
8.3 Une base de données simple (D) utilisée dans les exemples. 145
8.4 Exécution de l'algorithme *Apriori-Rare* avec $min\_supp = 3$ ($max\_supp = 2$). . . 149
8.5 Exécution de l'algorithme *Arima* avec $min\_supp = 3$ ($max\_supp = 2$). 150

A.1 Database characteristics. 159

B.1 Tables used in *Apriori*. 161
B.2 Fields of the tables of *Apriori*. 161
B.3 Execution of *Apriori* on dataset D with $min\_supp = 3$ (60%). 163
B.4 Execution of *Apriori-Close* on dataset D with $min\_supp = 3$ (60%). 165
B.5 Response times of *Apriori* and *Apriori-Close*. 165
B.6 Response times of *Eclat*. The column "skipped intersections" indicates the number of intersection operations that can be saved in the case of 2-itemsets using the upper-triangular matrix optimization (Appendix E). The last two columns show the number of rare candidates generated by *Eclat* with (+) and without (−) element reordering, respectively. 170
B.7 Another dataset (D') for *Charm*. 172
B.8 Response times of *Charm*. 176

F.1 Comparison of the generators representation and the FCI representation of frequent itemsets. 191
F.2 Comparison of the generators representation and the FCI representation of frequent itemsets. 192

G.1 A toy dataset for the examples. 194
G.2 Our example dataset (Table G.1) in different file formats. 194

Chapter 1

Introduction

In this chapter we give an introduction of the thesis. In Sections 1.1 – 1.3 we give the context of the thesis, i.e. knowledge discovery in databases and data mining. In Section 1.4 we provide an overview of the thesis and we give a summary of the forthcoming chapters.

1.1 Knowledge Discovery in Databases

The *knowledge discovery in databases* process is aimed at extracting from large databases information units that can be interpreted as knowledge units to be reused. This process is based on three major steps: the selection and preparation of data, the data mining operation, and finally the interpretation of the extracted units.

Knowledge discovery in databases can be likened to the process of searching for gold in the rivers: the gold nuggets that are researched are knowledge units, and the rivers are the databases under study.[1] Huge volumes of data –and particularly documents– are available, without any intended usage. A fundamental question is to know if there may be something interesting in these data, and to find methods for extracting these "interesting things". The *knowledge discovery in databases* process –hereafter KDD– consists in processing a huge volume of data in order to extract knowledge units that are non trivial, potentially useful, significant, and reusable. Generally, the KDD process is iterative and interactive, and controlled by an expert of the data domain, called the *analyst*, who is in charge of guiding the extraction process, on the base of his objectives, and of his domain knowledge. The analyst selects and interprets a subset of the units for building "models" that will be further considered as knowledge units with a certain plausibility. The KDD process is based on three major steps: **(i)** the data sources are prepared to be processed, **(ii)** then they are mined, and **(iii)** finally, the extracted information units are interpreted for becoming knowledge units. These units are in turn embedded within a representation formalism to be used within a knowledge-based system. The KDD process may also be understood as a process turning data into information and then knowledge (see Figure 1.1), considering the following definitions [SAA+99, Wil02]:

Data. Data are the uninterpreted *signals* that reach our senses every minute. A red, green, or yellow light at an intersection is one example. Computers are full of data: signals consisting of strings of numbers, characters, and other symbols that are blindly and mechanically handled in large quantities.

[1]Actually, the name of our research team "Orpailleur" means "gold miner" in English.

```
Data (rough data, databases)
      ↓       Domain understanding
      ↓       Data selection (windowing)
Selected data
      ↓       Cleaning / Transformation of data
      ↓       Preparation of the data set
Prepared data
      ↓       Data mining process (discovering patterns)
      ↓       Numerical and symbolic KDD methods
Discovered patterns
      ↓       Post-processing of discovered patterns
      ↓       Interpretation / Evaluation
Knowledge units (for knowledge systems)
```

Figure 1.1: The KDD loop: from rough data to knowledge units. The overall objective of the KDD process is to select, prepare and extract knowledge units from different sources, and then to represent the extracted knowledge units in adequate knowledge structures.

Information. Information is data equipped with *meaning*. For a car driver, a red traffic light is not just a signal of some colored object, rather, it is interpreted as an indication to stop. In contrast, a color-blind person will probably not attach the same meaning to a red light.

Knowledge. Knowledge is the whole body of data and information that people bring to bear to practical *use in action*, in order to carry out tasks and create new information. Knowledge adds two distinct aspects: first, a sense of *purpose*, since knowledge is the "intellectual machinery" used to achieve a goal; second, a *generative capability*, because one of the major functions of knowledge is to produce new information. It is not accidental, therefore, that knowledge is proclaimed to be a new "factor of production".

The KDD process is performed within a KDD system that is composed of the following elements: the databases, the either symbolic or numerical data mining modules, and the interfaces for interactions with the system, e.g. editing and visualization. Moreover, the KDD system may take advantage of domain knowledge embedded within an *ontology* relative to the data domain. Closing the loop, the knowledge units extracted by the KDD system must be represented in an adequate representation formalism and then they may be integrated within the ontology to be reused for problem-solving needs in application domains such as agronomy, biology, chemistry, medicine, etc.

There are a number of general books that can be used with profit for understanding the KDD principles and the usage of the KDD methods, such as [FPSSU96, MBK98], and more recent textbooks such as [HK01, HMS01, Dun03], and [WF00] that is associated with the WEKA[2] system.

[2]`http://www.cs.waikato.ac.nz/~ml/weka/`

1.2 Data Mining – The Central Step of KDD

Data mining is the central step of the KDD process. The KDD process involves several stages: selecting the target data, pre-processing the data, transforming them if necessary, performing data mining to extract patterns and relationships, and then interpreting and assessing the discovered structures.

Progress in digital data acquisition and storage technology has resulted in the growth of huge databases. Consequently, interest has grown in the possibility of extracting information from these data that might be of value to the owner of the database. The discipline concerned with this task has become known as *data mining.* Hand *et al.* in [HMS01] define data mining the following way: "Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner."

Data mining typically deals with data that have already been collected for some purpose other than the data mining analysis. This means that the objectives of the data mining exercise play no role in the data collection strategy. For this reason, data mining is often referred to as "secondary" data analysis.

The definition also mentions that the datasets examined in data mining are often large. When we are faced with large bodies of data, new problems arise. Some of these relate to such fundamental issues as what data management strategy to use, how to analyze the data in a reasonable period of time, or how to filter noise from the data. Often the available data comprise only a sample from the complete population; the aim may be to *generalize* from the sample to the population. For instance, we might wish to predict how future customers are likely to behave. Sometimes we may want to summarize or *compress* a very large dataset in such a way that the result is more comprehensible, without any notion of generalization. This issue would arise, for example, if we had complete census data for a particular country or a database recording millions of individual retail transactions.

Data mining should not be seen as a simple one-time exercise. Huge data collections may be analyzed and examined in an unlimited number of ways. As time progresses, so new kinds of structures and patterns may attract interest, and may be worth seeking in the data.

Data mining has, for good reason, recently attracted a lot of attention: it is a new technology, tackling new problems, with great potential for valuable commercial and scientific discoveries [HMS01].

1.3 Methods for KDD

1.3.1 An Introducting Example

First, let us examine what may be expected from the application of data mining methods to data. Let us consider a binary matrix $\mathtt{M_{ij}}$, also called a *formal context*, where the rows represent *customers*, and the columns represent *products* bought by particular customers (see Table 1.1): $\mathtt{M_{ij}} = \mathtt{1}$ whenever the customer $\mathtt{i}$ buys the product $\mathtt{j}$. In real-world formal contexts, such a binary matrix may have thousands of columns, and millions of lines... From this formal context, one may extract the following units:

Customers/Products	chips	mustard	sausage	soft drink	beer
C_1	1	0	0	0	1
C_2	1	1	1	1	1
C_3	1	0	1	0	0
C_4	0	0	1	0	1
C_5	0	1	1	1	1
C_6	1	1	1	0	1
C_7	1	0	1	1	1
C_8	1	1	1	0	0
C_9	1	0	0	1	0
C_{10}	0	1	1	0	1

Table 1.1: An example of a binary matrix representing transactions between customers (C) and products (P).

- The set $\mathtt{X} = \{\mathtt{beer}, \mathtt{sausage}, \mathtt{mustard}\}$ has frequency $\phi(\mathtt{X}) = 4$, i.e. there are four individuals of ten buying the three products together. In the same way, the set $\mathtt{Y} = \{\mathtt{beer}, \mathtt{sausage}\}$ has frequency $\phi(\mathtt{Y}) = 6$. The set X (respectively Y) may be interpreted as the fact that 40% (resp. 60%) of the customers buy the products in X (resp. in Y) at the same time.

- Moreover, the rule $\mathtt{R} = \{\mathtt{beer}, \mathtt{sausage} \rightarrow \mathtt{mustard}\}$ may be extracted from the sets X and Y (i.e. $\mathtt{Y} \rightarrow \mathtt{X}\setminus\mathtt{Y}$ where $\mathtt{X}\setminus\mathtt{Y}$ denotes the set X without Y), with the confidence 0.66 (66.6%), i.e. if a customer buys **beer** and **sausage**, then the probability that he buys **mustard** is 0.66 (among six customers buying beer and sausage, four customers also buy mustard).

  From the point of view of the analyst, the sets X and Y, and the rule R as well, may be interpreted and validated as knowledge units extracted from the data.

1.3.2 Data Mining Methods

The extraction process is based on *data mining methods* returning knowledge units from the considered data. The data mining methods can be either symbolic or numerical:

- Symbolic methods include among others: classification based on decision trees, lattice-based classification, frequent itemsets search and association rule extraction, classification based on rough sets [Paw91], learning methods, e.g. induction, instance-based learning, explanation-based learning [Mit97, MBK98], and database methods based on information retrieval and query answering...

- Numerical methods include among others: statistics and data analysis, hidden Markov models of order 1 and 2 (initially designed for pattern recognition), Bayesian networks, neural networks, genetic algorithms...

These methods are dependent on research domains to which the KDD process is linked [Man97]:

- *Statistics and data analysis*: the goal is similar, but the KDD process requires most of the time a combination of different methods, symbolic as well as numerical methods, and domain knowledge for the interpretation of the extracted units.

- *Database management*: database management system techniques may be used to help solving the data mining task, e.g. using the query capabilities for preparing data to be mined.
- *Machine learning*: machine learning methods are the core of the KDD process, but scalability, i.e. the amount of data that is considered, and the objectives are different, i.e. reusing the results of the KDD process for problem-solving or decision making.
- *Knowledge representation and reasoning*: the data mining process may be guided by a model –a domain ontology– for interpretation and problem-solving.

The KDD process may be considered as a kind of "supervised learning process" – since an analyst is in charge of controlling and guiding the KDD process. The analyst may take advantage of his own knowledge and of domain ontologies, for giving an interpretation of the results and for validating the results. In this way, the results of the KDD process may be reused for enlarging existing ontologies, showing that knowledge representation and KDD are two complementary processes: *no data mining without knowledge on the data domain!*

1.4 Overview of the Thesis

The main topic of the thesis is *knowledge discovery in databases* (KDD). More precisely, we have investigated one of the most important tasks of data mining today, namely itemset extraction and association rule generation. Throughout our work we have borne in mind that our goal is to find *interesting* association rules. We have developed specific algorithms in order to achieve this goal.

The main contributions of this thesis are: **(1)** We have developed algorithms for finding minimal non-redundant association rules; **(2)** We have defined a new basis for association rules called Closed Rules; **(3)** We have investigated an important but relatively unexplored field of KDD namely the extraction of rare itemsets and rare association rules; **(4)** We have packaged our algorithms along with other auxiliary operations for KDD into a unified software toolkit called CORON.

We now present a more detailed summary of the thesis contributions.

1.4.1 Frequent Itemset Search

In Chapter 3 we present two algorithms that we have specifically adapted to extract minimal non-redundant association rules ($\mathcal{MNR}$). These rules are lossless, sound and informative representations of all valid association rules. The output of most of the other algorithms cannot be used directly for generating $\mathcal{MNR}$ rules. The first algorithm, *Zart*, is a practical extension of *Pascal*, which is one of the most efficient levelwise algorithms for finding frequent itemsets (FIs). In addition to *Pascal*'s capabilities, *Zart* identifies the set of frequent closed itemsets (FCIs) and associates their generators to them. We show that this extra output from *Zart* is essential for extracting $\mathcal{MNR}$ rules. In the second algorithm, *Eclat-Z*, we go further and we show how to generalize the idea in *Zart* for *any* frequent itemset mining algorithm. This way, arbitrary FI-miner algorithms can be extended in order to support the extraction of $\mathcal{MNR}$ rules.

In Chapter 3 we propose a simple modification of *Charm* that we call *Charm-MFI* which identifies maximal frequent itemsets among frequent closed itemsets. This algorithm has been used for our border studies.

1.4.2 Frequent Association Rules

In Chapter 4 we present different sets of frequent association rules, namely all valid rules and the family of minimal non-redundant rules. We also introduce a new basis called Closed Rules that we position between the previously mentioned two sets of association rules, filling a gap between them. Closed Rules is a concise representation of all valid rules, and it only requires frequent closed itemsets. Closed Rules seems to be a good alternative to all valid association rules.

In the literature, most of the algorithms concentrate only on the support and the confidence values of rules. We show how to calculate some other statistical measures that require, in addition, the support value of the right sides of the rules. In our work, for the condensed representation of FIs we use FCIs that are stored in a trie data structure. Deriving the support values of both sides of rules would require *lots of* trie operations. To overcome this problem, we propose a hash-based itemset cache that proved to be a very efficient solution. The use of cache is advantageous even if we do not need other interestingness measures, and thus the caching technique can also be incorporated into the other algorithms.

1.4.3 Rare Itemsets and Rare Association Rules

Chapter 5 is one of the most original part of this thesis work. In this chapter, we address the problems of extracting rare itemsets and generating rare association rules. In the literature, these problems have not yet been studied in detail, although rare itemsets can also contain important information just as frequent itemsets do. A particularly relevant field for rare itemsets is medical diagnosis.

In Chapter 5.1 we present a method for finding all *rare itemsets*. For this task we use the well-known *Apriori* algorithm. *Apriori* is known to find all FIs, but actually it also finds a special subset of rare itemsets, the minimal rare itemsets (MRIs). A slight modification of *Apriori*, which we call *Apriori-Rare*, retains MRIs instead of dropping them. We show how to restore all rare itemsets from MRIs while avoiding itemsets with support 0.

In Chapter 5.3 we go further by showing how to generate valid *rare association rules*. Our work is motivated by the long-standing open question of devising an efficient algorithm for finding rules with low support and very high confidence. In order to find such rules using conventional frequent itemset mining algorithms like *Apriori*, the minimum support must be set very low, which drastically increases the runtime of the algorithm. Moreover, when minimum support is set very low, *Apriori* produces a huge number of frequent itemsets. This is also known as the *rare item problem*. For this long-existing problem we propose a solution. With our method we can extract a set of exact rare association rules (we call these rules "exact MRG rules"). We also show how to extract approximate MRG rules; however their interestingness is doubtful. Thus, we concentrate more on exact rare rules. Furthermore, these rules are non-redundant because the antecedent is minimal and the consequent is maximal, implying that among rules with the same support and same confidence, these rules contain the most information.

1.4.4 The Coron Toolkit

All the algorithms that we present in this thesis are implemented and grouped together in a unified software toolkit called CORON. CORON is a domain and platform independent, multi-purposed data mining platform, which incorporates not only a rich collection of data mining algorithms, but also allows a number of auxiliary operations. To the best of our knowledge, a data mining toolkit designed specifically for itemset extraction and association rule generation like CORON does not exist elsewhere. CORON also provides support for preparing and filtering data, and for interpreting the extracted units of knowledge.

Most of the experiments with CORON were performed on a real-life biomedical dataset called STANISLAS cohort. During these experiments, we realized that we needed a **(1)** methodology for mining, and **(2)** a tool for implementing the methodology. Chapter 6 presents our global data mining methodology that can be generalized to arbitrary datasets. The methodology can be used for both frequent and rare association rules.

At the end of Chapter 6, besides the STANISLAS cohort, we present three other projects that use the CORON toolkit with success.

1.5 Thesis Outline

In **Chapter 2** we begin by presenting the state of the art. Our main focus is symbolic KDD methods based on the classification operation, more precisely on lattice-based classification, frequent itemset search, and association rule extraction. We show how the whole transformation process from rough data to knowledge units is based on the underlying principle of classification.

In **Chapter 3** we present in depth concepts and algorithms of frequent itemset search. We propose algorithms that we have specifically adapted to our needs, i.e. for finding minimal non-redundant association rules and for studying the border. In Appendix B we present some classical algorithms that are very closely related to this chapter.

In **Chapter 4** we investigate the extraction of different sets of frequent association rules, namely all valid rules, Closed Rules, and the family of minimal non-redundant association rules.

In **Chapter 5** we address the problems of the extraction of rare itemsets and the generation of rare association rules. This is one of the most important part of this thesis.

In **Chapter 6** we present a global data mining methodology for mining both frequent and rare association rules. The second part of the chapter describes the CORON platform that implements the methodology in its entirety.

Finally, **Chapter 7** summarizes the main contribution of the thesis, and suggests avenues for future work.

It must be noted that the appendices are also an important part of the thesis. Among other things, we present some classical algorithms, an efficient data structure for itemsets, different optimization techniques, and a detailed user guide for the CORON toolkit.

Chapter 2

Symbolic Methods for Knowledge Discovery

This chapter provides an overview of the so-called *symbolic methods* in knowledge discovery, namely lattice-based classification, frequent itemset search and association rule extraction. Then, we detail some applications of the KDD process, and we propose a discussion of the main characteristics of the KDD process and a conclusion for ending the chapter.

2.1 Lattice-Based Classification

A number of classification problems can be formalized by means of a class of individuals (or objects) and a class of properties (or attributes), and a binary correspondence between the two classes, indicating for each individual-property pair whether the property applies to the individual or not [BM70, GVM93, GW99]. The properties may be features that are present or absent, or the values of a property that have been dichotomized into binary variables. These variables are collected into binary tables relating a set of individuals with a set of properties, where $(\mathtt{i}, \mathtt{j}) = 1$ or is *true* whenever the individual $\mathtt{i}$ has the property $\mathtt{j}$ (just as illustrated in Table 1.1).

Lattice-based classification relies on the analysis of such binary tables and may be considered as a symbolic data mining technique that can be used for extracting from a database a set of concepts organized within a hierarchy (i.e. a partial ordering), frequent itemsets, i.e. sets of properties or features of data occurring together with a certain frequency, and association rules with a given confidence emphasizing correlations between sets of properties.

More precisely, a *lattice* is an ordered set $(\mathtt{E}, \sqsubseteq)$, where $\sqsubseteq$ denotes a partial ordering such that every pair of elements $(\mathtt{x}, \mathtt{y})$ has an *upper bound* $\mathtt{x} \vee \mathtt{y}$ and a *lower bound* $\mathtt{x} \wedge \mathtt{y}$ [DP90]. The partial order relation $\sqsubseteq$ is also called a "subsumption relation". The powerset $2^{\mathtt{E}}$ of a set $\mathtt{E}$ equipped with the inclusion relation is a basic example of a lattice (see Figure 2.1). Note that the powerset of $\mathtt{E}$ is often denoted as $\mathfrak{P}(\mathtt{E})$. The set of natural numbers $\mathbb{N}$ equipped with the divisibility relation is also a lattice: $\mathtt{x} \sqsubseteq \mathtt{y}$ if and only if $\mathtt{y}$ is a divisor of $\mathtt{x}$ in $\mathbb{N}$ (see Figure 2.2).

A lattice may be built according to the so-called *Galois connection*, classifying within a formal concept a set of individuals, i.e. the *extension* of the concept, sharing a same set of properties, i.e. the *intension* of the concept. Considering the Boolean correspondence between individuals and properties (as shown in Table 1.1), it is possible to derive for each individual $\mathtt{i}$ the set of all properties that apply to $\mathtt{i}$. Similarly, it is possible to derive for each property $\mathtt{j}$ the set of all individuals to which $\mathtt{j}$ applies. One may further derive *rectangles*, i.e. pairs $\mathtt{O} \times \mathtt{A}$ where $\mathtt{O}$ is a set of individuals and $\mathtt{A}$ is a set of properties, such that every property of $\mathtt{A}$ applies to every

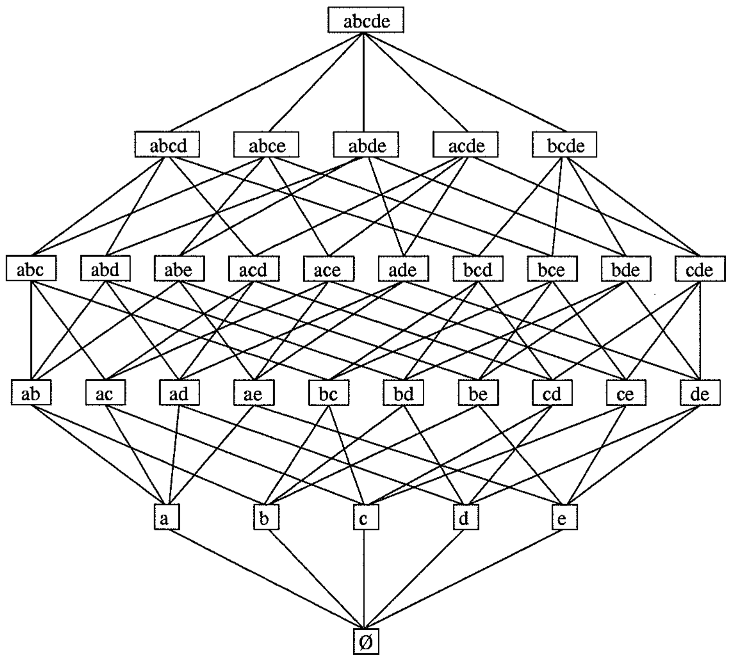


Figure 2.1: The lattice representing the power set of the set `{a,b,c,d,e}`.

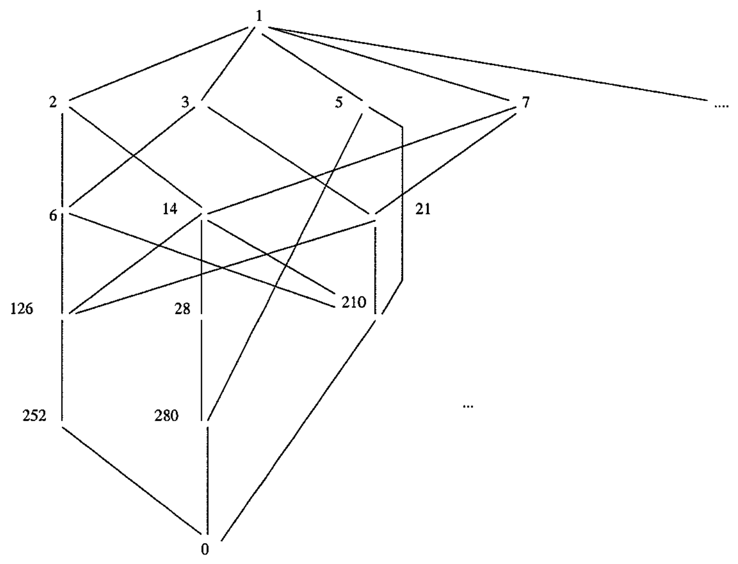


Figure 2.2: The lattice representing a part of the divisibility relation in $\mathbb{N}$.

Objects / Items	a	b	c	d	e
O_1	0	1	1	0	1
O_2	1	0	1	1	0
O_3	1	1	1	1	0
O_4	1	0	0	1	0
O_5	1	1	1	1	0
O_6	1	0	1	1	0

Table 2.1: An example of formal context.

individual of O. Moreover, *maximal rectangles* $O \times A$ are such that the property set A consists of *all* common properties of the individuals in O, and that the individual set O consists of *all* individuals to which the properties of A jointly apply. Maximal rectangles are called *formal concepts*: they are *concepts* because they actually represent a class of objects, where the individual set O is the *extension* of the class, and the property set A is the *intension* of the class; they are *formal concepts* because they are mathematical entities that do not necessarily refer to any reality.

From a mathematical point of view, let E and F be two finite sets, and R a binary relation on $E \times F$.

Definition 2.1 *The mapping* $f : E \rightarrow F$ *is such that, if* x *is an element of* E, $f(\{x\})$ *consists of all elements of* F *related to* x *by* R, *i.e.* $f(\{x\}) = \{y \in F \mid xRy\}$. *If* X *is an arbitrary subset of* E, $f(X) = \{y \in F \mid \forall x \in X : xRy\}$.

Dually, the mapping $g : F \rightarrow E$ *is such that, if* y *is an element of* F, $g(\{y\})$ *consists of all elements of* E *that are related to* y *by* R, *i.e.* $g(\{y\}) = \{x \in E \mid xRy\}$. *If* Y *is an arbitrary subset of* F, $g(Y) = \{x \in E \mid \forall y \in Y : xRy\}$.

The couple $\{f, g\}$ *is said to be a* Galois connection *(or a* Galois correspondence*) between the sets* E *and* F.

Two mappings f and g form a *Galois connection* if:
(1) $\forall X_1, X_2 \in \mathfrak{P}(E), X_1 \subseteq X_2 \Rightarrow f(X_2) \subseteq f(X_1)$
(2) $\forall Y_1, Y_2 \in \mathfrak{P}(F), Y_1 \subseteq Y_2 \Rightarrow g(Y_2) \subseteq g(Y_1)$
(3) $\forall X \in \mathfrak{P}(E), X \subseteq f \circ g\ (X)$ and $\forall Y \in \mathfrak{P}(F), Y \subseteq g \circ f\ (Y)$.

In terms of objects and attributes, $f(X)$ is the set of all attributes shared by all objects in X, and $g(Y)$ is the set of all objects that have all attributes of Y. Moreover, $X_1 \subseteq X_2 \Rightarrow f(X_2) \subseteq f(X_1)$, and $Y_1 \subseteq Y_2 \Rightarrow g(Y_2) \subseteq g(Y_1)$: the mappings f and g are decreasing. For example, considering the binary table of Table 2.1, we have $f(\{O_1\}) = \{b, c, e\}$ and $g(\{b, c, e\}) = \{O_1\}$, $f(\{O_1, O_2\}) = \{c\}$ and $g(\{c\}) = \{O_1, O_2, O_3, O_5, O_6\}$, $g(\{a, c\}) = \{O_2, O_3, O_5, O_6\}$ and $f(\{O_2, O_3, O_5, O_6\}) = \{a, c, d\}$.

The mapping $h_1 = g \circ f = g[f]$ maps every part of E onto a part of E, and the mapping $h_2 = f \circ g = f[g]$ maps every part of F onto a part of F. It can be shown that the mappings h_1 and h_2 are *closure operators*:

Definition 2.2 *Let* X, X_1 *and* X_2 *be subsets of* E. *A* closure operator h *is: (i) monotonously increasing, i.e.* $X_1 \subseteq X_2 \Rightarrow h(X_1) \subseteq h(X_2)$, *(ii) extensive, i.e.* $X \subseteq h(X)$, *and (iii) idempotent, i.e.* $h(X) = h[h(X)]$.

A subset X *of* E *is said to be* closed *if and only if* $X = h(X)$.

The closure operators $h_1 = g \circ f = g[f]$ for E and $h_2 = f \circ g = f[g]$ for F are said to be *Galois closures*. Let L_E and L_F be the sets of all closed parts of E and F respectively, partially ordered

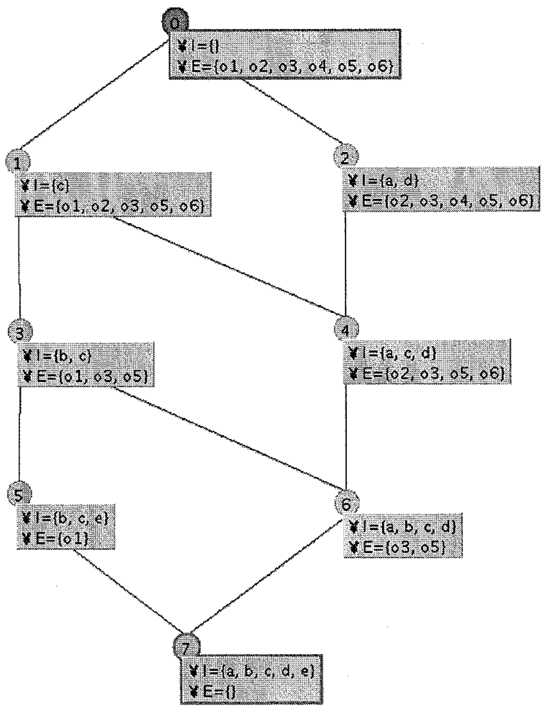


Figure 2.3: The Galois lattice associated to the formal context of Figure 2.1.

by set inclusion. Then, $(L_E, \subseteq)$ and $(L_F, \subseteq)$ have lattice structures: the meet of two parts is their intersection, whereas the join of two parts is the closure of their union[3]. The Galois connection $\{f, g\}$ restricted to the closed parts of E and F materializes a one-to-one correspondence between the lattices $(L_E, \subseteq)$ and $(L_F, \subseteq)$.

We may now consider the set L of all couples of corresponding parts of L_E and L_F, i.e. each element of L is the Cartesian product of closed parts of E and F, denoted by $(X, f(X))$, or $(g(Y), Y)$, with X, f(X), Y, and g(Y) being closed. The partial order relation $\sqsubseteq$ may be defined on L such that $(X_1, Y_1) \sqsubseteq (X_2, Y_2)$ if and only if $X_1 \subseteq X_2$ (or dually $Y_2 \subseteq Y_1$). The structure $(L, \sqsubseteq)$ is the *Galois lattice* or the *concept lattice* of the relation R on $E \times F$, and it can be demonstrated that the elements of L are the formal concepts derived from the relation R. For example, the Galois lattice associated to the formal context of Table 2.1 is shown in Figure 2.3.

More precisely, the partial order between two concepts $(X_1, Y_1) \sqsubseteq (X_2, Y_2)$ verifies that the extension X_2 of (X_2, Y_2), i.e. the *subsumer* concept, includes the extension X_1 of (X_1, Y_1), i.e. the *subsumed* concept, and, dually, that the intension Y_2 of (X_2, Y_2) is included in the intension Y_1 of (X_1, Y_1). Thus, there exists an order-reversing one-to-one correspondence between the extensions and the intensions of formal concepts, *covariant* for the extensions and *contravariant* for the intensions. Moreover, there exists a number of algorithms for building Galois lattices –see [Gue90, Duq99, GW99, KO01, KO02]– with different and specific characteristics.

Lattice-based classification may be used for a number of purposes in KDD [SWW98, Wil02, VMG04]:

[3]The union of two closed sets is not necessarily a closed set as it is the case for the intersection of two closed sets.

- Since the concepts are the basic units of human thought (and hence the basic structures of logic), the logical structure of information is based on concepts and concept systems. Therefore, Galois (or concept) lattices, as mathematical abstraction of concept systems, can support humans to discover information and then to create knowledge.

- It is important to have a mathematization of concepts that reflects the rich logical functionalities in which concepts are embedded in the real-world. Concept lattices and lattice-based classification are examples of such mathematical tools. Indeed, the mathematical structure of a concept lattice is effectively accessible to human reasoning by labeled line diagrams (lattice drawings).

- Lattice-based classification and formal concept analysis is a suitable paradigm for KDD, as discussed in [VMG04]. The mathematical and algorithmic backgrounds exist and may be used for real-sized applications [Kuz04, KO02]. Moreover, some improvements may be carried on, especially on facility, i.e. the ease of use of the data mining methods, on the cost-effectiveness of the methods allowing effective and efficient implementations, e.g. distributive and parallel architectures, and finally on adaptability, i.e. the ability to fit evolving situations with respect to the constraints that may be associated to the KDD process. Moreover, one other major research point is the extension of lattice-based classification to complex objects, where properties may be many-valued properties, or even relations.

- Lattice-based classification has also been used for Information Retrieval (IR) and document clustering [CR04, GMM95]. The application of FCA to the field of IR is motivated by the obvious analogy exisiting between object/attribute (in FCA) and document/term (in IR) tables. Concepts in a lattice are seen as classes of relevant documents that match a given user query. The subsumption relation allows moving from one query to another (to a more general or to a more specific query).

2.1.1 Classical Notation of Formal Concept Analysis

The notion of Galois lattice has given rise to the lattice-based classification, and to the active research domain of *formal concept analysis*[4] [GW99]. Formal concept analysis is used for a number of different tasks, among which the design of object hierarchies, especially in object-oriented programming for designing class hierarchies.

Here we recall the classical notation of Formal Concept Analysis for future reference, as presented in the book of Ganter and Wille [GW99].

The basic notions of Formal Concept Analysis are those of *formal context* and a *formal concept.* The adjective "formal" is meant to emphasize that we are dealing with mathematical notions. From now on, where we write *context* or *concept* we actually mean a *formal context* or a *formal concept*, respectively.

Definition 2.3 *Let $(M, \leq)$ be an ordered set and A a subset of M. A* ***lower bound*** *of A is an element s of M with $s \leq a$ for all $a \in A$. An* ***upper bound*** *of A is defined dually. If there is a largest element in the set of all lower bounds of A, it is called the* ***infimum*** *of A and is denoted by $\inf A$ or $\bigwedge A$; dually, a least upper bound is called* ***supremum*** *and denoted by $\sup A$ or $\bigvee A$.*

[4]fca-list@aifb.uni-karlsruhe.de
http://www.aifb.uni-karlsruhe.de/mailman/listinfo/fca-list

If $A = \{x, y\}$, we also write $x \wedge y$ for inf A and $x \vee y$ for sup A. Infimum and supremum are frequently also called ***meet*** *and* ***join****.*

Definition 2.4 *An ordered set $\mathbf{V} := (V, \leq)$ is a* ***lattice****, if for any two elements x and y in V the supremum $x \vee y$ and the infimum $x \wedge y$ always exist. $\mathbf{V}$ is called a* ***complete lattice****, if the supremum $\bigvee X$ and the infimum $\bigwedge X$ exist for any subset X of V. Every complete lattice $\mathbf{V}$ has a largest element, $\bigvee V$, called the* ***unit element*** *of the lattice, denoted by 1_V. Dually, the smallest element 0_V is called the* ***zero element****.*

Definition 2.5 *A* ***formal context*** *$\mathbb{K} := (G, M, I)$ consists of two sets G and M and a relation I between G and M. The elements of G are called the* ***objects*** *and the elements of M are called the* ***attributes*** *of the context.*[5] *In order to express that an object g is in relation I with an attribute m, we write gIm or $(g, m) \in I$ and read it as "the object g* has *the attribute m".*

A small context can be easily represented by a **cross table**, i.e. by a rectangular table the rows of which are headed by the object names and the columns headed by the attribute names. A cross in row g and column m means that the object g has the attribute m. See Table 3.1 for an example.

Definition 2.6 *For a set $A \subseteq G$ of objects we define*

$$A' := \{m \in M \mid gIm \text{ for all } g \in A\}$$

(the set of attributes common to the objects in A). Correspondingly, for a set B of attributes we define

$$B' := \{g \in G \mid gIm \text{ for all } m \in B\}$$

(the set of objects which have all attributes in B).

Note that these two mappings ($'$: $G \to M$ and $'$: $M \to G$) form a Galois connection.

Definition 2.7 *A* ***formal concept*** *of the context (G, M, I) is a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$. We call A the* ***extent*** *and B the* ***intent*** *of the concept (A, B). $\mathfrak{B}(G, M, I)$ denotes the set of all concepts of the context (G, M, I).*

Proposition 2.1 *If (G, M, I) is a context, A, A_1, $A_2 \subseteq G$ are sets of objects and B, B_1, $B_2 \subseteq M$ are sets of attributes, then*

(1) $A_1 \subseteq A_2 \Rightarrow A_2' \subseteq A_1'$ *(1') $B_1 \subseteq B_2 \Rightarrow B_2' \subseteq B_1'$*
(2) $A \subseteq A''$ *(2') $B \subseteq B''$*
(3) $A' \subseteq A'''$ *(3') $B' \subseteq B'''$*

(4) $A \subseteq B' \iff B \subseteq A' \iff A \times B \subseteq I$.

The proposition shows that the two derivation operators form a Galois connection between the powerset lattices $\mathfrak{P}(G)$ and $\mathfrak{P}(M)$. Hence we obtain two closure systems on G and M, which are dually isomorphic to each other:

For every set $A \subseteq G$, A' is an intent of some concept, since (A'', A') is always a concept. Consequently, a set $A \subseteq G$ is an extent if and only if $A = A''$. The same applies to intents. The union of extents generally does not result in an extent. On the other hand, the intersection of any number of extents (respectively intents) is always an extent (intent), as is proved by the following proposition:

[5]Strictly speaking: "formal objects" and "formal attributes".

Proposition 2.2 *If T is an index set and, for every $t \in T$, $A_t \subseteq G$ is a set of objects, then*

$$\left(\bigcup_{t\in T} A_t\right)' = \bigcap_{t\in T} A_t'.$$

The same holds for sets of attributes.

Definition 2.8 *If (A_1, B_1) and (A_2, B_2) are concepts of a context, (A_1, B_1) is called a* ***subconcept*** *of (A_2, B_2), provided that $A_1 \subseteq A_2$ (which is equivalent to $B_2 \subseteq B_1$). In this case, (A_2, B_2) is a* ***superconcept*** *of (A_1, B_1), and we write $(A_1, B_1) \leq (A_2, B_2)$. The relation $\leq$ is called the* ***hierarchical order*** *(or simply* ***order****) of the concepts. The set of all concepts of (G, M, I) ordered in this way is denoted by $\underline{\mathfrak{B}}(G, M, I)$ and is called the* ***concept lattice*** *of the context (G, M, I).*

Theorem 2.1 (The Basic Theorem of Concept Lattices) *The concept lattice $\underline{\mathfrak{B}}(G, M, I)$ is a complete lattice in which infimum and supremum are given by:*

$$\bigwedge_{t\in T}(A_t, B_t) = \left(\bigcap_{t\in T} A_t, \left(\bigcup_{t\in T} B_t\right)''\right)$$

$$\bigvee_{t\in T}(A_t, B_t) = \left(\left(\bigcap_{t\in T} A_t\right)'', \bigcap_{t\in T} B_t\right).$$

2.2 Frequent Itemset Search and Association Rule Extraction

In parallel with lattice-based classification, one may extract frequent itemsets and association rules from data (as shown in the introductory example in Section 1.3.1). Here we give an overview of these two methods, and we detail them in Sections 3 and 4, respectively. The extraction of frequent itemsets consists in extracting from formal binary contexts sets of properties occurring with a support, i.e. the number of individuals sharing the properties, greater than a given threshold. From the frequent itemsets, it is then possible to generate association rules of the form `A` $\longrightarrow$ `B` relating a subset of properties `A` with a subset of properties `B`, that can be interpreted as follows: the individuals including `A` include also `B` with a certain support and a certain confidence. The numbers of itemsets and rules that can be extracted from a formal binary context may be very large, and thus there is a need for pruning the sets of itemsets and the sets of extracted rules for ensuring a subsequent interpretation of the extracted units. This is especially true when the interpretation has to be done –and this is usually the case– by an analyst who is in charge of interpreting the results of the KDD process.

In the following, we introduce the principles of frequent itemset search and of the extraction of association rules. Then practical examples of both models are proposed.

2.2.1 Frequent Itemset Search

Definition 2.9 *Given a set of objects `O` and a set of properties `P`, an* item *corresponds to a property of an object, and an* itemset, *or a* pattern, *to a set of items: an object is said to* include *an item. The number of items in an itemset determines the* length *of the itemset. The* image *of an itemset corresponds to the set of objects including the item.*

The support *of an itemset corresponds to the number of objects including the itemset. An itemset is said to be* frequent *if its support is greater than or equal to a given* frequency threshold *minimum support (denoted by min_supp).*

For example, considering the formal context shown in Table 2.1, with *min_supp* = 3, we have: `{a}` is a frequent itemset of length 1 and of support 5; `{ac}` is of length 2, of support 4, and frequent; `{abc}` is of length 3, of support 2, and not frequent; `{abcde}` is of length 5, of support 0, and not frequent. It can be noticed that the support is a monotonously decreasing function, with respect to the length of an itemset.

When the number of properties in P is equal to n, the number of potential itemsets is equal to 2^n (actually, the number of all possible subsets of the set P): thus, a direct search for the frequent itemsets by directly testing the itemsets that are frequent is not conceivable. Heuristics have to be used for pruning the set of all itemsets to be tested. This is the purpose of the so-called *levelwise search* of frequent itemsets, and the associated well-known *Apriori* algorithm [AIS93, AS94, MTV94, AMS+96]. *Apriori* relies on two fundamental and dual principles: **(i)** *every sub-itemset of a frequent itemset is a frequent itemset,* **(ii)** *every super-itemset of an infrequent itemset is infrequent. Apriori* can be summarized as follows:

1. The search for frequent itemsets begins with the search for frequent itemsets of length 1.

2. The frequent itemsets are recorded and combined together to form *candidate* itemsets of greater length. The infrequent itemsets are discarded, and by consequence, all their super-itemsets. The candidate itemsets are then tested, and the process continues in the same way, until no more candidates can be formed.

For example, considering the formal context on Table 2.1, with *min_supp* = 2, the frequent itemsets of length 1 are `{a}` (3), `{b}` (5), `{c}` (5), `{d}` (5). The itemset `{e}` (1) is not frequent and pruned. Then the candidates of length 2 are formed, combining the frequent itemsets of length 1, e.g. `{ab}`, `{ac}`, `{ad}`, `{bc}`... and then tested. The frequent itemsets of length 2 are `{ab}` (2), `{ac}` (4), `{ad}` (5), `{bc}` (3), `{bd}` (2), `{cd}` (4). The candidates of length 3 are formed and tested: the frequent itemsets of length 3 are `{abc}` (2), `{abd}` (2), `{acd}` (4), `{bcd}` (2). Finally, the candidate of length 4 is formed, i.e. `{abcd}`, tested and recorded as a frequent itemset (`{abcd}` (2)). No other candidates can be formed, and the algorithm terminates.

When the data to be mined are huge, i.e. millions of rows and thousands of columns, there is a need for minimizing the access to the data for calculating the support. A number of studies have been carried out in this direction, giving rise to very efficient algorithms (see for example [PBTL99d, PBTL99b, ZH02]).

Lattices and itemsets are related: actually, the search for frequent itemsets corresponds to a breadth-first search in the concept lattice associated to the formal context under study. However, an itemset corresponds to a subset of properties, without being necessarily a closed set. In this way, the property of closure for an itemset is one of the characteristics which underlie fast algorithms for generating itemsets (it can be noticed that the name of one of these algorithms is *Close* [PBTL99b, PBTL99d]).

2.2.2 Association Rule Extraction

Definition 2.10 *An* association rule *has the form* $\mathtt{A} \longrightarrow \mathtt{B}$*, where* A *and* B *are two itemsets. The* support *of the rule* $\mathtt{A} \longrightarrow \mathtt{B}$ *is defined as the support of the itemset* $\mathtt{A} \cup \mathtt{B}$*. The* confidence *of a* *rule* $\mathtt{A} \longrightarrow \mathtt{B}$ *is defined as the quotient* $\mathtt{supp}(\mathtt{A} \cup \mathtt{B})/\mathtt{supp}(\mathtt{A})$. *The confidence can be seen as a conditional probability* $\mathtt{P}(\mathtt{B}|\mathtt{A})$, *i.e. probability of* $\mathtt{B}$ *knowing* $\mathtt{A}$.

A rule is said to be valid *(or* strong*) if its confidence is greater than or equal to a* confidence threshold *minimum confidence (denoted by min_conf), and its support is greater than or equal to the* frequency threshold *minimum support (denoted by min_supp). A valid rule can only be extracted from a frequent itemset. A rule is said to be* exact *if its confidence is equal to 1, i.e.* $\mathtt{supp}(\mathtt{A} \cup \mathtt{B}) = \mathtt{supp}(\mathtt{A})$, *otherwise the rule is* approximate.

For example, with $min\_supp = 3$ and $min\_conf = 3/5$, {ac} is frequent, and the rule $\mathtt{a} \longrightarrow \mathtt{c}$ is valid (with support 4 and confidence 4/5); the rule $\mathtt{c} \longrightarrow \mathtt{a}$ is valid (with support 4 and confidence 4/5). With $min\_supp = 2$ and $min\_conf = 3/5$, {abd} is frequent, the rule $\mathtt{b} \longrightarrow \mathtt{ad}$ is valid (with support 2 and confidence 2/3); the rule $\mathtt{ad} \longrightarrow \mathtt{b}$ is not valid (with support 2 and confidence 2/5).

The generation of valid association rules from a frequent itemset (of length necessarily greater than or equal to 2) proceeds in a similar way as the search for frequent itemsets. Given a frequent itemset P, the extraction starts by generating the valid rules with a right hand side (conclusion) of length 1, say rules of the form $\mathtt{P} \setminus \{\mathtt{i}\} \longrightarrow \{\mathtt{i}\}$, where $\{\mathtt{i}\}$ is an item of length 1, and $\mathtt{P} \setminus \{\mathtt{i}\}$ denotes the itemset P without the item $\{\mathtt{i}\}$. Then, the conclusions of the valid rules $\mathtt{P}\setminus\{\mathtt{i}\} \longrightarrow \{\mathtt{i}\}$ are combined for generating the candidate conclusions of length 2, e.g. $\mathtt{P} \setminus \{\mathtt{ij}\} \longrightarrow \{\mathtt{ij}\}$, and the process continues until no more valid rules can be generated from the frequent itemset.

For example, with our current formal context, given $min\_supp = 2$ and $min\_conf = 2/5$, when P = {ab}, the generated valid rules are $\{\mathtt{a}\} \longrightarrow \{\mathtt{b}\}$ (supp: 2; conf: 2/5) and $\{\mathtt{b}\} \longrightarrow \{\mathtt{a}\}$ (2; 2/3). Given the frequent itemset P = {abc} (2), the generated rules are $\{\mathtt{ab}\} \longrightarrow \{\mathtt{c}\}$ (2; 1), $\{\mathtt{ac}\} \longrightarrow \{\mathtt{b}\}$ (2; 1/2), $\{\mathtt{bc}\} \longrightarrow \{\mathtt{a}\}$ (2; 2/3); as {a,b,c} has three valid conclusions, they can be combined for producing the new conclusions {ab,ac,bc}, and generate the rules $\{\mathtt{c}\} \longrightarrow \{\mathtt{ab}\}$ (2; 2/5), $\{\mathtt{b}\} \longrightarrow \{\mathtt{ac}\}$ (2; 2/3), $\{\mathtt{a}\} \longrightarrow \{\mathtt{bc}\}$ (2; 2/5), which are all valid rules.

There exists a number of studies on the possible measures that can be attached to an association rule [LFZ99, TKS02, CNT03]. Considering the confidence of the rule $\mathtt{A} \longrightarrow \mathtt{B}$ as the conditional probability $\mathtt{P}(\mathtt{B}|\mathtt{A})$ (probability of B knowing A), other measures may be built on the basis of probability calculus:

- The *interest* or *lift* of the rule $\mathtt{A} \longrightarrow \mathtt{B}$ measure is defined as $\mathtt{P}(\mathtt{A} \cup \mathtt{B})/\mathtt{P}(\mathtt{A}) \times \mathtt{P}(\mathtt{B})$, i.e. the interest measures the degree of compatibility of A and B, i.e. the simultaneous occurrences of both events A and B.
- The *conviction* of the rule $\mathtt{A} \longrightarrow \mathtt{B}$ is defined as $\mathtt{P}(\mathtt{A}) \times \mathtt{P}(\neg\mathtt{B})/\mathtt{P}(\mathtt{A} \cup \neg\mathtt{B})$, i.e. the conviction measures the deviation of the rule $\mathtt{A} \longrightarrow \mathtt{B}$ from the rule $\mathtt{A} \longrightarrow \neg\mathtt{B}$, or, in other words, how high is the degree of implication of the rule $\mathtt{A} \longrightarrow \neg\mathtt{B}$.
- The *dependency* of the rule $\mathtt{A} \longrightarrow \mathtt{B}$ is defined as $|\mathtt{P}(\mathtt{B}|\mathtt{A}) - \mathtt{P}(\mathtt{B})| = |\mathtt{P}(\mathtt{A} \cup \mathtt{B})/\mathtt{P}(\mathtt{A}) - \mathtt{P}(\mathtt{B})|$, i.e. the dependency measures the degree of independence between the events A and B, i.e. the fact that the occurrence of the event A is or is not dependent on the occurrence of the event B.

In the same way as lattice-based classification, frequent itemset search and association rule extraction may be used with benefit for KDD tasks. In the following, we present three real-world applications where these two data mining methods have been successfully applied to real world data.

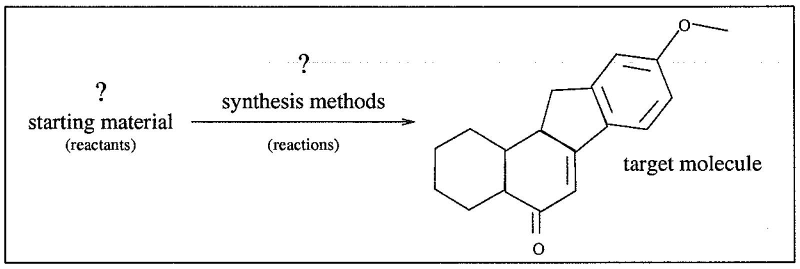


Figure 2.4: The general schema of a synthesis problem.

2.3 Applications

In the following, we detail three applications of the KDD process relying on the data mining techniques presented here-above: an experiment in mining reaction databases for organic chemistry planning, an application in mining gene expression databases in biology, and an introduction to Web mining.

2.3.1 Mining Chemical Reaction Database

In this subsection, we present an experiment on the application of knowledge discovery algorithms for mining chemical reaction databases [BLNN04b, BLNN04a]. Chemical reactions are the main elements on which relies synthesis in organic chemistry, and this is why chemical reaction databases are of first importance. Synthesis planning is mainly based on *retrosynthesis*, i.e. a goal-directed problem-solving approach, where the target molecule is iteratively transformed by applying reactions for obtaining simpler fragments, until finding accessible starting materials (see Figure 2.4). For a given target molecule, a huge number of starting materials and reactions may exist, e.g. thousands of commercially available chemical compounds. Thus, exploring all the possible pathways issued from a target molecule leads to a combinatorial explosion, and needs a strategy for choosing reaction sequences to be used within the planning process.

From a problem-solving process perspective, synthesis in organic chemistry must be considered at two main levels of abstraction: a *strategic* level, where general synthesis methods are involved, and a *tactic* level, where actual chemical reactions are applied. The present experiment is aimed at discovering generic reactions, also called *synthesis methods*, from chemical reaction databases in order to design generic and reusable synthesis plans. This can be understood in the following way: mining reaction databases at the tactic level for finding synthesis methods at the strategic level. This knowledge discovery process relies on one hand on mining algorithms, i.e. frequent itemset search and association rule extraction, and, on the other hand, on domain knowledge, that is involved at every step of the knowledge discovery process.

At present, reaction database management systems are the most useful tools for helping the chemist in synthesis planning. One aspect of the present experiment is to study how data mining techniques may contribute to knowledge extraction from reaction databases, and beyond that, to the structuring of these databases and the improvement of the database querying. Two reaction databases have been mined using frequent itemset search and association rule extraction. This experiment is original and novel within the domain of organic synthesis planning. Regarding the knowledge discovery research, this experiment stresses the fact that knowledge extraction within a complex application domain has to be guided by knowledge domain if substantial results have to be obtained.

The Chemical Context

Actually, the main questions for the synthesis chemist are related to chemical families to which a target molecule belongs, i.e. the molecule that has to be built, and to the reactions or sequence of reactions building structural patterns, to be used for building these families. Two main categories of reactions may be distinguished: reactions building the ***skeleton*** of a molecule –the arrangement of carbon atoms on which relies a molecule–, and reactions changing the ***functionality*** of a molecule, i.e. changing a function into another function (see Figure 2.5). Hereafter, we are mainly interested in reactions changing the functionality, and especially in the following question: what are the reactions allowing the transformation of a function F_i into a function F_j?

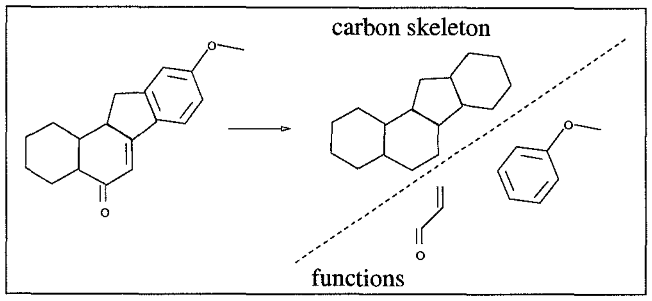


Figure 2.5: Skeleton and functional groups of a target molecule.

The experiment reported hereafter has been carried out on two reaction databases, namely the "Organic Syntheses" database ORGSYN-2000 including 5,486 records, and the "Journal of Synthetic Methods" database JSM-2002 including 75,291 records. The information items in databases such as ORGSYN-2000 and JSM-2002 may be seen as a collection of records, where every record contains one chemical equation involving structural information, that can be read, according to the reaction model, as the transformation of an *initial state* –or the set of *reactants*– into a *final state* –or the set of *products*– associated with an atom-to-atom mapping between the initial and final states (see Figure 2.6).

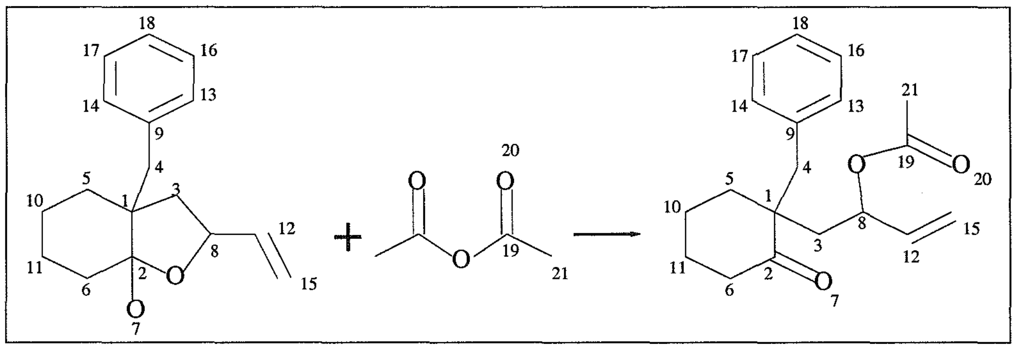


Figure 2.6: The structural information on a reaction with the associated atom-to-atom mapping (reaction #13426 in the JSM-2002 database).

The purpose of the pre-processing step of data mining is to improve the quality of the selected data by cleaning and normalizing the data. In this framework, data pre-processing has mainly consisted in exporting and analyzing the structural information recorded in the databases for extracting and for representing the functional transformations in a target format that has been processed afterwards. The considered transformations are functional modifications, functional addition and deletion, i.e. adding or deleting a function. The reactions have been considered at an abstract level, the so-called *block level* as shown in Figure 2.7. The transformation of a reaction at the block level is carried out thanks to the RESYN-ASSISTANT knowledge system [VL00, NLD94], whose objective is to help synthesis problem-solving in organic chemistry. This points out the role of knowledge and knowledge systems within the KDD process.

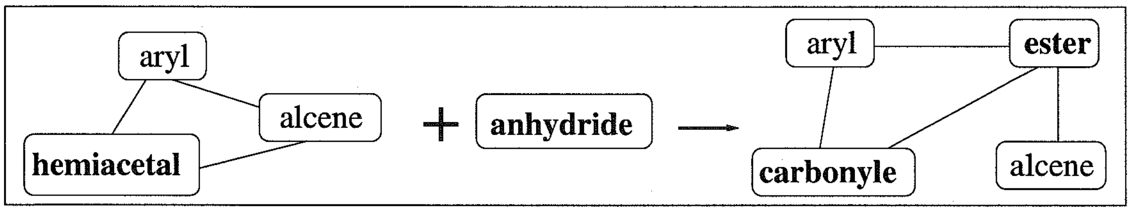


Figure 2.7: The representation of the reaction #13426 in the JSM-2002 database at the block level.

Mining of a Reaction Database

The RESYN-ASSISTANT system [VL00] has been used for recognizing the building blocks of reactions. Based on the atom-to-atom mapping, the system establishes the correspondence between the recognized blocks of the same nature, and determines their role in the reaction. A function may be present in a reactant, in a product, or in both. In the last case, the function is unchanged. In the two other cases, the function in the reactant is destroyed, or the function in the product is formed. During a reaction, either one or more reactant functions may contribute to form the functions in the products. At the end of the pre-processing step, the information obtained by the recognition process is incorporated into the representation of the reaction.

For allowing the application of the algorithms for frequent itemset search and association rule extraction, namely the *Close* algorithm [PBTL99d, PBTL99b], the data on reactions have been transformed into a binary table (losing the actual representation of a molecule as a composition of functional blocks). Then, a reaction can be considered from two main points of view (see Table 2.2):

- a global point of view on the functionality interchanges leads to consider a single entry R corresponding to a single analyzed reaction, to which a list of properties, i.e. formed and/or destroyed and/or unchanged functions, is associated,
- a specific point of view on the functionality transformations that is based on the consideration of two (or more) different entries R_k corresponding to the different functions being formed.

Both correspondences have been used during the experiment. *Close* has been applied to binary tables for generating first itemsets, i.e. sets of functions (with an associated support), and then association rules. The study of the extracted frequent itemsets may be done with different points of view. Studying frequent itemsets of length 2 or 3 enables the analyst to determine basic relations between functions. For example searching for a formed functions F_f ($_{-f}$ for formed) deriving from a destroyed function F_d ($_{-d}$ for destroyed) leads to the study of the itemsets $F_d \cup F_f$, where the symbol $\cup$ stands for the union of items or functions. In some cases, a reaction may depend on functions present in both reactants and products that remain unchanged ($_{-u}$ for unchanged) during the reaction application, leading to the study of frequent

Entries/Blocks	Destroyed		Formed		Unchanged	
	anhydride	hemiacetal	carbonyle	ester	alcene	aryle
without correspondence entry **R**	x	x	x	x	x	x
with correspondence entry $\mathbf{R}_1$	x	x		x	x	x
entry $\mathbf{R}_2$		x	x		x	x

Table 2.2: The original data are prepared for the mining task: the binary transformation of the data can be done without taking into account the atom mapping, i.e. one single line in the binary table, or by taking into account the atom mapping, i.e. two lines in the table.

itemsets such as $F_f \cup F_u \cup F_d$. This kind of itemsets can be searched and analyzed for extracting a "protection function" supposed to be stable under given experimental conditions.

The extraction of association rules gives a complementary perspective on the knowledge extraction process. For example, searching for the more frequent ways to form a function F_f from a function F_d leads to the study of rules such as $F_f \rightarrow F_d$: indeed, this rule has to be read in a retrosynthesis way, i.e. if the function F_f is formed then this means that the function F_d is destroyed. Again, this rule can be generalized in the following way: determining how a function F_f is formed from two destroyed functions F_{d1} and F_{d2}, knowing say that the function F_{d1} is actually destroyed, leads to the study of the association rules such as $F_f \cup F_{d1} \rightarrow F_{d2}$.

Looking at the Extracted Itemsets and Rules Results

A whole set of results of the application of the data mining process on the ORGSYN-2000 and JSM-2002 databases is given in [BLNN04b]. These results show that both reaction databases share many common points though they differ in terms of size and data coverage, i.e. among 500 functions included in the concept hierarchy of functional graphs within the knowledge base of the RESYN-ASSISTANT system, only 170 are retrieved from ORGSYN-2000 while 300 functions are retrieved from JSM-2002. The same five functions are ranked at the first places in both databases with the highest occurrence frequency. However, some significant differences can be observed: a given function may be much more frequent in the ORGSYN-2000 database than in JSM-2002 database, and reciprocally. These differences can be roughly explained by different data selection criteria and editor motivations for both databases.

A qualitative and statistical study of the results has shown the following behaviors. Some functions have a high stability, i.e. they mostly remain unchanged, and, on the contrary, some other functions are very reactive, i.e. they are mostly destroyed. All the reactive functions are more present in reactants than in products, and some functions are more often formed. Some functions, that are among the most widely used functions in organic synthesis, are more often present and destroyed in reactants, e.g. `alcohol` and `carboxylic acid`. For example, among the standard reactions involving functions, it is well-known –for chemists– that the `ester` function derives from a combination of two functions, one of them being mostly an `alcohol`. The search for a second function relies on the study of rules such as $\mathtt{ester_f} \cup \mathtt{alcohol_d} \rightarrow \mathtt{F_d}$. The main functions that are retrieved are `anhydride`, `carboxylic acid`, `ester`, and `acyl chloride`. If the chemist is interested in the unchanged functions, then the analysis of the rule $\mathtt{ester_f} \cup \mathtt{alcohol_d} \cup \mathtt{anhydride_d} \rightarrow \mathtt{F_u}$ gives functions such as `acetal`, `phenyl`, `alkene`, and `carboxylic acid`.

These first results provide a good overview on the function stability and reactivity. They also give partial answers to the question of knowing what are the reactions allowing the transformation of a function $\mathtt{F_i}$ into a function $\mathtt{F_j}$.

Discussion

A number of topics are discussed hereafter regarding this experiment in mining chemical reaction databases. First, it can be noticed that only a few research works hold on the application of data mining methods on reaction databases; the study on the lattice-based classification of dynamic knowledge units proposed in [GR01] has been a valuable source of inspiration for the present experiment. The abstraction of reactions within blocks and the separation in three kinds of blocks, namely formed, destroyed, and unchanged blocks, is one of the most original idea in that research work, which is responsible of the good results that have been obtained. This idea of the separation into three families may be reused in other contexts involving dynamic data. However, the transformation into a binary table has led to a loss of information, e.g. the connection information on reactions and blocks.

Frequent items or association rules are generic elements that can be used either to index (and thus organize) reactions or to retrieve reactions. Termed in another way, this means that frequent itemsets or extracted association rules may be in certain cases considered as a kind of meta-data giving meta-information on the bases that are under study.

Knowledge is used at every step of the knowledge extraction process, e.g. the coupling of the knowledge extraction process with the RESYN-ASSISTANT system, and domain ontologies such as the function ontologies, the role of the analyst, etc. Indeed, and this is one of the major lessons of this experiment: the knowledge discovery process in a specific domain such as organic synthesis has to be *knowledge-intensive*, and has to be guided by domain knowledge, and an analyst as well, for obtaining substantial results. The role of the analyst includes fixing the thresholds, and interpreting the results. The thresholds must be chosen in function of the objectives of the analyst, and in function of the content of the databases (it can be noticed that a threshold of 1% for an item support means that for a thousand of reactions, ten of them may form a reaction family, and this is not a bad hypothesis).

Moreover, the use of data mining methods such as frequent itemsets search or association rule extraction has proven to be useful, and has provided encouraging results. It could be interesting to test other (symbolic) data mining methods, and mainly relational mining for being able to take into account the structure of molecule for the data mining task [DT99, GGKS04, DL01].

2.3.2 An Experiment in Biology

In this section, we present an experiment on the mining of gene expression databases for extracting association rules, based on the article [CH03] (see also [WZTS04] for a recent overview on data mining in bioinformatics). Global gene expression profiling can be a valuable tool in the understanding of genes, biological networks, and cellular states. One goal in analyzing expression data is to try to determine how the expression of any particular gene might affect the expression of other genes; the genes involved in this case could belong to the same biological network. Another goal of analyzing expression data is to try to determine what genes are expressed as a result of certain cellular conditions, e.g. what genes are expressed in diseased cells that are not expressed in healthy cells.

As larger and larger gene expression data sets become available, data mining techniques can be applied to identify patterns of interest in the data. In [CH03] an experiment is detailed where the *Apriori* algorithm has been applied for mining association rules from gene expression data, using a set of data of 300 expression profiles for yeast. An example of extracted association rule is the following: $\{\texttt{cancer}\} \rightarrow \{\texttt{gene A} \uparrow, \texttt{gene B} \downarrow, \texttt{gene C} \uparrow\}$, meaning that, for the data set that has been mined, in most profile experiments where the cells used are cancerous, **gene A** has been measured as being up (highly expressed), **gene B** is down (low expression), and **gene C** is up. In the context of formal databases, a gene expression profile can be thought of a single transaction (corresponding to a row in a binary table), and each protein can be thought as an item. A gene expression profile transaction may include the set of genes that are up and the set of genes that are down in the profile. Items in the transaction can also include relevant facts describing the cellular environment. Moreover, in an expression profile each protein is assigned a real value that specifies the relative abundance of that protein in the profiled sample. These protein values have been made discrete for allowing the processing using standard techniques based on binary tables.

The extracted association rules that have been considered in the experiment are of the form $\{\texttt{LHS}\} \rightarrow \{\texttt{RHS}\}$, where $\{\texttt{LHS}\}$, i.e. *left hand side*, is composed of only one item, and $\{\texttt{RHS}\}$, i.e. *right hand side*, may have an arbitrary number of items. It can be noticed that such association rules (where $\{\texttt{LHS}\}$ is composed of only one item) are very interesting, as explained hereafter.

Furthermore, such rules may be used to check the validity of other rules as shown below. Let us consider the rule $X_1 \rightarrow Y \setminus X_1$, where $X_1 \subset Y$, then: $\text{supp}(X_1 \rightarrow Y \setminus X_1) = \text{supp}(X_1 \cup (Y \setminus X_1)) = \text{supp}(Y)$. If $X_1 \rightarrow Y \setminus X_1$ is a valid rule, then Y has to be a frequent itemset, and X_1, as a subset of Y, has to be frequent too. Then, any rule of the form $X_2 \rightarrow Y \setminus X_2$, where $X_1 \subseteq X_2 \subset Y$ is valid too. For example, knowing that $\{ab\} \rightarrow \{cd\}$ is valid, it can be deduced that $\{abc\} \rightarrow \{d\}$ and $\{abd\} \rightarrow \{c\}$ are valid too. This shows that the less is the length of the condition of an association rule of the form $X_1 \rightarrow Y \setminus X_1$, the more we can deduce valid rules of the form $X_2 \rightarrow Y \setminus X_2$, with $X_1 \subseteq X_2 \subset Y$. In [STB+02], minimal left hand sides of the rules are generators, and maximal right hand sides of the rules correspond to the closed itemsets related with closed sets of properties constituting the intension of the concepts in the associated lattice.

Actually, in [CH03], closed itemsets have been mainly considered, and the set of extracted association rules has been manually pruned for a better understandability of the results. In particular, this shows the importance of presenting small sets of association rules or frequent itemsets for a valuable human analysis of the results (as discussed in [CNT03] for example). Two examples of extracted association rules are the following, where the minimum support has been fixed to 10%, and the minimum confidence to 80%, and where a rule may be interpreted as follows: when the gene in $\{\texttt{LHS}\}$ is up, so are the genes in $\{\texttt{RHS}\}$. The expressions of rules have been simplified for a better readability for non-biologists.

$\{\texttt{YHM1}\} \rightarrow \{\texttt{SEQ1}, \texttt{ARO3}\}$

$\{\texttt{ARO3}\} \rightarrow \{\texttt{SEQ1}, \texttt{YHM1}\}$

where $\{\texttt{SEQ1}\} = \{\texttt{ARG1}, \texttt{ARG4}, \texttt{CTF13}, \texttt{HIS5}, \texttt{LYS1}, \texttt{RIB5}, \texttt{SNO1}, \texttt{SNZ1}, \texttt{YHR029C}, \texttt{YOL118C}\}$.

An analysis that may be of interest is the following. The genes $\{\texttt{YHM1}\}$ in the first rule and $\{\texttt{ARO3}\}$ in the second rule are found on opposite sides of the rules. The gene $\{\texttt{YHM1}\}$ has been identified as a suppressor of a gene having the property of being a binding factor. On the other hand, the gene $\{\texttt{ARO3}\}$ is activated by a binding factor itself. Whether the nature of the association suggested here between $\{\texttt{ARO3}\}$ and $\{\texttt{YHM1}\}$ has something to do with the fact that both of these genes have an association with a binding factor is an open –and very interesting– question.

The association rules that have been mined represent only a fraction of all the possible gene-to-gene interactions that remain to be discovered in yeast. More rules can be found using different search criteria, i.e. changing the support, the confidence, the data, and the form of the extracted rules. The extracted association rules can lead to the generation of new hypotheses explaining some aspects of the gene interactions, to be confirmed in wet laboratory experiments. Mining expression data for association rule extraction seems to be more useful to interpret and to understand gene networks: association rules can describe how the expression of one gene may be associated with the expression of a set of genes. It must be noticed that an association rule implies an "association" which is not necessarily a "cause and effect" relationship. Determining the precise nature of the association requires biological knowledge, as emphasized in the preceding paragraph on the mining of chemical reaction databases. This study shows that it becomes possible to develop bioinformatics applications that go further than storing and retrieving expression data, and to propose tools for exploratory data analysis.

2.3.3 An Introduction to Web Mining

In the framework of the Semantic Web, the machines are talking to machines for delivering services to people [FHLW03]. Tomorrow the Web will be a distributed, shared, declarative and navigable space; it will be mainly exploited by computers solving problems for humans, and providing the results to humans. The semantics of documents on the Web must be accessible to computers. One main element of this semantics is constituted by an explicit model of the domain of data, describing the vocabulary and the structure of informations in relation with the domain of interest. This model must be commonly accepted and shared: this is the essence of the notion of *ontology*, as it is considered in the framework of Semantic Web, and for building knowledge systems. For example, let us consider the following list of queries that leads to a series of different and complex problems:

- *A book on Béla Bartók.*
- *A book written by Béla Bartók* or *a book of Béla Bartók.*
- *A biography of Béla Bartók.*
- *An autobiography of Béla Bartók.*
- *A scorebook of Béla Bartók.*
- *A book on the work of Béla Bartók.*

For answering these questions, a computer system has to understand the actual meaning of the questions (the "intended meaning" of the user), and the system has to be able to make the difference between "on" in "a book on" and "of" in "a book of", and to understand the difference between terms such as "book", "songbook", "biography", "autobiography"... This is the purpose of ontologies in the framework of Semantic Web and Web mining [SS04]. Moreover, it can be also very useful for the system to know who is "Béla Bartók" for answering the questions above (as it should be for a human himself...).

The description of the content of documents may be made explicit by using document description languages such as XML, and a semantics can be attached to documents –and their content– using knowledge representation languages, e.g. description logics, OWL [FHLW03]. An intelligent manipulation of documents is based on the exploitation of the content and of the semantics of the documents, with respect to the knowledge on the domain of documents. The technology for the Semantic Web is based on one hand on the use of languages for ontology representation, and for document and resource description such as XML and RDF(S), and on the other hand on the use of intelligent search engines and mining modules for improving the retrieval of adequate resources for problem solving. In this way, information extraction –extraction of key terms from documents– and data mining –especially text mining– may be used for analyzing and classifying documents with respect to their content (the reference [CR04] may be of interest regarding content-based information retrieval and lattice-based classification of documents).

The *mining of documents on the Web*, or *Web mining*, can be carried out with three main points of view [KB00, BHS02]:

- The *mining of the content* of documents, in relation with text mining (see [JCK$^+$04] for example).
- The *mining of the structure* of the pages and of the links between pages (hypertext links).
- The *mining of usages* or mining the sets of operations applied to pages.

Web mining can be a major technique in the design of Semantic Web: on that base, ontologies can be designed in a semi-automatic way, leading the real-scale ontologies, semi-automatic design rather than manual design of ontologies. Ontologies can be used for *annotating* the documents, and thus to enhance the document mining process, on the base of the content of documents. The Web mining process can be used to improve annotation of documents, and thus the semantics attached to the documents, i.e. content, understandability, and structure.

Moreover, information retrieval can be guided by document mining: key terms are extracted and used to complete domain ontologies, that are in turn used for guiding the data mining process, and so on. Knowledge units extracted from documents can be used for classifying documents according to relations between units and the domain ontology, leading to alternative points of view on documents.

2.4 Discussion

The KDD process must be carried out in a KDD environment where data mining is guided by domain knowledge, embedded in ontologies and knowledge-based systems. The knowledge units used in knowledge systems may have two major different sources: explicit knowledge that can be given by domain experts, and implicit knowledge that must be extracted from databases of different kinds, e.g. rough data or textual documents. In addition, an important question in the framework of Semantic Web and Web mining for improving the KDD process is to be able to manipulate documents by their content, for searching, for annotating and for classifying the documents. The content-based manipulation of documents allows solving a number of problems such as information extraction, intelligent information retrieval, content-based document mining... More precisely, the following requirements for knowledge discovery tools are given in [BA96]:

- The system should represent and present to the user the underlying domain in a natural and appropriate fashion. Objects of the domain should be easily incorporated into queries.
- The domain representation should be extendible by the addition of new concepts or classes formed from queries. These concepts and their representative individuals must be usable in subsequent queries.

- It should be easy to form tentative segmentations of data, to investigate the segments, and to re-segment quickly and easily. There should be a powerful repertoire of viewing and analysis methods, and these methods should be applicable to segments (such as in the WEKA system for example [WF00]).
- Analysts should be supported in recognizing and abstracting common analysis (segmenting and viewing) patterns. These patterns must be easy to apply and modify.
- There should be facilities for monitoring changes in classes or concepts over time.
- The system should increase the transparency of the knowledge discovery process and should document its different stages.
- Analysis tools should take advantage of explicitly represented background knowledge of domain experts, but should also activate the implicit knowledge of experts.
- The system should allow highly flexible processes of knowledge discovery respecting the open and procedural nature of productive human thinking. This means in particular to support the intersubjective communication and argumentation.

The support of knowledge discovery by concept lattices, itemset search and association rule extraction, may be explained as follows [Wil02]. The mathematization of logical structures of concepts and concept hierarchies by formal concepts and concept lattices of formal contexts yields a close relationship between logical and mathematical thinking, which, in particular, allows activating a rich amount of mathematics to support human reasoning. Especially, the representation of concept lattices by labeled line diagrams enables an interplay between the mathematical analysis of relationships and the logical analysis of data and information, influenced by already existing background knowledge. Therefore, conceptual knowledge discovery, i.e. conceptual information discovery and knowledge creation, can be performed by first looking under the guidance of some purpose for discoveries of information in graphically represented concept lattices, and then creating new knowledge from the discovered information and appropriate pre-knowledge. These two steps should be repeated in a circular process which is open for critic and self-correction.

2.5 Conclusion

The knowledge discovery in databases process consists in processing a huge volume of data in order to extract knowledge units that can be reused either by an expert of the domain of data or by a knowledge-based system for problem-solving in the domain of data. The KDD process is based on three major steps, data preparation, data mining and interpretation of the extracted units. Moreover, the KDD process is iterative and interactive, and is controlled by an analyst, who is in charge of guiding and validating the extraction process. In addition, the KDD process may take advantage of domain knowledge, i.e. ontologies, knowledge base, for improving the process at every step. Data mining methods are divided into two main categories, symbolic and numerical methods. In this chapter, we have mainly focused on symbolic methods, and especially on lattice-based classification, frequent itemset search, and association rule extraction. These methods are operational and can provide good results in real-world problems. Indeed, three kinds of application have been detailed, an experiment on the mining of chemical reaction databases, an experiment on the mining of gene expression databases, and finally, a research field with a growing importance, Web mining.

Regarding the future of KDD, there remains many problems to be solved, at every step of the process, especially considering the KDD process as a knowledge-guided process, as we have tried to demonstrate it, and considering the complete environment of a KDD system as a combination of a database and of a knowledge base operations. Another important investigation field for symbolic methods is the extension to the processing of complex data (contrasting with binary data). Finally, let us mention that important challenges are linked to the application domains, and must still be undertaken, e.g. biology, chemistry, medicine, space, weather forecast, finance,... In Chapter 1, we have compared knowledge discovery to gold research or archaeology: first, it is necessary to get to know the domain of data, then to apply a number of data mining methods that produce more or less useful results, and then to validate these results. Meanwhile, the analyst has to be patient because the process is iterative –the work may be long without being successful– but it is worth continuing the job, being confident and optimistic!

Chapter 3

Frequent Itemset Search

In this chapter, we present in depth concepts and algorithms of frequent itemset search. The chapter is organized as follows. Section 3.1 presents the basic concepts. In Section 3.2, we classify algorithms in four families, such as *levelwise*, *vertical*, *hybrid* and *other* algorithms. Selected algorithms namely *Zart*, *Eclat-Z* and *Charm-MFI* are detailed in Section 3.3.

3.1 Basic Concepts

Frequent Itemsets. Below we use standard definitions of Data Mining. We consider a set of *objects* or *transactions* $O = \{o_1, o_2, \ldots, o_m\}$, a set of *attributes* or *items* $A = \{a_1, a_2, \ldots, a_n\}$, and a relation $R \subseteq O \times A$, where $R(o, a)$ means that the object o has the attribute a. In formal concept analysis [GW99] the triple (O, A, R) is called a *formal context*. A set of items is called an *itemset*[6] or a *pattern*. Each transaction has a unique identifier (*tid*), and a set of transactions is called a *tidset*. The *length* of an itemset is the cardinality of the itemset, i.e. the number of items included in the itemset. An itemset of length i is called an i-long itemset, or simply an i-itemset[7]. An itemset P is said to be *larger* (resp. *smaller*) than Q if $|P| > |Q|$ (resp. $|P| < |Q|$). We say that an itemset $P \subseteq A$ is *included* in an object $o \in O$, if $(o, p) \in R$ for all $p \in P$. Let f be the function that assigns to each itemset $P \subseteq A$ the set of all objects that include P: $f(P) = \{o \in O \mid o \text{ includes } P\}$. The (absolute) *support* of an itemset P indicates how many objects include the itemset, i.e. $supp(P) = |f(P)|$. The support of an itemset P can also be defined in relative value, which corresponds to the proportion of objects including P, with respect to the whole population of objects. To distinguish relative support from absolute support, we denote them by *rsupp* and *supp*, respectively.[8] Thus, the *relative support* of an itemset $P \subseteq A$ is: $rsupp(P) = |f(P)|/|O| = supp(P)/|O|$. An itemset P is called *frequent*, if its support is not less than a given *minimum support* (below often denoted by *min_supp*), i.e. $supp(P) \geq min\_supp$. An itemset G is called *generator* if it has no proper subset H ($H \subset G$) with the same support. An itemset X is called *closed* if there exists no proper superset Y ($X \subset Y$) with the same support. The *closure* of an itemset X (denoted by $\gamma(X)$) is the largest superset of X with the same support. Naturally, if $X = \gamma(X)$, then X is a closed itemset. A frequent itemset is a *maximal frequent itemset* (MFI) if all its proper supersets are not frequent.

[6]Following standard FCA notations, we will use separator-free set notations throughout the thesis, e.g. AB stands for $\{A, B\}$.

[7]For instance, ABE is a 3-itemset.

[8]Note that throughout the thesis, we use *absolute support* everywhere, except for the quality measures in Section 4.4.

MFIs are closed itemsets. The task of frequent (closed) itemset mining consists of generating all (closed) itemsets (with their supports) with supports greater than or equal to a specified *min_supp*.

Frequent Association Rules. An association rule is an expression of the form $P_1 \rightarrow P_2$, where P_1 and P_2 are arbitrary itemsets ($P_1, P_2 \in A$), $P_1 \cap P_2 = \emptyset$ and $P_2 \neq \emptyset$. The left side, P_1 is called *antecedent*, the right side, P_2 is called *consequent.*[9] The support of an association rule r: $P_1 \rightarrow P_2$ is defined as: $supp(r) = supp(P_1 \cup P_2)$. An association rule r is called *frequent*, if its support is not less than a given *minimum support* (below often denoted by *min_supp*), i.e. $supp(r) \geq min\_supp$. The *confidence* of an association rule r: $P_1 \rightarrow P_2$ is defined as the conditional probability that an object includes P_2, given that it includes P_1: $conf(r) = supp(P_1 \cup P_2)/supp(P_1)$. An association rule r is called *confident*, if its confidence is not less than a given *minimum confidence* (below often denoted by min_conf), i.e. $conf(r) \geq min\_conf$. An association rule r with $conf(r) = 1.0$ (i.e. 100%) is an *exact* association rule, otherwise it is an *approximate* association rule. A frequent association rule is *valid* or *strong* (their set is denoted by $\mathcal{AR}$) if it is both frequent and confident, i.e. $supp(r) \geq min\_supp$ and $conf(r) \geq min\_conf$. The problem of mining frequent association rules in a database D consists of finding all frequent valid rules in the database.

3.2 Classification of Itemset Mining Algorithms

3.2.1 Levelwise Algorithms

In this subsection, we present levelwise algorithms in a general way. The most well-known algorithm of this kind, without doubt, is *Apriori*. This algorithm addresses the problem of finding all frequent itemsets in a dataset. *Apriori* has been followed by lots of variations, and several of these levelwise algorithms concentrate on a special subset of frequent itemsets, like closed itemsets or generators. We present shortly some of these algorithms too. Mannila and Toivonen provided a general framework for levelwise algorithms in [MT97].

The levelwise algorithm for finding all FIs is a breadth-first, bottom-up algorithm, which means the following. First it finds all 1-long frequent itemsets[10], then at each i^{th} iteration it identifies all i-long frequent itemsets. The algorithm stops when it has identified the largest frequent itemset. Frequent itemsets are computed iteratively, in ascending order by their length. At each iteration one database pass is needed to count support values, thus the number of database passes is equal to the length of the largest frequent itemset. This approach is very simple and efficient for sparse, weakly correlated data. The levelwise algorithm is based on two basic properties.

Property 3.1 (downward closure) *All subsets of a frequent itemset are frequent.*[11]

Property 3.2 (anti-monotonocity) *All supersets of a non-frequent itemset are non-frequent.*

Consider the following dataset D (Table 3.1) that we will use for examples throughout the thesis. Figure 3.1 shows the powerset lattice of this dataset. By defining $min\_supp = 3$, the

[9]Note that the union of the antecedent and consequent is also called sometimes the "base" of a rule [Kry02].
[10]That is, first it identifies all frequent items (attributes).
[11]The name of the property comes from the fact that the set of frequent itemsets is closed w.r.t. set inclusion.

	A	B	C	D	E
1	x	x		x	x
2	x		x		
3	x	x	x		x
4		x	x		x
5	x	x	x		x

Table 3.1: A toy dataset (D) for the examples.

itemsets can be grouped in two sets: frequent and non-frequent (or rare) itemsets. Between the two sets a border can be drawn that cuts the itemset lattice in a positive and a negative space. Frequent itemsets are on the positive side of the border. (The concept of the border theory was introduced in [MT97], and we also investigate it in detail in Chapter 5 with the rare itemsets). At the bottom of the lattice we can find the smallest itemset, the empty set. At each level there are itemsets of the same length. At the top of the lattice we can find the *largest itemset* that contains all the attributes. At each itemset at the top right-hand corner the support of the itemset is indicated. The task of frequent itemset mining consists of generating all itemsets that are on the positive side of the border.

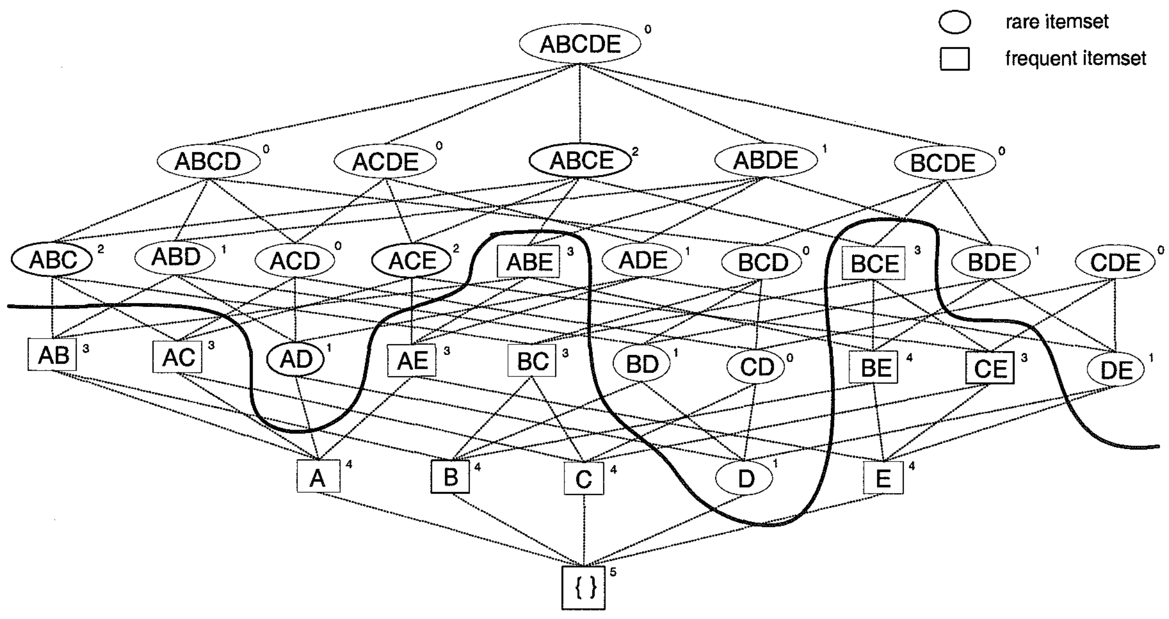


Figure 3.1: Powerset lattice of database D (Table 3.1).

Levelwise Exploration of the Positive Side of the Border

The levelwise algorithm discovers frequent itemsets of the itemset lattice in a levelwise manner, i.e. at each level i it only uses i-long frequent itemsets to generate their $(i+1)$-long supersets. These supersets are called *candidates*, and only potentially frequent itemsets are kept. For storing itemsets, two kinds of tables are used: F_i for i-long frequent itemsets, and C_i for potentially frequent i-long candidates. An itemset of length $(i+1)$ is called *potentially frequent* if all its

i-long subsets are frequent. Otherwise, if it has an i-long subset not present in F_i, then it means that it has an infrequent subset, and by Property 3.2 the candidate is also infrequent and can be pruned. With one database pass the support of potentially frequent candidates is counted, and itemsets that turn out to be infrequent are pruned. The frequent $(i+1)$-long itemsets are used then to generate $(i+2)$-long candidates, etc. The process continues until no new candidates can be generated. The generation of $(i+1)$-long potentially frequent candidates from i-long frequent itemsets consists of two steps. First, in the *join* step, table F_i is joined with itself. The union $p \cup q$ of itemsets $p, q \in F_i$ is inserted in C_{i+1} if they share their $i-1$ first items. Next, in the *prune* step, candidates in C_{i+1} are deleted if they have an i-long subset not present in F_i. This way infrequent itemsets are pruned (it is not necessary to count their supports), and only potentially frequent candidates are kept in C_{i+1}.

Note that the itemsets that are potentially frequent candidates, but turn out to be infrequent, form the set of *minimal rare itemsets*. A minimal rare itemset is a rare itemset such that all its proper subsets are frequent (and necessarily, all its supersets are not frequent). The set of minimal rare itemsets is also known as *negative border* [MT97]. By the two basic properties, the levelwise algorithm guarantees that after having found a minimal rare itemset, it will not generate any of its supersets later. As a consequence, supports of proper supersets of minimal rare itemsets will never be counted, and this is the way that the levelwise algorithm reduces the search space in the powerset lattice (see Figure 3.1).

As an example of the join step, consider the F_2 table of dataset D: {AB, AC, AE, BC, BE, CE}. After the join step, C_3 is: {ABC, ABE, ACE, BCE}. Since all their 2-long subsets are present in F_2 (which means that all their subsets are frequent), they are also potentially frequent and kept in C_3. That is, the prune step will not delete any elements of C_3.

The candidate generation and the support counting process require a subset test. In candidate generation, having an $(i+1)$-long candidate, we need to identify its subsets in F_i. In the support counting process, the dataset is read object by object. We need to find the subsets of the corresponding itemset of each object (i.e. the itemset that is included by the given object) in C_k, and the support value of each subset in C_k must be incremented by 1. As it can be seen, these subset operations must be performed *lots of* times, thus it must be implemented in a very efficient way for a good performance. There are two commonly used data structure for subset test: hash-tree, as suggested in [AMS$^+$96], and trie (a.k.a. prefix-tree) [AHU85, PBTL99c]. Note that for all of our implementations of levelwise algorithms we have used the trie data structure. For the description of this data structure, see Appendix C.1; for the description of the subset function, see Appendix C.2.

Short Overview of Levelwise Algorithms

Apriori, a concrete realization of the levelwise algorithm, is presented in detail in Appendix B.1. *Apriori-Close*, detailed in Appendix B.2, extends *Apriori* with the possibility to find not only frequent but frequent closed itemsets too. *Pascal*, a very efficient improvement of *Apriori*, is presented together with *Zart* in Section 3.3.1. *Pascal* uses the so-called pattern counting inference, and thanks to this technique it can greatly reduce the expensive database passes and support counts. *Pascal* also finds the frequent generators. *Zart*, presented in Section 3.3.1, is a practical extension of *Pascal*. It was designed to facilitate the generation of a special set of rules (called minimal non-redundant association rules). To achieve it, *Zart* finds frequent closed itemsets too, and it associates generators to their closures. *Close*, that we also implemented in our platform CORON, is also a levelwise algorithm for finding frequent closed itemsets. First it finds a subset of generators, and then it calculates their closures.

3.2.2 Vertical Algorithms

In this subsection, we present the common parts of two well-known algorithms of Zaki, namely *Eclat* and *Charm*. This subsection mainly relies on [ZPOL97], [Zak00] and [ZH02].

Short Overview of Vertical Algorithms

Eclat was the first successful algorithm proposed to generate all frequent itemsets in a depth-first manner. *Charm* is a modification of *Eclat* to explore frequent closed itemsets only. Both algorithms use a vertical layout of the database. In this way, the support of an itemset can be easily computed by a simple intersection operation. As we will see in the experiments, the vertical database layout shows significant performance improvements over levelwise algorithms that usually use horizontal representation.

Basic concepts. Here we would like to present the necessary concepts specific to *Eclat* and *Charm* using the terminology of Zaki. Let $\mathcal{I}$ be a set of items, and $\mathcal{D}$ a database of transactions, where each transaction[12] has a unique identifier (*tid*) and contains a set of items. The set of all tids is denoted as $\mathcal{T}$. A set of items is called an *itemset*, and a set of transactions is called a *tidset*. For convenience we write an itemset $\{A, B, E\}$ as ABE, and a tidset $\{2,3\}$ as 23. For an itemset X, we denote its corresponding tidset as $t(X)$, i.e. the set of all tids that contain X as a subset. For a tidset Y, we denote its corresponding itemset as $i(Y)$, i.e. the set of items common to all the tids in Y. Note that $t(X) = \bigcap_{x \in X} t(x)$, and $i(Y) = \bigcap_{y \in Y} i(y)$. For instance, using our dataset D (Table 3.1), $t(ABE) = t(A) \cap t(B) \cap t(E) = 1235 \cap 1345 \cap 1345 = 135$ and $i(23) = i(2) \cap i(3) = AC \cap ABCE = AC$. The support of an itemset X is equal to the cardinality of its tid-list, i.e. $supp(X) = |t(X)|$.

Lemma 3.1 *Let X and Y be two itemsets. Then, $X \subseteq Y \Rightarrow t(X) \supseteq t(Y)$.*

Proof. Follows from the definition of support. □

Itemset-tidset search tree and prefix-based equivalence classes. Let $\mathcal{I}$ be the set of items. Define a function $p(X, k) = X[1 : k]$ as the k length prefix of X, and a *prefix-based* equivalence relation θ_k on itemsets as follows: $\forall X, Y \subseteq \mathcal{I}, X \equiv_{\theta_k} Y \iff p(X, k) = p(Y, k)$. That is, two itemsets are in the same k-class if they share a common k-length prefix.

Eclat and *Charm* perform a search for frequent (closed) itemsets over a so-called IT-tree search space, as shown in Figure 3.2. While most previous methods exploit only the itemset search space, *Eclat* and *Charm* simultaneously explore both the itemset space *and* the transaction space. Each node in the IT-tree, called an IT-node, represented by an itemset-tidset pair, $X \times t(X)$, is in fact a prefix-based equivalence class (or simply a prefix-based class). All the children of a given node X belong to its prefix-based class, since they all share the same prefix X. We denote a prefix-based class as $[P] = \{l_1, l_2, \ldots l_n\}$, where P is the parent node (the prefix), and each l_i is a single item, representing the node $Pl_i \times t(Pl_i)$. For example, the root of the tree corresponds to the class $[\,] = \{A, B, C, D, E\}$. The left-most child of the root consists of the class $[A]$ of all itemsets containing A as the prefix. Each class member represents one child of the parent node. A class represents items that the prefix can be extended with to obtain a new frequent node. Clearly, no subtree of an infrequent prefix has to be examined. The power of the prefix-based class approach is that it breaks the original search space into independent sub-problems. When

[12]In the case of levelwise algorithms, we used the FCA terminology "objects" instead of "transactions".

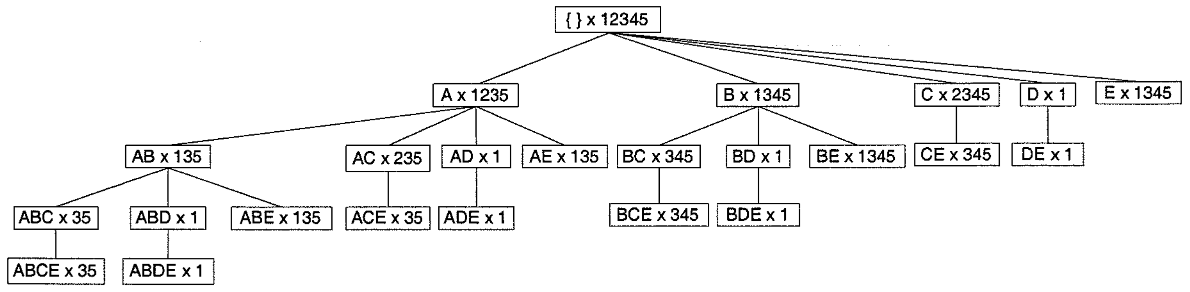


Figure 3.2: IT-tree: Itemset-Tidset search tree of dataset D (Table 3.1).

all direct children of a node X are known, one can treat it as a completely new problem; one can enumerate the itemsets under it and simply prefix them with the item X, and so on.

Lemma 3.1 states that if X is a subset of Y, then the cardinality of the tid-list of Y (i.e. its support) must be less than or equal to the cardinality of the tid-list of X. A practical and important consequence of this lemma is that the cardinalities of intermediate tid-lists shrink as we descend in the IT-tree. This results in very fast intersection and support counting.

Vertical layout. It is necessary to access the dataset in order to determine the support of a collection of itemsets. Itemset mining algorithms work on binary tables, and such a database can be represented by a binary two-dimensional matrix. There are two commonly used layout for the implementation of such a matrix: *horizontal* and *vertical* data layout. Levelwise algorithms use horizontal layout. *Eclat* and *Charm* use instead vertical layout, in which the database consists of a set of items and their tid-lists. To count the support of an itemset X using the horizontal layout, we need one full database pass to test for every transaction T if $X \subseteq T$. For a large collection of itemsets, this can be done at once using the trie data structure. The vertical layout has the advantage that the support of an itemset can be computed by a simple intersection operation. In [Zak00], it is shown that the support of any k-itemset can be determined by intersecting the tid-lists of any two of its $(k-1)$-long subsets. A simple check on the cardinality of the resulting tid-list tells us whether the new itemset is frequent or not. It means that in the IT-tree, only the lexicographically first two subsets at the previous level are required to compute the support of an itemset at any level. One layout can be easily transformed to the other layout on-the-fly (see Appendix D for details).

Other Optimizations

Element reordering. As pointed out in [Goe03] and [CG05], *Eclat* does not fully exploit the monotonocity property. It generates a candidate itemset based on only two of its subsets, thus the number of candidate itemsets is much larger as compared to breadth-first approaches such as *Apriori*. *Eclat* essentially generates candidate itemsets using only the join step of *Apriori*, since the itemsets necessary for the prune step are not available due to the depth-first search. A technique that is regularly used is to reorder the items in support ascending order, which leads to the generation of less candidates. In *Eclat* and *Charm*, such reordering can be performed on the children of a node N when all direct children of N are discovered. Experimental evaluations show that item reordering results in significant performance gains in the case of both algorithms.

Support count of 2-itemsets. It is well known that many itemsets of length 2 turn out to be infrequent. A naive implementation for computing the frequent 2-itemsets requires $n(n-1)/2$ intersection operations, where n is the number of frequent 1-items. Considering that 1-items have the largest tid-lists (see Lemma 3.1), these operations are quite expensive. Here we present a method that can be used not only for depth-first, but for breadth-first algorithms too, such as *Apriori*. First, the database must be transformed in horizontal format (see Appendix D). Secondly, through a database pass on the horizontal layout, an upper-triangular 2D matrix is built containing the support values of 2-itemsets [ZH02]. Consult Appendix E for a detailed description and an example.

Diffsets for further optimizing memory usage. Recently, Zaki proposed a new approach to efficiently compute the support of an itemset using the vertical data layout [ZG03]. Instead of storing the tidset of a k-itemset P in a node, the difference between the tidset of P and the tidset of the $(k-1)$-prefix of P is stored, denoted by the *diffset* of P. To compute the support of P, we simply need to subtract the cardinality of the diffset from the support of its $(k-1)$-prefix. Support values can be stored in each node as an additional information. The diffset of an itemset $P \cup \{i, j\}$, given the two diffsets of its subsets $P \cup \{i\}$ and $P \cup \{j\}$, with $i < j$, is computed as follows: $\text{diffset}(P \cup \{i, j\}) \leftarrow \text{diffset}(P \cup \{j\}) \setminus \text{diffset}(P \cup \{i\})$. Diffsets also shrink as larger itemsets are found. Diffsets can be used together with the other optimizations presented above. This technique can significantly reduce the size of memory required to store intermediate results. Diffsets can be used for both *Eclat* and *Charm*, resulting in *dEclat* and *dCharm*, respectively. Note that we have not used diffsets in our implementations yet.[13]

Conclusion

In this subsection we presented the common parts of *Eclat* and *Charm*. Detailed description of *Eclat* can be found in Appendix B.3. *Charm* is presented in Appendix B.4. Section 3.3.2 presents an algorithm called *Eclat-Z*, which is based on *Eclat* and produces the same result as *Zart*, i.e. a result that can be used directly to generate minimal non-redundant association rules. Finally, Section 3.3.3 presents *Charm-MFI*, a simple extension of *Charm*. *Charm-MFI* marks maximal frequent itemsets among frequent closed itemsets.

3.2.3 Hybrid Algorithms

Under "hybrid" we mean such algorithms that are a combination of levelwise and vertical algorithms. We present two algorithms of this type, namely *Eclat-Z* (Section 3.3.2) and *Charm-MFI* (Section 3.3.3). *Eclat-Z* is composed of two parts: first all FIs are found by *Eclat*, then these FIs are post-processed in a levelwise manner, i.e. in ascending order by their length. During this post-processing step generators and closed itemsets are marked among FIs, and then generators are associated to their closures. *Charm-MFI* finds FCIs with the help of *Charm*, then it also post-processes FCIs in ascending order by their length in order to filter maximal frequent itemsets.

3.2.4 Other Algorithms

In our work we have examined algorithms of the previous three types (levelwise, vertical and hybrid algorithms), but there exist other algorithms too. Maybe the most well-known is *FP-growth*

[13]We plan to investigate this technique as a future perspective.

by Han *et al.* [HPY00]. *FP-growth* is also a depth-first algorithm that uses a new data structure called FP-tree (frequent pattern tree), which is a compressed representation of all the transactions in the database. Every item has a linked list going through all transactions that contain that item. The FP-tree structure is a lossless representation of the complete dataset for the generation of frequent itemsets. *FP-growth* also has several modifications, like *Closet* [PHM00] or *Closet*$^+$ [WHP03] that are designed to extract frequent closed itemsets only.

3.3 Detailed Description of Selected Algorithms

In the forthcoming sections, the following algorithms will be presented in details: *Zart* (Section 3.3.1), *Eclat-Z* (Section 3.3.2) and *Charm-MFI* (Section 3.3.3). In the appendices we present the following classical algorithms: *Apriori* (Appendix B.1), *Apriori-Close* (Appendix B.2), *Eclat* (Appendix B.3) and *Charm* (Appendix B.4). The *Pascal* and *Pascal*$^+$ algorithms are presented together with *Zart* in Section 3.3.1.

3.3.1 Zart

Short Overview of the Zart Algorithm

In this subsection we present a multifunctional itemset mining algorithm called *Zart*[14], which is a refinement of the *Pascal* algorithm. We have designed *Zart* for this thesis work. The most well-known algorithm for finding frequent itemsets is *Apriori*. The main problem with *Apriori* and most of its variations is the high number of operations required for counting support values. To overpass this drawback, a new algorithm called *Pascal* was proposed, which introduced the notion of *pattern counting inference*. Using this technique, the support of an itemset can be determined without accessing the database if its so-called *generators* are already known. We propose a refinement of this algorithm, called Zart, which extends *Pascal* in certain ways: it can simultaneously identify *frequent closed itemsets*, and in parallel it can *associate* the generators to their closures. At present, there are no algorithms that propose to extract at the same time frequent itemsets, frequent closed itemsets, and their associated frequent generators. We show that these refinements are necessary and sufficient conditions for finding minimal non-redundant association rules.

Contribution and Motivation

Minimal non-redundant association rules ($\mathcal{MNR}$) have the following form: $P \rightarrow Q \setminus P$, where $P \subset Q$ and P is a *generator* and Q is a *closed itemset*. In this part of our work we are interested to find these rules because of two reasons. First, this set of rules is *lossless* (enables the derivation of all valid rules), *sound* (forbids the derivation of rules that are not valid) and *informative* (allows the determination of rules parameters such as support and confidence). In [Kry02], it is shown that with the so-called cover operator, which is an inference mechanism, all valid rules can be restored from these rules with their proper support and confidence values. Secondly, among rules with the same support and same confidence, these rules contain the most information and these rules can be the most useful in practice [Pas00b]. See Section 4.3.1 for the whole explanation.

The minimal non-redundant association rules were introduced in [BTP+00b]. In [BTP+00a] and [BTP+02] Bastide *et al.* presented *Pascal*, and claimed that $\mathcal{MNR}$ can be extracted with this algorithm. However, we do not agree with them because of two reasons. First, frequent closed itemsets must also be known. Secondly, frequent generators must be *associated* to their closures. Figure 3.3 shows what information is provided by *Pascal* when applied on dataset D (Table 3.1), i.e. it finds frequent itemsets and marks frequent generators. Unfortunately, this information is insufficient. However, with an extension *Pascal* can be enriched to fulfill the previous two criteria. This is the so-called *Zart* algorithm, whose result is shown in Figure 3.4. Obviously, this result is necessary and sufficient to generate minimal non-redundant association rules.

We have chosen *Pascal* because of the following reasons. First, among levelwise frequent itemset mining algorithms it may be the most efficient thanks to its pattern counting inference mechanism. *Pascal* also marks which frequent itemsets are generators. This result serves as a good basis for identifying closed itemsets and then associating generators to them. Secondly, the authors of *Pascal* claimed that this algorithm is capable of extracting $\mathcal{MNR}$. We show that it is possible with our refinement.

Furthermore, the idea introduced in *Zart* can be generalized, and thus it can be used with *any* frequent itemset mining algorithm. As we will see in *Eclat-Z* (Section 3.3.2), it is not

[14]The name of the algorithm comes from the Hungarian word "zárt", which means "closed".

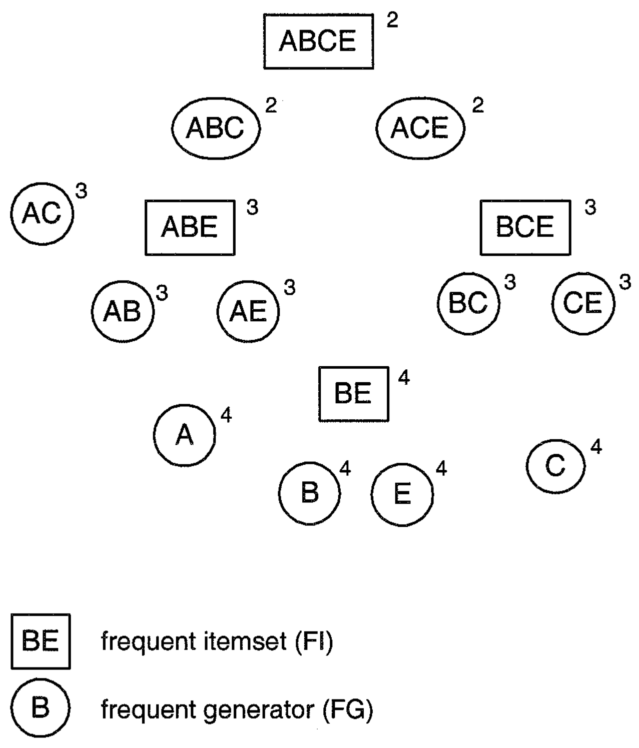


Figure 3.3: Result of *Pascal* on D (Table 3.1) with $min\_supp = 2$ (40%).

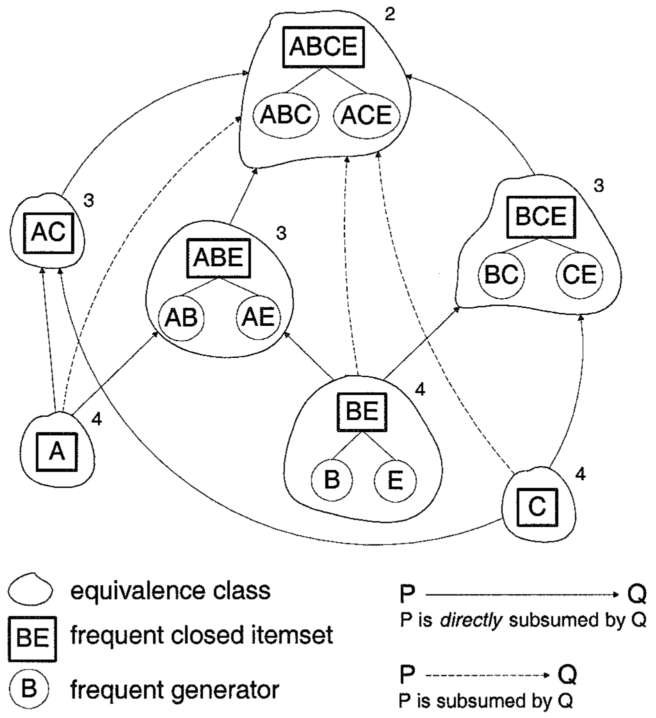


Figure 3.4: Result of *Zart* on D (Table 3.1) with $min\_supp = 2$ (40%). Note that information on the subsumption relation (see Def. 3.5) is provided by the trie data structure.

even necessary that frequent itemsets be generated in ascending order by their length. That is, with the generalized idea of *Zart* any frequent itemset mining algorithm can be turned into an algorithm that supports the extraction of minimal non-redundant association rules.

Detailed Description of Zart

Zart is a levelwise algorithm that enumerates candidate itemsets in ascending order by their length. It means that the generators of an equivalence class are found first. Their support is calculated, and later when finding other (larger) elements of the equivalence class, their support does not have to be counted since it is equal to the support of the generators that are already known. *Apriori* has a serious drawback: it has to count the support of *each* candidate itemset, and it necessitates one whole database pass at each iteration. Due to the counting inference support, the number of expensive database passes and support counts can be reduced seriously, especially in the case of dense, highly correlated data.

Shortly, *Zart* works in the following way: as it is based on *Pascal*, first it finds frequent itemsets and marks frequent generators. Then, it filters frequent closed itemsets among frequent itemsets, like *Apriori-Close*. The idea is that an itemset is not closed if it has a superset with the same support. Thus, if at the i^{th} iteration an itemset has a subset of size $(i-1)$ with the same support, then the subset is not a closed itemset. This way all frequent closed itemsets can be found. The last step consists in associating generators to their closures. This can be done by collecting the non-closed, generator subsets of the given closed itemset that have the same support.

Thus, *Zart* has three main characteristics, namely **(1)** pattern counting inference, **(2)** identifying frequent closed itemsets, and **(3)** identifying generators of frequent closed itemsets. In this section we give the theoretical basis of *Zart*.

Pattern counting inference. The first part of *Zart* is based on *Pascal*, thus the definitions of this paragraph mainly rely on [BTP$^+$00a]. *Pascal* introduced *pattern counting inference*, which is based on the following observation. Frequent itemsets in a database are not completely independent one from another. Itemsets that are common to the same set of objects belong to the same equivalence class. In a class three kinds of elements are distinguished: the maximal element, the minimal element(s) (w.r.t. set inclusion), and all other elements. The maximal element, which is a unique element, is the closure of all the elements in the class, thus it is a frequent closed itemset. The minimal elements are called generators.[15] An equivalence class has the special property that all its elements have exactly the same support. This is the key idea behind the counting inference.

Levelwise algorithms are based on two basic properties of *Apriori*, namely *downward closure* (see Def. 3.1) and *anti-monotonicity* (see Def. 3.2). Like *Apriori* or *Pascal*, *Zart* traverses the powerset lattice of a database in a levelwise manner. At the i^{th} iteration, the algorithm first generates candidate i-itemsets. Using the *Apriori* properties, only the potentially frequent candidates are kept, i.e. those whose $(i-1)$-long subsets are all frequent. After this, with one database pass, the support of all candidate i-long itemsets is determined.

Pattern counting inference is based on the observation that frequent itemsets can be grouped in equivalence classes. All itemsets in a class are equivalent, in the sense that they describe exactly the same set of objects:

[15]In the literature these itemsets have various names: key itemsets, minimal generators, free-itemsets, etc. Throughout the thesis we will refer to them as "generators" or "key generators".

Definition 3.1 (equivalence class) *Let f be the function that assigns to each itemset $P \subseteq A$ the set of all objects that include P: $f(P) = \{o \in O \mid o$ includes $P\}$. Two itemsets $P, Q \subseteq A$ are said to be* equivalent *($P \cong Q$) iff $f(P) = f(Q)$. The set of itemsets that are equivalent to an itemset P (also called P's* equivalence class*) is denoted by $[P] = \{Q \subseteq A \mid P \cong Q\}$.*

If P and Q are equivalent ($P \cong Q$), then their support is the same:

Lemma 3.2 *Let P and Q be two itemsets.*
(i) $P \cong Q \Rightarrow supp(P) = supp(Q)$
(ii) $P \subseteq Q$ and $(supp(P) = supp(Q)) \Rightarrow P \cong Q$

Definition 3.2 *An itemset $P \in [P]$ is called a* generator *(or* key generator*), if P has no proper subset in $[P]$, i.e. it has no proper subset with the same support. It means that generators are* minimal elements *in their equivalence classes (w.r.t. set inclusion). A* candidate generator *is an itemset such that all its proper subsets are generators.*

Property 3.3 (downward closure for generators) *All subsets of a frequent generator are frequent generators.*

Property 3.3 can be generalized the following way:

Property 3.4 *All subsets of a generator are generators [Kry01].*

Property 3.5 (anti-monotonocity for generators) *If an itemset is not a (frequent) generator, then none of its supersets are (frequent) generators.*

Theorem 3.1 *Let P be a frequent itemset.*
(i) Let $p \in P$. Then $P \in [P \setminus \{p\}]$ iff $supp(P) = supp(P \setminus \{p\})$.
(ii) P is a generator iff $supp(P) \neq min_{p \in P}(supp(P \setminus \{p\}))$.

Theorem 3.2 *If P is not a generator, then $supp(P) = min_{p \in P}(supp(P \setminus \{p\}))$.*

The proofs of Properties 3.3, 3.5 and Theorems 3.1 and 3.2 can be found in [BTP$^+$00a]. Property 3.4 is proven in [Kry01].

Let $max[P]$ be the maximal element of the equivalence class of P, i.e. $max[P]$ is the closure of the class of P. Let $min[P]$ be the set of minimal elements (w.r.t. set inclusion) of the equivalence class of P, i.e. $min[P]$ is the set of generators of the class of P ($|min[P]| \geq 1$).

Definition 3.3 *An equivalence class P is* simple *if P has only one generator and this generator is equivalent to the closure of P.*

Definition 3.4 *An equivalence class P is* complex *if P has at least one generator that is not equivalent to the closure of P.*

This distinction is interesting from the point of view of rule extraction. For example, the generic basis (see Def. 4.3) can only be generated from *complex* equivalence classes.

Definition 3.5 (subsumption relation on equivalence classes) *The equivalence class Q is an* ascendant *(or a* subsumer*) of the equivalence class P if $max[P] \subset max[Q]$. The equivalence class Q is a* direct ascendant *(or a* direct subsumer*) of the equivalence class P if Q is an ascendant of P and there exists no equivalence class T such that $max[P] \subset max[T] \subset max[Q]$. The subsumption relation on equivalence classes is transitive.*

For example, let us consider the equivalence class with closure {C} in Figure 3.4. This class is *directly* subsumed by the equivalence class with closure {BCE}. This class is subsumed by the equivalence class with closure {ABCE} too. However, between this class and the equivalence class with closure {ABE} there is no subsumption relation.

How to use pattern counting inference? Thanks to the levelwise traversal of frequent itemsets, first the smallest elements of an equivalence class are discovered, and these are exactly the key generators. Later when finding a larger itemset, it is tested if it belongs to an already discovered equivalence class. If it does, the database does not have to be accessed to determine its support, since it is equal by definition to the support of the already found generator in the equivalence class (see Theorem 3.2).

Note that a class can have more than one generator, and the length of generators can be different. For instance in the database D'={ABC, ABC, B, C}, the frequent closed itemset {ABC} has two generators: {A} and {BC}.

Figure 3.4 shows the equivalence classes of database D (Table 3.1) with minimum support 2 (40%). In a class only the maximal (frequent closed itemset) and minimal elements (frequent generators) are indicated. Support values are shown in the top right-hand corner of classes. The empty set is not indicated since its closure is itself in the example, thus it is not interesting from the point of view of association rule generation.

The first part of the algorithm that enumerates all frequent itemsets can be summarized as follows: it works like *Apriori*, but counts only those supports that cannot be derived from previously computed steps. This way the expensive database passes and support counts can be reduced to the frequent generators only. From some level on, all frequent generators are found, thus all remaining frequent itemsets and their supports can be inferred without any database pass. In the worst case (when all frequent itemsets are also generators) the algorithm works exactly like *Apriori*.

Identifying closed itemsets among frequent itemsets. The second part of *Zart* consists in the identification of frequent closed itemsets among frequent itemsets, adapting this idea from *Apriori-Close*. By definition, a closed itemset has no proper superset with the same support. At each i^{th} step all i-long itemsets are marked "closed". At the $(i+1)^{th}$ iteration for each $(i+1)$-long itemset we test if it has an i-long subset with the same support. If so, then the i-long itemset is not a closed itemset since it has a proper superset with the same support and it is marked "not closed". When the algorithm terminates with the enumeration of all frequent itemsets, itemsets still marked "closed" are the frequent closed itemsets of the dataset. This way we manage to identify the maximal elements of equivalence classes.

Associating the generators to their closures. During the previous two steps we have found the frequent itemsets, marked frequent generators, and filtered the frequent closed itemsets that are the maximal elements of equivalence classes. What remains is to find the links between the generators and closed itemsets, i.e. to find the equivalence classes.

Because of the levelwise itemset search, when a frequent closed itemset is found, all its frequent subsets are already known. This means that its generators are already computed, they only have to be identified. We have already seen that a generator is a minimal subset (w.r.t. set inclusion) of its closed itemset with the same support. Consider first the following straightforward approach to associate generators: given a frequent closed i-long itemset z, find all its subsets (length from 1 to $(i-1)$) having the same support as z, and store them in a list. This results in all the elements of an equivalence class, not only its generators. If the list is empty then it means that the closed itemset only has one generator, itself. We can find the generators in the list as follows: for each itemset delete all its proper supersets in the list. What remains are the generators. However, this approach is very slow and inefficient, since it looks for the subsets of a closed itemset in a redundantly large space. We show that the search space for generators can be narrowed to "not closed" key itemsets. At step i the previously found frequent itemsets do not have to be kept in memory. After registering the not closed key itemsets in a list, the frequent and frequent closed itemsets can be written to the file system and deleted from the memory. This way at each iteration a great amount of memory can be reused, and thus the algorithm can work on especially large datasets. Furthermore, we show that it is not needed to store the support of not closed key itemsets, thus the space requirement of the algorithm is further reduced. This is justified by the following properties:

Property 3.6 *A closed itemset cannot be a generator of a larger itemset.*

Property 3.7 *The closure of a frequent not closed generator* g *is the smallest proper superset of* g *in the set of frequent closed itemsets.*

By using these two properties, the algorithm for efficiently finding generators is the following: key itemsets are stored in a list l. At the i^{th} iteration frequent closed i-itemsets are filtered. For each frequent closed i-itemset z the following steps are executed: find the subsets of z in list l, register them as generators of z, and delete them from l. Before passing to the $(i+1)^{th}$ iteration, add the i-long not closed key itemsets to list l. Properties 3.6 and 3.7 *guarantee* that whenever the subsets of a frequent closed itemset are looked for in list l, only its generators are returned. The returned subsets have the same support as the frequent closed itemset, it does not even have to be tested! Since only the generators are stored in the list, it means that we need to test much less elements than the whole set of frequent itemsets. When all frequent itemsets are found, the list l is empty. This method has another feature: since at step i in list l the length of the largest element can be maximum $(i-1)$, we do not find the generators that are identical to their closures. It must be added when equivalence classes are processed. Whenever a frequent closed itemset is read that has no generator registered, it simply means that its generator is itself.

As for the implementation, instead of using a "normal" list for storing generators, the trie data structure is suggested (see Appendix C.1), since it allows a very quick lookup of stored subsets of a set (see Appendix C.2).

The Algorithm

In this section, we present the *Zart* algorithm.

Pseudo code. The main block of the algorithm is given in Algorithm 1. *Zart* uses three different kinds of tables, their description is provided in Tables 3.2 and 3.3. We assume that

C_i	potentially frequent candidate i-itemsets fields: **(1)** itemset, **(2)** pred_supp, **(3)** key, **(4)** support
F_i	frequent i-itemsets fields: **(1)** itemset, **(2)** key, **(3)** support, **(4)** closed
Z_i	frequent closed i-itemsets fields: **(1)** itemset, **(2)** support, **(3)** gen

Table 3.2: Tables used in *Zart*.

itemset	–	an arbitrary itemset
pred_supp	–	the minimum of the supports of all $(i-1)$-long frequent subsets of the itemset
key	–	is the itemset a key generator?
closed	–	is the itemset a closed itemset?
gen	–	generators of a closed itemset

Table 3.3: Fields of the tables of *Zart*.

an itemset is an ordered list of attributes, since we will rely on this in the `Zart-Gen` function (Algorithm 2).[16]

`SupportCount` procedure: this method gets a C_i table with potentially frequent candidate itemsets, and it fills the *support* field of the table. This step requires one database pass. For a detailed description see Algorithm 21.

`Subsets` function: this method gets a set of itemsets S, and an arbitrary itemset l. The function returns such elements of S that are subsets of l. For a detailed description see Algorithm 22.

Note that the empty set is only interesting, from the point of view of rule generation, if its closure is not itself. By definition, the empty set is always a generator and its support is 100%, i.e. it is present in each object of a dataset ($supp(\emptyset) = |O|$). As a consequence, it is the generator of an itemset whose support is 100%, i.e. of an itemset that is present in each object. In a binary table it means a rectangle that fills one or more columns completely. In this case, the empty set is registered as a frequent generator (line 15 of Algorithm 1), and attributes that fill full columns are marked as "not keys" (line 10 of Algorithm 1). Since in our database D (Table 3.1) there is no full column, the empty set is not registered as a frequent generator, and not shown in Figure 3.4 either.

Optimizing the support count of 2-itemsets. It is well known that many itemsets of length 2 turn out to be infrequent. Counting the support of 2-itemsets can be done more efficiently the following way. Through a database pass, an upper-triangular 2D matrix can be built containing the support values of 2-itemsets. This technique is especially useful for vertical algorithms where the number of intersection operations can thus be significantly reduced, but this optimization can also be applied to levelwise algorithms. Consult Appendix E for a detailed description and an example. Note that for a fair comparison with other algorithms, we disabled this option in the experiments.

[16]Note that we have this assumption for all levelwise algorithms presented in the thesis.

Algorithm 1 (Zart):

```
$fullColumn \leftarrow$ false;
$FG \leftarrow \{\}$; // global list of frequent generators
filling $C_1$ with 1-itemsets; // copy attributes to $C_1$
SupportCount($C_1$);
$F_1 \leftarrow \{c \in C_1 \mid c.\text{support} \geq min\_supp\}$;
loop over the rows of $F_1$ ($l$)
{
  $l$.closed $\leftarrow$ true;
  if ($l$.supp $= |O|$) {
    $l$.key $\leftarrow$ false; // the empty set is its generator
    $fullColumn \leftarrow$ true;
  }
  else $l$.key $\leftarrow$ true;
}
if ($fullColumn$ = true) $FG \leftarrow \{\emptyset\}$;
for ($i \leftarrow 1$; true; $++i$)
{
  $C_{i+1} \leftarrow$ Zart-Gen($F_i$);
  if ($C_{i+1} = \emptyset$) break; // exit from loop
  if $C_{i+1}$ has a row whose "key" value is true, then
  {
    loop over the elements of the database ($o$) {
      $S \leftarrow$ Subsets($C_{i+1}$, $o$);
      loop over the elements of $S$ ($s$):
        if ($s$.key = true) $++s$.support;
    }
  }
  loop over the rows of $C_{i+1}$ ($c$)
  {
    if ($c$.support $\geq min\_supp$) {
      if (($c$.key = true) and ($c$.support = $c$.pred_supp)):
        $c$.key $\leftarrow$ false;
      $F_{i+1} \leftarrow F_{i+1} \cup \{c\}$;
    }
  }
  loop over the rows of $F_{i+1}$ ($l$) {
    $l$.closed $\leftarrow$ true;
    $S \leftarrow$ Subsets($F_i$, $l$);
    loop over the elements of $S$ ($s$):
      if ($s$.support = $l$.support) $s$.closed $\leftarrow$ false;
  }
  $Z_i \leftarrow \{l \in F_i \mid l.\text{closed} = \text{true}\}$;
  Find-Generators($Z_i$);
}
$Z_i \leftarrow F_i$;
Find-Generators($Z_i$);

Result:
  FIs: $\bigcup_i F_i$
  FCIs + their generators: $\bigcup_i Z_i$
```

Algorithm 2 (Zart-Gen function):

Input: F_i – set of frequent itemsets
Output: table C_{i+1} with potentially frequent candidate itemsets.
Plus: *key* and *pred_supp* fields will be filled in C_{i+1}.

```
 1)  insert into $C_{i+1}$
     select $p[1], p[2], \ldots, p[i], q[i]$
     from $F_i$ $p$, $F_i$ $q$
     where $p[1] = q[1], \ldots, p[i-1] = q[i-1], p[i] < q[i]$; // like in Apriori
 2)  loop over the rows of $C_{i+1}$ ($c$)
 3)  {
 4)    $c$.key $\leftarrow$ true;
 5)    $c$.pred_supp = $|O| + 1$; // number of objects in the database + 1 (imitating $+\infty$)
 6)    $S \leftarrow (i-1)$-long subsets of $c$;
 7)    loop over the elements of $S$ ($s$)
 8)    {
 9)      if ($s \notin F_i$) then $C_{i+1} \leftarrow C_{i+1} \setminus \{c\}$; // remove it if it is rare
10)      else {
11)        $c$.pred_supp $\leftarrow$ min($c$.pred_supp, $s$.support);
12)        if ($s$.key = false) then $c$.key $\leftarrow$ false; // by Property 3.5
13)      }
14)    }
15)    if ($c$.key = false) then $c$.support $\leftarrow$ $c$.pred_supp; // by Theorem 3.2
16)  }
17)  return $C_{i+1}$;
```

Algorithm 3 (Find-Generators procedure):

Method: fills the *gen* field of the table Z_i with generators
Input: Z_i – set of frequent closed itemsets

```
1)  loop over the rows of $Z_i$ ($z$)
2)  {
3)    $S \leftarrow$ Subsets($FG, z$);
4)    $z$.gen $\leftarrow S$;
5)    $FG \leftarrow FG \setminus S$;
6)  }
7)  $FG \leftarrow FG \cup \{l \in F_i \mid l.\text{key} = \text{true} \wedge l.\text{closed} = \text{false}\}$;
```

Running example. The execution of *Zart* on dataset D (Table 3.1) with $min\_supp = 2$ (40%) is illustrated in Table 3.4.

DB scan$_1$ $\rightarrow$

C_1	pred_supp	key	supp
{A}			4
{B}			4
{C}			4
{D}			1
{E}			4

F_1	key	supp	closed
{A}	yes	4	yes
{B}	yes	4	~~yes~~
{C}	yes	4	yes
{E}	yes	4	~~yes~~

Z_1	supp	gen
{A}	4	
{C}	4	

$FG_{before} = \{\}$
$FG_{after} = \{B, E\}$

DB scan$_2$ $\rightarrow$

C_2	pred_supp	key	supp
{AB}	4	yes	3
{AC}	4	yes	3
{AE}	4	yes	3
{BC}	4	yes	3
{BE}	4	~~yes~~	4
{CE}	4	yes	3

F_2	key	supp	closed
{AB}	yes	3	~~yes~~
{AC}	yes	3	yes
{AE}	yes	3	~~yes~~
{BC}	yes	3	~~yes~~
{BE}	no	4	yes
{CE}	yes	3	~~yes~~

Z_2	supp	gen
{AC}	3	
{BE}	4	{B, E}

$FG_{before} = \{B, E\}$
$FG_{after} = \{AB, AE, BC, CE\}$

DB scan$_3$ $\rightarrow$

C_3	pred_supp	key	supp
{ABC}	3	yes	2
{ABE}	3	~~yes~~	3
{ACE}	3	yes	2
{BCE}	3	~~yes~~	3

F_3	key	supp	closed
{ABC}	yes	2	~~yes~~
{ABE}	no	3	yes
{ACE}	yes	2	~~yes~~
{BCE}	no	3	yes

Z_3	supp	gen
{ABE}	3	{AB, AE}
{BCE}	3	{BC, CE}

$FG_{before} = \{AB, AE, BC, CE\}$
$FG_{after} = \{ABC, ACE\}$

C_4	pred_supp	key	supp
{ABCE}	2	~~yes~~	2

F_4	key	supp	closed
{ABCE}	no	2	yes

Z_4	supp	gen
{ABCE}	2	{ABC, ACE}

$FG_{before} = \{ABC, ACE\}$
$FG_{after} = \{\}$

C_5	pred_supp	key	supp
$\emptyset$			

Table 3.4: Execution of *Zart* on dataset D with $min\_supp = 2$ (40%).

The algorithm first performs one database scan to count the supports of 1-itemsets. The candidate itemset {D} is pruned because it is infrequent. At the next iteration, all candidate 2-itemsets are created and stored in C_2. Then a database scan is performed to determine the supports of the six potentially frequent candidate itemsets. In C_2 there is one itemset that has the same support as one of its subsets, thus {BE} is not a key generator (see Theorems 3.1 and 3.2). Using F_2, the itemsets {B} and {E} in F_1 are not closed because they have a proper superset in F_2 with the same support. The remaining closed itemsets {A} and {C} are copied to Z_1 and their generators are determined. In the global list of frequent generators (*FG*), which is still empty, they have no subsets, which means that both {A} and {C} are generators themselves. The not closed key itemsets of F_1 ({B} and {E}) are added to *FG* .

In C_3 there are two itemsets, {ABE} and {BCE}, that have a non-key subset ({BE}), thus by Property 3.5 they are not key generators either. By Theorem 3.2, their support values are equal to 3, i.e. their supports can be determined without any database access. By F_3, the itemsets {AB}, {AE}, {BC} and {CE} turn out to be "not closed". The remaining closed itemsets {AC} and {BE} are copied to Z_2. The generator of {AC} is itself, and the generators of {BE} are {B} and {E}. These two generators are deleted from *FG* and {AB}, {AE}, {BC} and {CE} are added to *FG*.

At the fourth iteration, it turns out in `Zart-Gen` that the newly generated candidate itemset contains at least one non-key subset. By Property 3.5 the new candidate itemset is not a candidate key generator, and its support is determined directly in `Zart-Gen` by Theorem 3.2.

All frequent itemsets ($\bigcup_i F_i$)		All frequent closed itemsets with their generators ($\bigcup_i Z_i$)
{a} (4) +	{b, e} (4) +	{a} (4); [{a}]
{b} (4)	{c, e} (3)	{c} (4); [{c}]
{c} (4) +	{a, b, c} (2)	{a, c} (3); [{a, c}]
{e} (4)	{a, b, e} (3) +	{b, e} (4); [{b}, {e}]
{a, b} (3)	{a, c, e} (2)	{a, b, e} (3); [{a, b}, {a, e}]
{a, c} (3) +	{b, c, e} (3) +	{b, c, e} (3); [{b, c}, {c, e}]
{a, e} (3)	{a, b, c, e} (2) +	{a, b, c, e} (2); [{a, b, c}, {a, c, e}]
{b, c} (3)		

Table 3.5: Output of *Zart*.

As there are no more candidate generators in C_4, from this step on no more database scan is needed.

In the fifth iteration no new candidate itemset is found and the algorithm breaks out from the main loop. The largest frequent closed itemset is {ABCE}, its generators are read from *FG*. When the algorithm stops, all frequent and all frequent closed itemsets with their generators are determined, as shown in Table 3.5. In the table the "+" sign means that the frequent itemset is closed. The support values are indicated in parenthesis. If *Zart* leaves the generators of a closed itemset empty, it means that the generator is identical to the closed itemset (as this is the case for {A}, {C} and {AC} in the example). Due to the property of equivalence classes, the support of a generator is equal to the support of its closure.

Apriori-Close. Here we provide the pseudo code of *Apriori-Close* (see Appendix B.2). This algorithm is an extension of *Apriori*. When it finds i-long frequent itemsets, it filters closed itemsets among $(i-1)$-long frequent itemsets. The algorithm relies on the property that a closed itemset has no proper superset with the same support. The pseudo code of the extension of *Apriori-Close* over *Apriori* can be found between lines 36 and 41 in Algorithm 1.

Pascal$^+$. Actually, *Zart* can be specified to another algorithm that we call *Pascal$^+$*. Previously we have seen that *Zart* has three main characteristics. Removing the 3rd part of *Zart* (associating generators to their closures), we get *Pascal$^+$* that can filter FCIs among FIs, just like *Apriori-Close*. To obtain *Pascal$^+$*the `Find-Generators()` procedure calls must be deleted from Algorithm 1 in lines 43 and 46.

Generalizing Zart

The idea presented in *Zart* can be generalized and thus it can be applied to *any* frequent itemset mining algorithm, be it either breadth-first or depth-first. We show it in details in Section 3.3.2, in the algorithm called *Eclat-Z*. *Eclat-Z* is a vertical algorithm that finds all frequent itemsets. Since it is a depth-first algorithm, it does not produce frequent itemsets in ascending order by their length. We will show how to use the ideas of *Zart* even in such a case, and how to extend an arbitrary frequent itemset mining algorithm to produce such results that can be used directly to generate not only all valid association rules, but minimal non-redundant association rules too.

	# Objects	# Attributes	Avg. length	Largest attr.
T20I6D100K	100,000	893	20	1,000
C20D10K	10,000	192	20	385
MUSHROOMS	8,416	119	23	128

Table 3.6: Characteristics of databases used for *Zart*.

Experimental Results

We evaluated *Zart* against *Apriori* and *Pascal*. Here we recall the characteristics of our test environment from Appendix A. All times reported are real, wall clock times as obtained from the Unix *time* command between input and output. Table 3.6 shows the characteristics of the databases used in our evaluation. It shows the number of objects, the number of different attributes, the average transaction length, and the largest attribute in each database.

The T20I6D100K[17] is a sparse dataset, constructed according to the properties of market basket data that are typically weakly correlated data. The number of frequent itemsets is small, and nearly all FIs are closed. The C20D10K is a census dataset from the PUMS sample file, while the MUSHROOMS[18] describes mushrooms characteristics. The last two are highly correlated datasets. It has been shown that weakly correlated data, such as synthetic data, constitute easy cases for the algorithms that extract frequent itemsets, since few itemsets are frequent. For such data, all algorithms give similar response times. On the contrary, dense and highly-correlated data constitute far more difficult cases for the extraction due to large differences between the number of frequent and frequent closed itemsets. Such data represent a huge part of real-life datasets.

Weakly correlated data. The T20I6D100K synthetic dataset mimics market basket data that are typically sparse, weakly correlated data. In this dataset, the number of frequent itemsets is small and nearly all frequent itemsets are generators. *Apriori*, *Pascal* and *Zart* behave identically. Response times for the T20I6D100K dataset are presented numerically in Table 3.7 and graphically in Figure 3.5.

Table 3.7 also contains some statistics about the datasets, provided by *Zart*. It shows the number of FIs, the number of FCIs, the number of frequent generators, the proportion of the number of FCIs to the number of FIs, and the proportion of the number of frequent generators to the number of FIs, respectively. As we can see in T20I6D100K, above 0.75% minimum support all frequent itemsets are closed and generators at the same time. It means that each equivalence class has one element only. Because of this, *Zart* and *Pascal* cannot use the advantage of pattern counting inference, and they work exactly like *Apriori*.

Strongly correlated data. Response times obtained for the C20D10K and MUSHROOMS datasets are given numerically in Table 3.7, and graphically in Figures 3.6 and 3.7, respectively. In these two datasets, the number of frequent generators is much less than the total number of frequent itemsets. Hence, using pattern counting inference, *Zart* has to perform much fewer support counts than *Apriori*. We can observe that in all cases the execution times of *Zart* and *Pascal* are almost identical: adding the frequent closed itemset derivation *and* the identification of their generators to the frequent itemset discovery does not induce serious additional computation

[17]`http://www.almaden.ibm.com/software/quest/Resources/`
[18]`http://kdd.ics.uci.edu/`

min_supp (%)		Apriori	Pascal	Zart		# FIs	# FCIs	# FGs	$\frac{\#FCIs}{\#FIs}$	$\frac{\#FGs}{\#FIs}$
T20I6D100K										
2		72.67	71.15	71.13		378	378	378	100.00%	100.00%
1		107.63	106.24	107.69		1,534	1,534	1,534	100.00%	100.00%
0.75		134.49	132.00	133.00		4,710	4,710	4,710	100.00%	100.00%
0.5		236.10	228.37	230.17		26,836	26,208	26,305	97.66%	98.02%
0.25		581.11	562.47	577.69		155,163	149,217	149,447	96.17%	96.32%
C20D10K										
50		61.18	16.68	17.94		1,823	456	456	25.01%	25.01%
40		71.60	19.10	19.22		2,175	544	544	25.01%	25.01%
30		123.57	26.74	26.88		5,319	951	967	17.88%	18.18%
20		334.87	53.28	54.13		20,239	2,519	2,671	12.45%	13.20%
10		844.44	110.78	118.09		89,883	8,777	9,331	9.76%	10.38%
MUSHROOMS										
60		3.10	2.04	2.05		51	19	21	37.25%	41.18%
50		6.03	3.13	3.13		163	45	53	27.61%	32.52%
40		13.93	6.00	5.94		505	124	153	24.55%	30.30%
30		46.18	12.79	12.75		2,587	425	544	16.43%	21.03%
20		554.95	30.30	34.88		53,337	1,169	1,704	2.19%	3.19%

Table 3.7: Response times of *Zart* and other statistics.

time. *Apriori* is very efficient on sparse datasets, but on strongly correlated data the other algorithms perform much better.

Comparing Pascal$^+$ and Pascal. We also compared the efficiency of *Pascal*$^+$ with *Pascal.* We got the same results like with *Apriori-Close* and *Apriori* in Appendix B.2. *Pascal*$^+$ gives almost equivalent response times to *Pascal* on both weakly and strongly correlated data, i.e. the filtering of closed itemsets among frequent itemsets is not an expensive step. Since the results are almost identical (w.r.t. differences of response times) with the results shown in Table B.5 and Figures B.1, B.2 and B.3, we do not include experimental results of *Pascal*$^+$ and *Pascal* here. As *Pascal* is more efficient than *Apriori* on strongly correlated data (see Table 3.7), *Pascal*$^+$ is necessarily more efficient than *Apriori-Close.* If we need both frequent and frequent closed itemsets, then *Pascal*$^+$ is recommended instead of *Apriori-Close.*

Conclusion

In this subsection we presented a new, multifunctional itemset mining algorithm called *Zart*, which is a refinement of *Pascal.* With pattern counting inference, using the generators of equivalence classes, it can reduce the number of itemsets counted and the number of database passes. In addition, it can identify frequent closed itemsets among frequent itemsets, and it can associate generators to their closure. We showed that these extra features are required for the generation of minimal non-redundant association rules. *Zart* can also be specified to another algorithm that we call *Pascal*$^+$. *Pascal*$^+$ finds both frequent and frequent closed itemsets, like *Apriori-Close.*

We compared the performance of *Zart* with *Apriori* and *Pascal.* The results showed that *Zart* gives almost equivalent response times to *Pascal* on both weakly and strongly correlated data, though *Zart* also identifies closed itemsets and their generators. Since *Zart* finds all frequent itemsets too, if needed, it can also be used to extract all valid association rules.

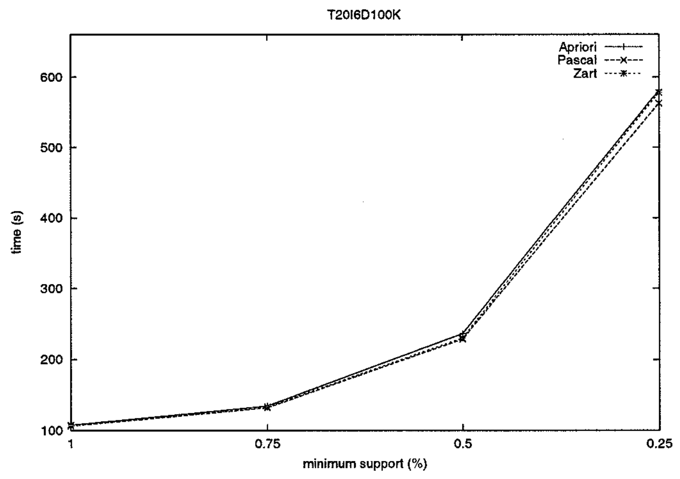


Figure 3.5: Response times of *Zart* for T20I6D100K.

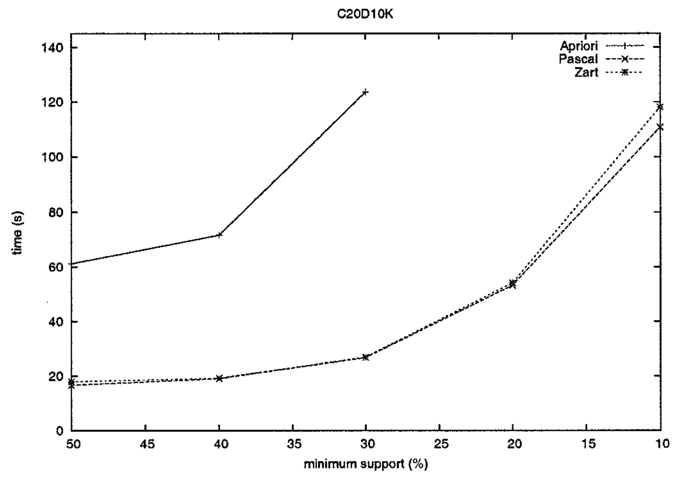


Figure 3.6: Response times of *Zart* for C20D10K.

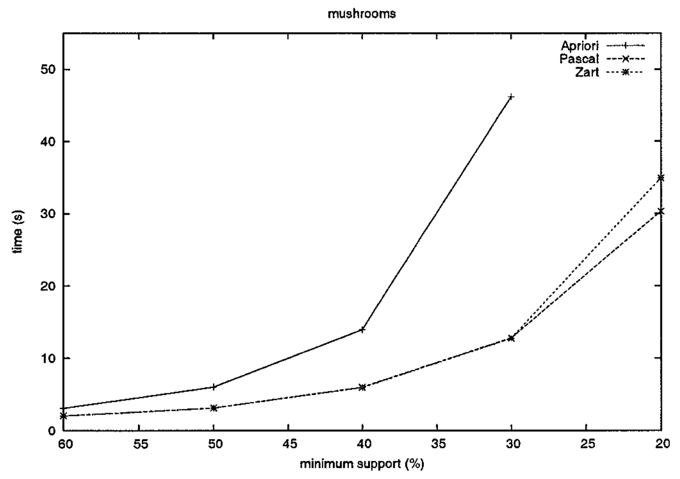


Figure 3.7: Response times of *Zart* for MUSHROOMS.

3.3.2 Eclat-Z

Eclat-Z is a hybrid algorithm that produces the same result as *Zart*, i.e. its output can also be used directly to generate minimal non-redundant association rules ($\mathcal{MNR}$). *Eclat-Z* combines two algorithms: *Eclat* and *Zart*. *Eclat* (Appendix B.3) is a vertical, depth-first algorithm that finds all frequent itemsets in a very efficient way. *Zart* (Section 3.3.1) is a levelwise algorithm that filters frequent closed itemsets among frequent itemsets and associates generators to their closures. In *Eclat-Z* we present an idea how to extend an arbitrary frequent itemset mining algorithm in order to support the generation of minimal non-redundant association rules too.

Motivation and Contribution

Our motivation is the same as in the case of *Zart*: we want to find minimal non-redundant association rules. Two things are needed for generating these rules: frequent closed itemsets *and* their associated generators. *Zart* is a levelwise algorithm, based on *Pascal*. Thanks to its pattern counting inference mechanism, it can greatly reduce the number of database passes. *Pascal* finds all FIs and marks frequent generators. *Zart*, in addition, filters FCIs and associates the generators to their closures. Among levelwise FI-miner algorithms, *Pascal* may be the most efficient. Experimental results show that *Zart* almost gives equivalent response times to *Pascal*, i.e. its extra features do not induce any serious additional computation time.

We posed ourselves the following question: can the idea of *Zart* be generalized and used with *any* arbitrary frequent itemset mining algorithm, be it either breadth-first or depth-first? Could we somehow extend these algorithms in a universal way to produce such results that can be used directly to generate not only all valid association rules, but minimal non-redundant association rules too? Our answer is positive, and this is the subject of this section.

The idea of *Zart* can be generalized the following way. Find all frequent itemsets and organize them in ascending order by their length. Then process FIs in a levelwise manner. First, filter frequent generators, and then continue like *Zart*: filter frequent closed itemsets too and associate generators to their closures. In *Zart*, we used *Pascal* for finding FIs, but this part of *Zart* can be replaced with another FI-miner algorithm, whose output can be post-processed in a levelwise manner. We chose Zaki's *Eclat* algorithm as the new "engine" of *Zart*. As we saw in Appendix B.3, *Eclat* is more efficient than *Pascal* on both sparse and dense datasets. However, *Pascal* has two advantages. First, due to its levelwise nature, it finds FIs in ascending order by their length. Secondly, it tells which itemsets are generators. This provides a very good basis for *Zart* that adds two more things: filtering FCIs and associating generators to their closures. On the contrary, *Eclat* provides FIs in a completely unordered way due to its depth-first nature. However, the levelwise post-processing of FIs requires itemsets in ascending order by their length. There are two possibilities. First, we could modify *Eclat* to produce itemsets in an ordered way. It means a breadth-first traversal of the IT-tree. Unfortunately, in practice it is impossible, because it would keep too many elements in the main memory at the same time. The only way is to produce itemsets in a depth-first manner, write these itemsets in a file, and then process the itemsets in the file in ascending order by length. Obviously, because of the high cost of I/O operations, we need an efficient indexing technique for this file. We have made two versions of *Eclat-Z*. In the first version, we keep an index structure in main memory. For each itemset length we build a list that only stores file positions that indicate where an itemset is written in the file. For instance, if there are twenty 2-itemsets, then we build a list of twenty file positions. When reading back itemsets of a given length, simply the appropriate list must be traversed, seek to the given positions, and read itemsets from the file. Instead of a whole itemset and its support value, we only keep one integer in main memory for each itemset. However, this version has a drawback: when finding too many itemsets, the index structure slowly fills up the memory. In a revised version of *Eclat-Z*, we managed to eliminate this problem by building the index structure *in the file* itself. In the memory we only keep a simple index array, whose memory requirement is insignificant.

Our contribution can be summarized as follows. We present a universal way to extend any FI-miner algorithm in order to produce minimal non-redundant association rules too. We present an efficient method how to store FIs in a file if FIs are not provided in ascending order by length. Thanks to our special indexing technique, FIs can be read in ascending order by length. Our indexing requires no additional memory. Once itemsets are available in ascending order by length, we show how to filter generators, closed itemsets, and how to associate generators to their closures.

Detailed Description of Eclat-Z

Eclat-Z combines ideas of *Eclat* and *Zart*. *Eclat* finds all FIs that we save in the file system. Then, this file is processed in a levelwise manner, i.e. itemsets are read in ascending order by length, generators and closed itemsets are filtered, and finally generators are associated to their closures. In the following two subsections we present the algorithm in detail.

Processing Itemsets in Ascending Order by Length

This step is required for such algorithms that do not generate FIs in ascending order by length. *Eclat* is a good example for such an algorithm that we used as our "engine" in *Eclat-Z*. Levelwise algorithms, like *Apriori*, represent an easier case, because they produce FIs in a good order. If someone wants to use such an algorithm, he can continue with the second part of the algorithm. Here, in the first part, we present an efficient, file-system based approach to process FIs in ascending order by their length.

For our example, we use dataset D (Table 3.1) with $min\_supp = 2$ (40%). *Eclat* produces FIs of D in an unordered way, as shown in Table 3.8.

order	itemset	support
1)	ABCE	2
2)	ABC	2
3)	ABE	3
4)	AB	3
5)	ACE	2
6)	AC	3
7)	AE	3
8)	A	4
9)	BCE	3
10)	BC	3
11)	BE	4
12)	B	4
13)	CE	3
14)	C	4
15)	E	4

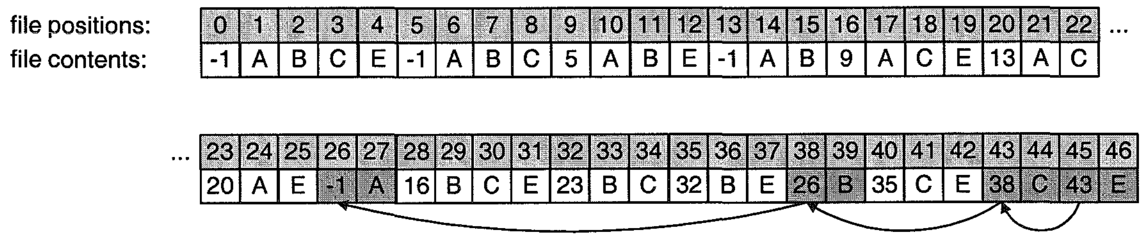
Table 3.8: Order of frequent itemsets produced by *Eclat*.

As it is impossible to keep all FIs in the main memory, we write FIs in a binary file. For this, we need to modify the save method of *Eclat*, as follows. In the main memory we have an index, called *PosIndex*, for file positions (Figure 3.8). *PosIndex* is a simple array of integers. At position i it indicates where the last i-long itemset is written in the binary file. *PosIndex* must always be kept up-to-date. On the left part of Figure 3.8, it is indicated how *PosIndex* changes in time between t_0 and t_{15}. The right side of the same figure shows the final state of *PosIndex*. Figure 3.9 shows the contents of the file (file positions are also indicated).

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
0	-1															
1	-1								26				38		43	45
2	-1				13		20	23			32	35		40		
3	-1		5	9		16				28						
4	-1	0														

0	-1
1	45
2	40
3	28
4	0

Figure 3.8: The PosIndex structure of *Eclat-Z*. Timeline (left) and final state (right).



file positions:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
file contents:	-1	A	B	C	E	-1	A	B	C	5	A	B	E	-1	A	B	9	A	C	E	13	A	C

...	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
	20	A	E	-1	A	16	B	C	E	23	B	C	32	B	E	26	B	35	C	E	38	C	43	E

Figure 3.9: Contents of the file with the FIs. File positions are also indicated.

Running example. The first frequent itemset found by *Eclat* is $ABCE$ (see Table 3.8). It is a 4-itemset. The size of the *PosIndex* array is dynamically increased to size $4 + 1$ ($+1$, because position 0 is not used). The array is initialized: at each of its position we store -1 (t_0). As the length of the found itemset is 4, we read the value of *PosIndex* at position 4. This value (-1), together with the itemset is written to the binary file (see Figure 3.9). The value that we read from *PosIndex* is a *backward pointer* that shows the file position of the previous itemset *with the same length.* As the value is -1 here, it simply means that this is the first itemset of this length. After writing $ABCE$ to the file, the 4^{th} position of *PosIndex* is updated to 0 (t_1), because the last 4-long itemset was written to position 0 in the file. ABC is written similarly, and *PosIndex* is updated (t_2). When ABE is written to the file, its backward pointer is set to 5. This value is read from *PosIndex* at position 3, since ABE is a 3-itemset. The process continues until all FIs are found. The final state of *PosIndex* is indicated on the right side of Figure 3.8.

Reading itemsets of a given length. Figure 3.9 demonstrates how to read back 1-itemsets from the file (shown in darker grey). First, we need to know where the last 1-itemset is written.

It is registered in *PosIndex* (Figure 3.8), at position 1. The value here shows that the last 1-itemset is at position 45 in the file. Itemset E is read, and we seek to the previous 1-itemset at position 43. C is read, seek to position 38. B is read, seek to position 26. A is read, and -1 indicates that there are no more 1-itemsets. This way FIs can be processed in ascending order by length.

Some implementation details. For an easier understanding, we presented our indexed file structure in a slightly simplified way. Actually, after each itemset we also have to save the appropriate support values. In the schema, we used simple byte values. All of our algorithms are implemented in Java, where an integer (type int) requires 4 bytes. We save all values as integers (int), thus to get the real file positions, all values in the schema must be multiplied by 4.

Finding Generators, Closures, and Associating them

In the previous subsection, we presented the 1st part of the algorithm, i.e. how to get frequent itemsets in ascending order by their length, even if they are produced in an unordered way. In this subsection we continue with the 2nd part, namely how to associate generators and their closures, once FIs are available in a good order.

The main block is shown in Algorithm 4. Two kinds of tables are used, namely F_i for frequent, and Z_i for frequent closed itemsets. They are equivalent to *Zart*'s tables (see Tables 3.2 and 3.3). Here we suppose that FIs are available in ascending order by their length.

`readTable` function (Algorithm 5): this method is responsible for reading frequent itemsets of a given length. If *Eclat* is used as the FI-miner part of the algorithm, then `readTable` reads FIs from the binary file, as explained previously. The function returns FIs in an F_i table. Fields of the table are initialized: itemsets are marked as "keys" and "closed". Of course, during the post-processing step these values may change. Frequent attributes (frequent 1-itemsets) represent a special case. If they are present in each object of the dataset, then they are not key generators, because they have a smaller subset with the same support, namely the empty set. In this case the empty set is a useful generator (w.r.t. rule generation).

`findKeysAndClosedItemsets` procedure (Algorithm 6): this method is responsible for filtering FCIs and FGs among FIs. The filtering procedure is based on the following definitions. A frequent closed itemset has no proper superset with the same support. Furthermore, a frequent generator has no proper subset with the same support.

`Subsets` function: this method is identical to Algorithm 22.

`Find-Generators` procedure: this method, which is responsible for associating FCIs and FGs, is identical to Algorithm 3.

Running example. The execution of *Eclat-Z* on dataset D (Table 3.1) with $min\_supp = 2$ (40%) is illustrated in Table 3.9. Since this 2nd part of the algorithm is based on *Zart*, we do not repeat the explanation of the example here.

```
Algorithm 4 (Eclat-Z):

1)  $maxItemsetLength \leftarrow$ (length of the largest FI previously found by the FI-miner);
2)  $fullColumn \leftarrow$ false;
3)  $FG \leftarrow \{\}$; // global list of frequent generators
4)  $F_1 \leftarrow$ readTable(1); // get frequent 1-itemsets
5)  for ($i \leftarrow 1$; $i < maxItemsetLength$; $++i$)
6)  {
7)    $F_{i+1} \leftarrow$ readTable($i + 1$); // get frequent (i+1)-itemsets
8)    findKeysAndClosedItemsets($F_{i+1}$, $F_i$);
9)    $Z_i \leftarrow \{l \in F_i \mid l$.closed = true$\}$;
10)   Find-Generators($Z_i$);
11) }
12) $Z_i \leftarrow \{l \in F_i \mid l$.closed = true$\}$;
13) Find-Generators($Z_i$);
```

```
Algorithm 5 (readTable function):

Input:   $i$ – length of frequent itemsets that are to be read
Method:  reads $i$-long itemsets and returns them in an $F$ table.
         It also initializes other fields of the table.

1)  $F \leftarrow \{\}$; // $F$ is an empty table
2)  loop over $i$-long itemsets ($curr$)
3)  {
4)    $row$.itemset $\leftarrow$ $curr$.itemset;
5)    $row$.supp $\leftarrow$ $curr$.supp;
6)    $row$.closed $\leftarrow$ true;
7)    if ($i = 1$)
8)    {
9)      if ($row$.supp = $|O|$) {
10)       $row$.key $\leftarrow$ false; // the empty set is its generator
11)       $fullColumn \leftarrow$ true;
12)     }
13)     else $row$.key $\leftarrow$ true;
14)   }
15)   else $row$.key $\leftarrow$ true;
16)   $F \leftarrow F \cup row$;
17) }
18) if ($fullColumn$ = true) $FG \leftarrow \{\emptyset\}$; // the empty set is a useful generator
19)
20) return $F$;
```

Algorithm 6 (findKeysAndClosedItemsets procedure):

Input: F_{i+1} – set of $(i+1)$-long frequent itemsets
F_i – set of i-long frequent itemsets
Method: **(1)** using F_{i+1}, it filters FCIs in F_i, and
(2) using F_i, it filters key generators in F_{i+1}

```
1)  loop over the rows of $F_{i+1}$ ($sup$)
2)  {
3)    $S \leftarrow$ Subsets($F_i$, $sup$); // find subsets of $sup$ in $F_i$
4)    loop over the elements of $S$ ($sub$)
5)    {
6)      if ($sub$.supp = $sup$.supp)
7)      {
8)        $sub$.closed $\leftarrow$ $false$; // it has a proper superset with the same support
9)        $sup$.key $\leftarrow$ $false$; // it has a proper subset with the same support
10)     }
11)   }
12) }
```

F_1	key	supp	closed
{A}	yes	4	yes
{B}	yes	4	~~yes~~
{C}	yes	4	yes
{E}	yes	4	~~yes~~

Z_1	supp	gen
{A}	4	
{C}	4	

$FG_{before} = \{\}$
$FG_{after} = \{B, E\}$

F_2	key	supp	closed
{AB}	yes	3	~~yes~~
{AC}	yes	3	yes
{AE}	yes	3	~~yes~~
{BC}	yes	3	~~yes~~
{BE}	~~yes~~	4	yes
{CE}	yes	3	~~yes~~

Z_2	supp	gen
{AC}	3	
{BE}	4	{B, E}

$FG_{before} = \{B, E\}$
$FG_{after} = \{AB, AE, BC, CE\}$

F_3	key	supp	closed
{ABC}	yes	2	~~yes~~
{ABE}	~~yes~~	3	yes
{ACE}	yes	2	~~yes~~
{BCE}	~~yes~~	3	yes

Z_3	supp	gen
{ABE}	3	{AB, AE}
{BCE}	3	{BC, CE}

$FG_{before} = \{AB, AE, BC, CE\}$
$FG_{after} = \{ABC, ACE\}$

F_4	key	supp	closed
{ABCE}	~~yes~~	2	yes

Z_4	supp	gen
{ABCE}	2	{ABC, ACE}

$FG_{before} = \{ABC, ACE\}$
$FG_{after} = \{\}$

Table 3.9: Execution of *Eclat-Z* on dataset D with $min\_supp = 2$ (40%).

min_supp (%)		Eclat-Z	Zart
T20I6D100K			
2		7.08	71.13
1		7.87	107.69
0.75		9.21	133.00
0.5		19.40	230.17
0.25		96.18	577.69
C20D10K			
50		1.31	17.94
40		1.36	19.22
30		1.75	26.88
20		4.27	54.13
10		16.76	118.09
MUSHROOMS			
60		0.91	2.05
50		0.91	3.13
40		0.98	5.94
30		1.27	12.75
20		10.04	34.88

Table 3.10: Response times of *Eclat-Z*.

Experimental Results and Conclusion

We compared the efficiency of *Eclat-Z* with *Zart*. These two algorithms are similar in the sense that they both produce exactly the same results. The execution times of the algorithms on different datasets is illustrated in Table 3.10. As we can see, *Eclat-Z* performs much better than *Zart*, on both weakly and strongly correlated data. This difference is due to the FI-miner "engine" of the algorithms. *Zart* is based on *Pascal*, and *Eclat-Z* is based on *Eclat*. In Appendix B.3 it is shown that *Eclat* –a vertical, depth-first algorithm– performs much better than *Pascal*, which is a levelwise algorithm. The 2nd part of *Eclat-Z* and *Zart* are almost identical: they find frequent generators and frequent closed itemsets, and they associate generators to their closures. This way, the result of both algorithms can be used directly to generate minimal non-redundant association rules.

As a conclusion we can say that the idea of *Zart*, i.e. associating generators and their closures, is very useful for the generation of interesting association rules. However, *Zart* has one drawback, namely that it is based on a levelwise algorithm. Although *Pascal* is very efficient among other levelwise algorithms, its performance degrades in the case of dense datasets, or when minimum support is set very low. In *Eclat-Z* we generalized the idea of *Zart*, and we applied this idea on a very efficient FI-miner algorithm called *Eclat*. With *Eclat* we had to face another problem: it produces itemsets in an unordered way. Thanks to a special file indexing technique, we managed to solve this problem, and then we could use the idea of *Zart* in a post-processing step. Naturally, beside *Eclat* other FI-miner algorithms can also be extended this way in order to support the extraction of minimal non-redundant association rules too.

3.3.3 Charm-MFI

Short Overview of the Charm-MFI Algorithm

In this section, we present an extension of *Charm* called *Charm-MFI.* This algorithm can identify not only frequent closed itemsets (FCIs), but *maximal frequent itemsets* (MFIs) too. By definition, a maximal frequent itemset is a frequent closed itemset such that all its proper supersets are rare itemsets (and necessarily, all its subsets are frequent itemsets). Clearly, MFI $\subseteq$ FCI $\subseteq$ FI. The idea for filtering MFIs among FCIs is the following. At each i^{th} step, all i-long FCIs are marked as "maximal" (where "maximal" means maximal frequent itemset). At the next $(i+1)^{th}$ iteration for each $(i+1)$-long FCI we test if it has an i-long subset. If so, then the i-long subset is not a maximal frequent itemset because it has a frequent superset, thus we mark it as "not maximal". When the algorithm terminates with the enumeration of all frequent closed itemsets, the itemsets still marked "maximal" are the MFIs. As a consequence, the largest frequent closed itemsets are always maximal frequent itemsets. This technique is very similar to the filtering of *Apriori-Close* (*Apriori-Close* marks FCIs among FIs). As we will see it in the experimental results, this kind of filtering of maximal itemsets does not induce any serious additional computation time. In our implementation we extended *Charm*, but the idea presented here can be used with *any* frequent closed itemset mining algorithm. The only criterion is that FCIs must be processed in ascending order by their length.

MFIs are called *maximal*, because they have no frequent supersets. On the other hand, regarding the number of these itemsets, they are *minimal*, i.e. they form a minimal generator set from which all frequent itemsets can be restored.[19] The MFI representation is a condensed, but not lossless representation of FIs. Subset frequency is not available, thus from MFIs all frequent itemsets can be restored the following way. First, we need to take all possible subsets of MFIs, and then with one database pass their supports can be counted. This kind of support count is a very expensive step, thus MFIs are not really suitable for generating association rules. In our work we have only used MFIs for our border studies (see Chapter 5). In our approach, first we find FCIs then we filter MFIs, but there are several other algorithms that find MFIs directly [Bay98, AAP00, GZ01].

The Algorithm

In this section, we present the *Charm-MFI* algorithm.

Pseudo code. The main block of the algorithm is given in Algorithm 7. We assume that all frequent closed itemsets are already discovered, because FCIs provide the input of *Charm-MFI.*

`readTable` function: this method collects i-long (given as a parameter) FCIs and returns them in a table. In the table all itemsets are marked as "maximal". In our implementation, we extended *Charm.* As can be seen in Appendix B.4, *Charm* stores all FCIs in the main memory in a hash structure. This structure can be easily traversed in order to gather itemsets of a given length. As mentioned, our approach can be generalized and used with any FCI-miner algorithm. To do so, only this procedure needs to be customized.

`Subsets` function: this method is identical to Algorithm 22.

[19] Maybe they should rather be called "largest frequent itemsets", because they have no larger frequent supersets.

```
Algorithm 7 (Charm-MFI):

Description:  finds maximal frequent itemsets
Input:        frequent closed itemsets

1)  $maxItemsetLength \leftarrow$ (length of the largest FCI);
2)  $T_1 \leftarrow$ readTable(1);
3)  for ($i \leftarrow 1$; $i < maxItemsetLength$; $++i$)
4)  {
5)      $T_{i+1} \leftarrow$ readTable(i+1);
6)      findMaximalFrequentItemsets($T_{i+1}$, $T_i$);
7)  }
```

```
Algorithm 8 (findMaximalFrequentItemsets procedure):

Method:  filter maximal frequent itemsets in $T_i$ using $T_{i+1}$
Input:   $T_{i+1}$ – table of frequent closed itemsets of length $(i+1)$
         $T_i$ – table of frequent closed itemsets of length $i$

1)  loop over the rows of $T_{i+1}$ ($sup$)
2)  {
3)      $S \leftarrow$ Subsets($T_i$, $sup$); // find subsets of $sup$ in $T_i$
4)      loop over the elements of $S$ ($sub$):
5)        $sub$.maximal $\leftarrow$ false;
6)  }
```

T_1	supp	maximal
{C}	4	~~yes~~
{A}	4	~~yes~~

T_2	supp	maximal
{BE}	4	~~yes~~
{AC}	3	yes

T_3	supp	maximal
{BCE}	3	yes
{ABE}	3	yes

Table 3.11: Execution of *Charm-MFI* on dataset D with $min\_supp = 3$ (60%).

Running example. The execution of *Charm-MFI* on dataset D (Table 3.1) with min_supp = 3 (60%) is illustrated in Table 3.11. The algorithm processes FCIs in ascending order by their length. Itemsets of the same length are stored in a table. Note that itemsets do not have to be ordered in a table. In dataset D by $min\_supp = 3$ there are six frequent closed itemsets altogether. The largest FCI is a 3-itemset. Filtering of maximal frequent itemsets works the following way. First, all itemsets are marked "maximal" in T_1. Then, using T_2, itemsets {C} and {A} are marked "not maximal" because they have a frequent proper superset in T_2. All itemsets are marked "maximal" in T_2, but {BE} turns out to be "not maximal" because of {BCE} in T_3. At the end, there are 3 itemsets marked "maximal": {AC}, {BCE} and {ABE}. From these three itemsets, all frequent itemsets could be restored.

Experimental Results

We compared the efficiency of *Charm-MFI* with *Charm*. The execution times of the algorithms on different datasets is illustrated in Table 3.12. This table also shows the number of FIs, the number of FCIs, the number of MFIs, the proportion of the number of FCIs to the number of FIs and the proportion of the number of MFIs to the number of FIs. It can be seen that MFIs are typically orders of magnitude fewer than all frequent itemsets, especially on dense datasets.

As a conclusion, we can say that *Charm-MFI* gives almost equivalent response times to *Charm* on both weakly and strongly correlated data, i.e. the filtering of maximal frequent itemsets among frequent closed itemsets is not an expensive process. The idea presented here can be easily applied to *any* frequent closed itemset mining algorithm.

min_supp (%)		Charm	Charm-MFI		# FIs	# FCIs	# MFIs	$\frac{\#FCIs}{\#FIs}$	$\frac{\#MFIs}{\#FIs}$
T20I6D100K									
2		32.80	33.24		378	378	365	100.00%	96.56%
1		73.25	74.63		1,534	1,534	914	100.00%	59.58%
0.75		91.77	91.52		4,710	4,710	1,791	100.00%	38.03%
0.5		135.87	134.98		26,836	26,208	4,520	97.66%	16.84%
0.25		361.96	372.34		155,163	149,217	12,467	96.17%	8.03%
C20D10K									
50		1.35	1.24		1,823	456	6	25.01%	0.33%
40		1.55	1.34		2,175	544	3	25.01%	0.14%
30		1.60	1.41		5,319	951	24	17.88%	0.45%
20		2.28	1.97		20,239	2,519	26	12.45%	0.13%
10		3.99	3.65		89,883	8,777	152	9.76%	0.17%
MUSHROOMS									
60		0.86	0.91		51	19	7	37.25%	13.73%
50		0.86	0.96		163	45	17	27.61%	10.43%
40		0.91	0.96		505	124	39	24.55%	7.72%
30		1.12	1.19		2,587	425	89	16.43%	3.44%
20		1.43	1.53		53,337	1,169	245	2.19%	0.46%

Table 3.12: Response times of *Charm-MFI*.

Chapter 4

Frequent Association Rules

Finding association rules is one of the most important tasks in data mining today. Generating valid association rules (denoted by $\mathcal{AR}$) from frequent itemsets often results in a huge number of rules, which limits their usefulness in real life applications. To solve this problem, different concise representations of association rules have been proposed, e.g. generic basis ($\mathcal{GB}$), informative basis ($\mathcal{IB}$) [BTP$^+$00b], representative rules ($\mathcal{RR}$) [Kry98], Duquennes-Guigues basis ($\mathcal{DG}$) [GD86], Luxenburger basis ($\mathcal{LB}$) [Lux91], proper basis ($\mathcal{PB}$), structural basis ($\mathcal{SB}$) [PBTL99a], etc. A very good comparative study of these bases can be found in [Kry02], where it is stated that a rule representation should be *lossless* (should enable the derivation of all valid rules), *sound* (should forbid the derivation of rules that are not valid) and *informative* (should allow the determination of rules parameters such as support and confidence).

In this chapter, we present different sets of association rules, namely all valid association rules and the family of minimal non-redundant association rules. We also introduce a new basis called Closed Rules ($\mathcal{CR}$) that we position among the other sets of association rules. All these sets presented here can be extracted with the CORON platform. Pseudo code and running examples are provided in all cases. The chapter is organized[20] as follows: Sections 4.1 – 4.3 present the selected sets of association rules. In Section 4.4 we show how to calculate other interestingness measures beside support and confidence. In Section 4.5 we present an efficient way of support derivation for frequent itemsets. Experimental results are given in Section 4.6. Finally, we review the related works, and we close the chapter with the conclusions.

4.1 All Association Rules

From now on, by "all association rules" we mean all (frequent) *valid* association rules. The concept of association rules was introduced by Agrawal *et al.* in [AIS93]. Originally, the extraction of association rules was used on sparse market basket data. The first efficient algorithm for this task was *Apriori*. It is interesting to note that in [AIS93] Agrawal *et al.* extracted rules that only have one item in the consequent.
The generation of all association rules consists of two main steps:

1. Find all *frequent* itemsets P in dataset D, i.e. where $supp(P) \geq min\_supp$.
2. For each frequent itemset P_1 found, generate all confident association rules r of the form $P_2 \rightarrow (P_1 \setminus P_2)$, where $P_2 \subset P_1$ and $conf(r) \geq min\_conf$.

[20]The necessary basic concepts are provided in Section 3.1.

The more difficult task is the first step, which is computationally and I/O intensive. Some selected itemset mining algorithms for this task have been presented in the previous chapter.

Generating All Valid Association Rules

Once all frequent itemsets and their supports are known, this step can be done in a relatively straightforward manner. The general idea is the following: for every frequent itemset P_1, all subsets P_2 of P_1 are derived, and the ratio $supp(P_1)/supp(P_2)$ is computed.[21] If the result is higher or equal to min_conf, then the rule $P_2 \rightarrow (P_1 \setminus P_2)$ is generated.

The support of any subset P_3 of P_2 is greater than or equal to the support of P_2. Thus, the confidence of the rule $P_3 \rightarrow (P_1 \setminus P_3)$ is necessarily less than or equal to the confidence of the rule $P_2 \rightarrow (P_1 \setminus P_2)$. Hence, if the rule $P_2 \rightarrow (P_1 \setminus P_2)$ is not confident, then neither is the rule $P_3 \rightarrow (P_1 \setminus P_3)$. Conversely, if the rule $(P_1 \setminus P_2) \rightarrow P_2$ is confident, then all rules of the form $(P_1 \setminus P_3) \rightarrow P_3$ are confident. For example, if the rule $A \rightarrow BE$ is confident, then the rules $AB \rightarrow E$ and $AE \rightarrow B$ are confident as well.

Using this property for efficiently generating valid association rules, the algorithm works as follows [AMS+96]. For each frequent itemset P_1, all *confident* rules with one item in the consequent are generated. Then, using the `Apriori-Gen` function (see Algorithm 23) on the set of 1-long consequents, we generate consequents with 2 items. Only those rules with 2 items in the consequent are kept whose confidence is greater than or equal to min_conf. The 2-long consequents of the confident rules are used for generating consequents with 3 items, etc.

The Algorithm

Pseudo Code. Here we present a slightly modified version of the algorithm of Agrawal *et al.* for generating all association rules [AMS+96]. We present the algorithm in a more general way, thus it can be used for all association rules and for closed association rules too. Furthermore, the algorithm is extended to support other statistical measures, as we will see later in the chapter. Note that the support of the right side of a rule is not needed for the *support* and *confidence* values. If we do not want to calculate other statistical measures, this can be removed from Algorithm 9 (line 9) and Algorithm 10 (line 9). In the algorithm, L_k is a set of frequent k-itemsets (resp. frequent closed k-itemsets). The set AR collects the generated association rules.

`getSupportOf` function: this function returns the support of an arbitrary frequent itemset. Since all frequent itemsets have already been explored before, finding the support of an itemset is a trivial task. Note that this function will be modified when generating closed association rules (Section 4.2).

`Apriori-Gen` function: this function is already known from *Apriori*. Using m-long itemsets, it generates their $(m+1)$-long supersets. For a detailed description see Algorithm 23.

[21] $supp(P_1)/supp(P_2)$ is the confidence of the rule $P_2 \rightarrow (P_1 \setminus P_2)$.

Algorithm 9 (main block):

Description: generates all association rules (resp. closed association rules)
Input: frequent itemsets (resp. frequent closed itemsets)
Output: all association rules (resp. closed association rules)

```
1)  loop over the elements of $L_k$ where $k \geq 2$ ($l_k$)
2)  {
3)    $H_1 \leftarrow$ {itemsets of size 1 that are subsets of $l_k$};
4)    loop over the elements of $H_1$ ($h_1$)
5)    {
6)      $leftSide \leftarrow (l_k \setminus h_1)$;
7)      $rightSide \leftarrow h_1$;
8)      $leftSupp \leftarrow$ getSupportOf($leftSide$);
9)      $rightSupp \leftarrow$ getSupportOf($rightSide$); // for other measures only
10)
11)     $conf \leftarrow$ getSupportOf($l_k$) / $leftSupp$;
12)     if ($conf \geq min\_conf$) then $AR \leftarrow AR \cup \{r : leftSide \rightarrow rightSide\}$;
13)     else $H_1 \leftarrow H_1 \setminus \{h_1\}$;
14)   }
15)   Gen-Rules($l_k$, $H_1$);
16) }
17) return $AR$;
```

Algorithm 10 (Gen-Rules procedure):

Input: l_k – frequent k-itemsets (resp. frequent closed k-itemsets)
H_m – set of m-long consequents

```
1)  if ($k > m + 1$)
2)  {
3)    $H_{m+1} \leftarrow$ Apriori-Gen($H_m$);
4)    loop over the elements of $H_{m+1}$ ($h_{m+1}$)
5)    {
6)      $leftSide \leftarrow (l_k \setminus h_{m+1})$;
7)      $rightSide \leftarrow h_{m+1}$;
8)      $leftSupp \leftarrow$ getSupportOf($leftSide$);
9)      $rightSupp \leftarrow$ getSupportOf($rightSide$); // for other measures only
10)
11)     $conf \leftarrow$ getSupportOf($l_k$) / $leftSupp$;
12)     if ($conf \geq min\_conf$) then $AR \leftarrow AR \cup \{r : leftSide \rightarrow rightSide\}$;
13)     else $H_{m+1} \leftarrow H_{m+1} \setminus \{h_{m+1}\}$;
14)   }
15)   Gen-Rules($l_k$, $H_{m+1}$);
16) }
```

$\mathcal{AR}$	supp.	conf.	$\mathcal{CR}$	$\mathcal{GB}$	$\mathcal{IB}$	$\mathcal{MNR}$
$B \rightarrow A$	3	0.75				
$A \rightarrow B$	3	0.75				
$C \rightarrow A$	3	0.75	+		+	+
$A \rightarrow C$	3	0.75	+		+	+
$E \rightarrow A$	3	0.75				
$A \rightarrow E$	3	0.75				
$C \rightarrow B$	3	0.75				
$B \rightarrow C$	3	0.75				
$E \Rightarrow B$	4	1.0	+	+		+
$B \Rightarrow E$	4	1.0	+	+		+
$E \rightarrow C$	3	0.75				
$C \rightarrow E$	3	0.75				
$BE \rightarrow A$	3	0.75	+			
$AE \Rightarrow B$	3	1.0	+	+		+
$AB \Rightarrow E$	3	1.0	+	+		+
$E \rightarrow AB$	3	0.75	+		+	+
$B \rightarrow AE$	3	0.75	+		+	+
$A \rightarrow BE$	3	0.75	+		+	+
$CE \Rightarrow B$	3	1.0	+	+		+
$BE \rightarrow C$	3	0.75	+			
$BC \Rightarrow E$	3	1.0	+	+		+
$E \rightarrow BC$	3	0.75	+		+	+
$C \rightarrow BE$	3	0.75	+		+	+
$B \rightarrow CE$	3	0.75	+		+	+

Table 4.1: Different sets of association rules extracted from dataset D (Table 3.1) by $min\_supp = 3$ (60%) and $min\_conf = 0.5$ (50%).

Example. Table 4.1 depicts which valid association rules ($\mathcal{AR}$) can be extracted from dataset D (Table 3.1) by $min\_supp = 3$ (60%) and $min\_conf = 0.5$ (50%). First, all frequent itemsets have to be extracted from the dataset. In D by $min\_supp = 3$ there are 12 frequent itemsets, as shown in Table B.3. Only those itemsets can be used for generating association rules that contain at least 2 items. Eight itemsets satisfy this condition. Given the itemset ABE, which is composed of 3 items, the following rules can be generated: $BE \rightarrow A$ (supp: 3; conf: 0.75), $AE \Rightarrow B$ (3; 1.0) and $AB \Rightarrow E$ (3; 1.0). Since all these rules are confident, their consequents are used to generate 2-long consequents: AB, AE and BE. This way, the following rules can be constructed: $E \rightarrow AB$ (3; 0.75), $B \rightarrow AE$ (3; 0.75) and $A \rightarrow BE$ (3; 0.75). In general, it can be said that from an m-long itemset, one can potentially generate $2^m - 2$ association rules.

4.2 Closed Association Rules

In the previous subsection we presented all association rules that are generated from frequent itemsets. Unfortunately, the number of these rules can be very large, and many of these rules are redundant, which limits their usefulness. Applying concise rule representations (a.k.a. *bases*) with appropriate inference mechanisms can lessen the problem [Kry02]. By definition, a *concise representation of association rules* is a subset of all association rules with the following properties: **(1)** it is much smaller than the set of all association rules, and **(2)** the whole set of all association rules can be restored from this subset (possibly with no access to the database, i.e. very efficiently) [JB02].

Related Work. In addition to the first method presented in Section 4.1, there are two other approaches for finding all association rules. The second approach was introduced in [PBTL99c] by Bastide *et al.* They have shown that frequent closed itemsets are a lossless, condensed representation of frequent itemsets, since the whole set of frequent itemsets can be restored from them with the proper support values. They propose the following method for finding all association rules. First, they extract frequent closed itemsets[22], then they restore the set of frequent itemsets from them, and finally they generate all association rules. The number of FCIs is usually much less than the number of FIs, especially in dense and highly correlated datasets. In such databases the exploration of all association rules can be done more efficiently by this way. However, this method has some disadvantages: **(1)** the restoration of FIs from FCIs needs *lots of* memory, **(2)** the final result is still "all the association rules", which means lots of redundant rules.

The third approach is based on the extraction of maximal frequent itemsets. Recall that a maximal frequent itemset has the following properties: all its proper supersets are infrequent and all its subsets are frequent. Experiments have shown that this approach is very efficient for finding large itemsets in databases [Bay98, AAP00, LK98, GZ01], but these itemsets cannot be used very efficiently for rule generation since the support values of their subsets are not known, i.e. MFIs are a condensed but not a lossless representation of FIs. Even though it is not difficult to make one more database pass to count the support of the subsets, due to the high number of inclusion tests it can be a very expensive step. Algorithms based on this approach identify all association rules, like *Apriori*.

Contribution. We introduce a new basis called Closed Association Rules, or simply Closed Rules ($\mathcal{CR}$). This basis requires frequent closed itemsets only. The difference between our work and the work presented in [PBTL99c] stems from the fact that although we also extract FCIs, instead of restoring FIs from them, we use them *directly* to generate valid association rules. This way, we find less and probably more interesting association rules.

$\mathcal{CR}$ is a generating set for all valid association rules with their proper support and confidence values. Our basis fills a gap between all association rules and minimal non-redundant association rules ($\mathcal{MNR}$), as depicted in Figure 4.1. $\mathcal{CR}$ contains all valid rules that are derived from frequent closed itemsets. Since the number of FCIs are usually much less than the number of FIs, the number of rules in our basis is also much less than the number of all association rules. Using our basis the restoration of all valid association rules can be done without any loss of information. It is possible to deduce efficiently, without access to the dataset, all valid association rules with their supports and confidences from this basis, since frequent closed itemsets are a lossless representation of frequent itemsets. Furthermore, we will show that minimal non-redundant association rules are a special subset of the Closed Rules, i.e. $\mathcal{MNR}$ can be defined in the framework of our basis. $\mathcal{CR}$ has the advantage that its rules can be generated very easily since only the frequent closed itemsets are needed. As there are usually much less FCIs than FIs, the derivation of the Closed Rules can be done much more efficiently than generating all association rules.

[22]For this task they introduced a new algorithm called "Close". Close is a levelwise algorithm for finding FCIs.

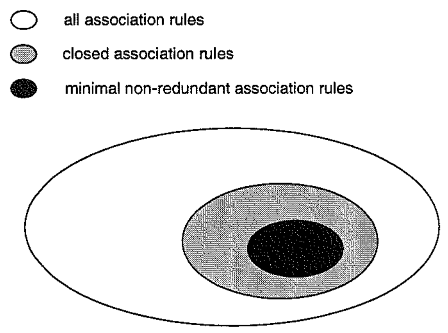


Figure 4.1: Position of Closed Rules.

Definitions

Before showing our algorithm for finding the Closed Rules, we present the essential definitions.

Definition 4.1 (closed association rule) *An association rule r: $P_1 \rightarrow P_2$ is called* closed *if $P_1 \cup P_2$ is a closed itemset.*

This definition means that the rule is derived from a closed itemset.

Definition 4.2 (Closed Rules) *Let FC be the set of frequent closed itemsets. The set of Closed Rules contains* all *valid closed association rules:*

$$\mathcal{CR} = \{r : P_1 \rightarrow P_2 \mid (P_1 \cup P_2) \in FC \wedge supp(r) \geq min\_supp \wedge conf(r) \geq min\_conf\} \ .$$

Property 4.1 *The support of an arbitrary frequent itemset is equal to the support of its smallest frequent closed superset [PBTL99c].*

By this property, FCIs are a condensed lossless representation of FIs. This is also called the *frequent closed itemset representation* of frequent itemsets. Property 4.1 can be generalized the following way:

Property 4.2 *If an arbitrary itemset X has a frequent closed superset, then X is frequent and its support is equal to the support of its smallest frequent closed superset. If X has no frequent closed superset, then X is not frequent.*

The idea behind generating all valid association rules is the following. First we need to extract all frequent itemsets. Then rules of the form $X \setminus Y \rightarrow Y$, where $Y \subset X$, are generated for all frequent itemsets X, provided the rules have at least minimum confidence.

Finding closed association rules is done similarly. However, this time we only have frequent *closed* itemsets available. In this case the left side of a rule $X \setminus Y$ can be non-closed. For calculating the confidence of rules its support must be known. Thanks to Property 4.1, this support value can be calculated by only using frequent closed itemsets. It means that only FCIs are needed; all frequent itemsets do not have to be extracted. This is the principle idea behind this part of our work.

The Algorithm

Pseudo Code. Algorithms 9 and 10 in Section 4.1 were presented in a general way, thus they are the same for Closed Rules. There are two differences. First, Algorithm 9 gets frequent closed itemsets as its input. Secondly, the getSupportOf function must be adapted to frequent closed itemsets.

`getSupportOf` function: this function returns the support of an arbitrary itemset, either closed or non-closed. If the itemset is closed, then its support is known since FCIs have been extracted. If the itemset is not among closed itemsets, then its support is derived by using Property 4.1.

`getSmallestSupersetOf` function: this function finds the smallest superset of the input frequent itemset among the frequent closed itemsets. This function can be implemented very efficiently using the trie data structure (see Appendix C.3).

Algorithm 11 (getSupportOf function):

```
Input:   $it$ – an arbitrary itemset (closed or non-closed)
Output:  support value of $it$

1) if $it$ is in the list of closed itemsets then return $it$.support; // i.e. $it$ is a closed itemset
2) // else, if $it$ is a non-closed frequent itemset
3) $s \leftarrow$ getSmallestSupersetOf($it$);
4) return $s$.support;
```

Example. Table 4.1 depicts which closed association rules ($\mathcal{CR}$) can be extracted from dataset D (Table 3.1) by $min\_supp = 3$ (60%) and $min\_conf = 0.5$ (50%). First, frequent closed itemsets must be extracted from the dataset. In D by $min\_supp = 3$ there are 6 FCIs, as shown in Table B.4. Note that the total number of frequent itemsets by these parameters is 12. Only those itemsets can be used for generating association rules that contain at least 2 items. There are 4 itemsets that satisfy this condition, namely itemsets AC (supp: 3), BE (4), ABE (3) and BCE (3). Let us see which rules can be generated from the itemset BCE for instance. Algorithm 9 produces three rules: $CE \rightarrow B$, $BE \rightarrow C$ and $BC \rightarrow E$. Their support is known, it is equal to the support of BCE. To calculate the confidence values we need to know the support of the left sides too. The support of BE is known since it is a closed itemset, but CE and BC are non-closed. Their supports can be derived by Property 4.1. The smallest frequent closed superset of both CE and BE is BCE, thus their supports are equal to the support of this closed itemset, which is 3. Algorithm 10 produces three more rules: $E \rightarrow BC$, $C \rightarrow BE$ and $B \rightarrow CE$. Their confidence values are calculated similarly. From the four frequent closed itemsets 16 closed association rules can be extracted altogether. In general, we can say that from an m-long itemset one can potentially generate $2^m - 2$ association rules.

4.3 Family of "Minimal Non-Redundant Association Rules"

In this section, we present the family of minimal non-redundant association rules. We call it a family because 5 different sets of these rules can be distinguished.

Definition 4.3 (generic basis for exact association rules) *Let FC be the set of frequent closed itemsets. For each frequent closed itemset* f, *let FG_f denote the set of frequent generators of* f. *The generic basis:*

$$\mathcal{GB} = \{r : g \Rightarrow (f \backslash g) \mid f \in FC \wedge g \in FG_f \wedge g \neq f\} \ .$$

Definition 4.4 (informative basis for approximate association rules) *Let FC be the set of frequent closed itemsets and let FG denote the set of frequent generators. The notation $\gamma(g)$ signifies the closure of itemset* g. *The informative basis:*

$$\mathcal{IB} = \{r : g \rightarrow (f \backslash g) \mid f \in FC \wedge g \in FG \wedge \gamma(g) \subset f\} \ .$$

Definition 4.5 (transitive reduction of the informative basis) *Let $\mathcal{IB}$ be the informative basis for approximate association rules, and let FC denote the set of frequent closed itemsets. The transitive reduction of the informative basis:*

$$\mathcal{RIB} = \{r : g \rightarrow (f \backslash g) \in \mathcal{IB} \mid \gamma(g) \text{ is a maximal proper subset of } f \text{ in } FC\} \ .$$

Definition 4.6 *Minimal non-redundant rules ($\mathcal{MNR}$) are defined as: $\mathcal{MNR} = \mathcal{GB} \cup \mathcal{IB}$.*

Definition 4.7 *Transitive reduction of minimal non-redundant rules ($\mathcal{RMNR}$) is defined as: $\mathcal{RMNR} = \mathcal{GB} \cup \mathcal{RIB}$.*

Clearly, $\mathcal{MNR}$ is the largest set of these rules, and $\mathcal{GB}$, $\mathcal{IB}$, $\mathcal{RIB}$ and $\mathcal{RMNR}$ are its subsets. These five sets form the family of minimal non-redundant association rules. The following inclusions are true for these sets: $\mathcal{RIB} \subseteq \mathcal{IB}$, $\mathcal{RMNR} \subseteq \mathcal{MNR}$, $\mathcal{GB} \subseteq \mathcal{RMNR} \subseteq \mathcal{MNR}$, $\mathcal{IB} \subseteq \mathcal{MNR}$ and $\mathcal{RIB} \subseteq \mathcal{RMNR}$. From the definitions above it can be seen that we only need frequent closed itemsets *and* their generators to produce these rules. In the previous chapter we have presented some algorithms that are designed specifically for finding the $\mathcal{MNR}$ rules (like *Zart* and *Eclat-Z*).

Kryszkiewicz has shown in [Kry02] that $\mathcal{MNR}$ with the cover operator, and $\mathcal{RMNR}$ with the cover operator and the confidence transitivity property are *lossless* (they enable the derivation of all valid rules), *sound* (they forbid the derivation of rules that are not valid) and *informative* (they allow the determination of rules parameters such as support and confidence) representations of *all* valid association rules.

Frequent itemsets have several condensed representations, e.g. closed itemsets [PBTL99c, PBTL99b, STB$^+$02, ZH02], generators representation [Kry01, BTP$^+$00a], approximate free-sets [BBR00], disjunction-free sets [BR01], disjunction-free generators [Kry01], generalized disjunction-free generators [KG02] and non-derivable itemsets [CG02, CG05]. However, from the application point of view, the most useful representations are frequent closed itemsets and frequent generators that proved to be useful not only in the case of representative and non-redundant association rules, but as well as representative episode rules [HDST01], which constitute concise, lossless representations of all association/episode rules of interest. Common feature of these rule representations is that only closed itemsets and generators are involved.

The Algorithm

Pseudo Code. Here we present an algorithm that finds $\mathcal{MNR}$ or $\mathcal{RMNR}$ (Algorithm 12). The algorithm can be easily adapted to $\mathcal{GB}$, $\mathcal{IB}$ and $\mathcal{RIB}$. As input, three things are needed: **(1)** frequent closed itemsets, **(2)** frequent generators of closed itemsets, and **(3)** support values of itemsets. The right side of Table 3.5 shows a sample input, produced by *Zart.*

`getProperSupersets` function: it has two parameters: a set of itemsets S and an itemset p. The function returns the proper supersets of p in S. This function can be implemented very efficiently using the trie data structure (see Appendix C.3).

`getSupportOf` function: this function returns the support of an arbitrary frequent itemset (Algorithm 13). This time FCIs and FGs are all available, thus first we check if the given itemset is among them. If yes, then its support is known, otherwise we will have to look up its support from the trie of FCIs using Property 4.1.

Finding MNR Rules with Zart. By Definitions 4.3 – 4.7, rules in the family of minimal non-redundant association rules have the following form: the antecedent is a frequent generator, the union of the antecedent and consequent is a frequent closed itemset, and the antecedent is a proper subset of this frequent closed itemset.

Shortly, the algorithm for finding $\mathcal{MNR}$ rules is the following: for each frequent generator P_1 find its proper supersets P_2 in the set of FCIs. Then add the rule $r : P_1 \rightarrow P_2 \setminus P_1$ to the set of $\mathcal{MNR}$. As can be seen, for the generation of such rules the frequent closed itemsets *and* their generators are needed. Since *Zart* can find both, the output of *Zart* (see Table 3.5) can be used directly to generate these rules. For a very quick lookup of proper supersets of generators we suggest storing the frequent closed itemsets in the trie data structure (see Appendix C.1 and Appendix C.3).

Example. Figure 3.4 shows what equivalence classes are found in dataset D (Table 3.1) by $min\_supp = 2$. For instance, using the generator E in Figure 3.4, three rules can be determined. Rules within an equivalence class form the generic basis ($\mathcal{GB}$), which are exact association rules ($E \Rightarrow B$), while rules between equivalence classes form the informative basis ($\mathcal{IB}$) that are approximate rules ($E \rightarrow BC$ and $E \rightarrow ABC$). For extracting $\mathcal{RMNR}$, the search space for finding frequent closed proper supersets of generators is reduced to equivalence classes that are in direct subsumption relation (see Def. 3.5), i.e. transitive subsumption relations are eliminated. Thus, for instance, in the previous example only the first two rules are generated: $E \Rightarrow B$ and $E \rightarrow BC$. Removing exact association rules from $\mathcal{RMNR}$, we can get the transitive reduction of the informative basis ($\mathcal{RIB}$).

Table 4.1 depicts the sets $\mathcal{GB}$, $\mathcal{IB}$ and $\mathcal{MNR}$ extracted from dataset D (Table 3.1) if $min\_supp = 3$ (60%) and $min\_conf = 0.5$ (50%).

4.3.1 Why MNR Rules are "Minimal" and "Non-Redundant"

Definition 4.6 simply states that $\mathcal{MNR}$ rules have the following form: $P \rightarrow Q \setminus P$, where $P \subset Q$, P is a generator and Q is a closed itemset. That is, an $\mathcal{MNR}$ rule has a minimal antecedent and a maximal consequent. Minimal (resp. maximal) means that the antecedent (resp. consequent) is a minimal (resp. maximal) element in its equivalence class. Note that P and Q are not necessarily in the same equivalence class.

Kryszkiewicz gives a different definition of $\mathcal{MNR}$ rules in [Kry02]. We will refer to this definition

Algorithm 12:

Description: generates $\mathcal{MNR}$ or $\mathcal{RMNR}$
Input: FCIs + their generators + support values
Output: set of $\mathcal{MNR}$ or $\mathcal{RMNR}$

```
loop over the generators ($g$)
{
  // find proper supersets of $g$ among the frequent closed itemsets:
  $C_G \leftarrow$ getProperSupersets(FCI, $g$);

  loop over the elements of $C_G$ ($c$)
  {
    $leftSide \leftarrow g$;
    $rightSide \leftarrow (c \setminus g)$;
    $leftSupp \leftarrow g$.support;
    $rightSupp \leftarrow$ getSupportOf($rightSide$); // for other measures only

    $rule \leftarrow (leftSide \rightarrow rightSide)$;

    if (conf($rule$) $\geq min\_conf$) // conf($rule$) = supp($c$) / supp($g$)
    {
      /* The following step is optional. It is to be executed
         if we want to extract $\mathcal{RMNR}$ instead of $\mathcal{MNR}$. */
      if (conf($rule$) $\neq$ 1.0)
      {
        // delete proper supersets of $c$ in $C_G$:
        $C_G \leftarrow C_G \setminus$ getProperSupersets($C_G$, $c$);
      }
      $R \leftarrow R \cup \{rule\}$;
    }
  }
}
return $R$;
```

Algorithm 13 (getSupportOf function):

Input: *it* – an arbitrary itemset
Output support value of *it*

```
if $it$ is in the list of closed itemsets then return $it$.support;
// else, if $it$ is non-closed
if $it$ is in the list of frequent generators then return $it$.support;
// else, if $it$ is non-closed and non-generator
$s \leftarrow$ getSmallestSupersetOf($it$);
return $s$.support;
```

as "Kryszkiewicz's definition of $\mathcal{MNR}$ rules". As we will see later, this definition is equivalent to Def. 4.6:

$$\mathcal{MNR} = \{(r_1 : X_1 \rightarrow Y_1 \setminus X_1) \in \mathcal{AR} \mid \neg\exists(r_2 : X_2 \rightarrow Y_2 \setminus X_2) \in \mathcal{AR}, r_2 \neq r_1 \wedge \\ X_2 \subseteq X_1 \wedge Y_2 \supseteq Y_1 \wedge supp(r_2) = supp(r_1) \wedge conf(r_2) = conf(r_1)\} \ .$$

In other words, it means the following. Let r_2 be a rule. If there exists a rule r_1 (different from r_2) that we can get by shortening the antecedent of r_2 and/or enlarging the consequent of r_2 (by moving itemsets from the left side of the rule to the right side), and the parameters (support and confidence) of r_1 are equal to the parameters of r_2, then r_2 is not an $\mathcal{MNR}$ rule. Why not? **(1)** *The rule r_2 is not minimal.* As mentioned before, Kryszkiewicz's definition of $\mathcal{MNR}$ rules is equivalent to Def. 4.6, i.e. $\mathcal{MNR}$ rules have minimal antecedents and maximal consequents. The rule r_2 does not satisfy this condition. **(2)** *The rule r_2 is redundant*, because from the set of $\mathcal{MNR}$ rules, this rule can be deduced with its proper support and confidence values.

We show here that Kryszkiewicz's definition of $\mathcal{MNR}$ rules defines a set of rules that have the following form: $P \rightarrow Q \setminus P$, where $P \subset Q$ and P is a *generator* and Q is a *closed itemset.*

Minimal antecedent. Let r_1 and r_2 be two rules:
$r_1 : P_1 \rightarrow Q \setminus P_1$
$r_2 : P_2 \rightarrow Q \setminus P_2$

$supp(r_1) = supp(r_2) = supp(Q)$
$conf(r_1) = supp(Q)/supp(P_1)$
$conf(r_2) = supp(Q)/supp(P_2)$

Make the following hypothesis: $conf(r_1) = conf(r_2)$ and $P_1 \subseteq P_2$. Since $conf(r_1) = conf(r_2)$, we can deduce that $supp(P_1) = supp(P_2)$. As $P_1 \subseteq P_2$ and $supp(P_1) = supp(P_2)$, it means that P_1 and P_2 are in the same equivalence class (by Lemma 3.2).

That is, those rules contain the most information that have the smallest antecedents. The minimal elements of an equivalence class are the generators. Thus, P_1 is a generator.

Maximal consequent. Let r_1 and r_2 be two rules:
$r_1 : P \rightarrow Q_1 \setminus P$
$r_2 : P \rightarrow Q_2 \setminus P$

Make the following hypothesis:

$Q_1 \subseteq Q_2$ and
$supp(r_1) = supp(r_2)$ and
$conf(r_1) = conf(r_2)$.

If $supp(r_1) = supp(r_2)$, then $supp(Q_1) = supp(Q_2)$. If $Q_1 \subseteq Q_2$ and $supp(Q_1) = supp(Q_2)$, then it means that Q_1 and Q_2 are in the same equivalence class.

That is, those rules contain the most information that have the largest consequents. The largest elements of an equivalence class is the closure of the equivalence class. This is a closed itemset, which is unique in its equivalence class. Thus, Q_2 is a closed itemset.

4.3.2 Deducing Valid Association Rules from MNR Rules

Minimal non-redundant association rules contain the most information among rules with the same support and same confidence. This statement underlies the following reasons.

Let r_1: $P_1 \rightarrow Q \backslash P_1$ be a valid (not necessarily an $\mathcal{MNR}$) rule, where $P_1 \subset Q$. If r_1: $P_1 \rightarrow Q \backslash P_1$ is a valid rule, then r_2: $P_2 \rightarrow Q \setminus P_2$ is also a valid rule, where $P_1 \subseteq P_2$.

If $P_1 \subseteq P_2$, then $supp(P_1) \geq supp(P_2)$.
$supp(r_1) = supp(r_2) = supp(Q)$
$conf(r_1) = supp(Q)/supp(P_1)$
$conf(r_2) = supp(Q)/supp(P_2)$
Since $supp(P_1) \geq supp(P_2)$, $conf(r_1) \leq conf(r_2)$.

As can be seen, most valid rules can be deduced from rules with minimal antecedents and maximal consequents.

4.4 Other Statistical Measures

Typically, the number of extracted association rules in a dataset is very large, and only a few of these rules are likely to be of interest to the domain expert analyzing the data. To increase the utility, relevance and usefulness of the discovered rules, techniques are required to reduce the number of rules that need to be considered. There are two well-known techniques:

1. The number of all association rules is usually very large, and many of these association rules are either irrelevant or obvious, and do not provide new knowledge. In order to reduce the number of extracted rules, different *bases* have been defined. In the previous section we have seen two bases (closed rules and minimal non-redundant rules) that try to find the most interesting rules instead of generating all possible valid rules.

2. Another approach that tries to find the most interesting rules is referred to as *interestingness measures* (or *quality measures*). The most well-known statistical measures are *support* and *confidence*, but beside them there exist a large number of other statistical measures too [LFZ99, TKS02, CNT03, HH99]. In this section we investigate some of them.

The best result may be achieved by combining these two techniques.

4.4.1 Interestingness Measures

In this section, the following interestingness measures are investigated: *interest* (or *lift*), *conviction*, *dependency*, *novelty* and *satisfaction*. We will use the following terminology in the section: the antecedent of a rule is referred to as *left side* and denoted by L; the consequent of a rule is referred to as *right side* and denoted by R. Support values are *relative* values this time.[23] The following data are needed for these measures:

- support of the left side (supp(L))

[23]Recall that the absolute support (denoted by supp) of an association rule r: $P_1 \rightarrow P_2$ is: supp(r) = supp($P_1 \cup P_2$). The relative support (denoted by rsupp) of the same rule is: rsupp(r) = supp(r)/|O|, where |O| signifies the total number of objects in the input database.

- support of the right side ($\texttt{supp(R)}$)
- support of the union of the left and right sides ($\texttt{supp}(\texttt{L} \cup \texttt{R})$)
- total number of objects in the input database (having this, supports in absolute values can be easily converted to relative values)

Some measures require the support of the negation of an itemset, e.g. $\texttt{rsupp}(\neg\texttt{L})$, which means: proportion of objects not including itemset L. Calculating this value is very easy: $\texttt{rsupp}(\neg\texttt{L}) = 1 - \texttt{rsupp}(\texttt{L})$.

A comparative table of the characteristics of the different measures is shown in Table 4.2. In the "Preferred value" column, the arrow $\nearrow$ (resp. $\searrow$) indicates that a higher (resp. lower) value signifies a more interesting rule.

Support

The support of the rule $\texttt{L} \rightarrow \texttt{R}$ shows how many objects include L and R at the same time.

$$\texttt{supp}(\texttt{L} \rightarrow \texttt{R}) = \texttt{supp}(\texttt{L} \cup \texttt{R})$$

In the rest of the thesis we use absolute values, but interestingness measures require *relative* values, i.e. absolute values must be divided by the total number of objects in the dataset ($|\texttt{O}|$). The relative support of the rule $\texttt{L} \rightarrow \texttt{R}$ is the probability $\texttt{P}(\texttt{L} \cup \texttt{R})$, which shows the probability of having L and R at the same time.

$$\texttt{rsupp}(\texttt{L} \rightarrow \texttt{R}) = \texttt{P}(\texttt{L} \cup \texttt{R}) = \frac{\texttt{supp}(\texttt{L} \cup \texttt{R})}{|\texttt{O}|}$$

This way, relative support values are between 0 and 1.0. Example: given the itemset BCE in dataset D (Table 3.1), its absolute support is 3 and its relative support is 3/5 = 0.6.

Confidence

The confidence of the rule $\texttt{L} \rightarrow \texttt{R}$ is the conditional probability $\texttt{P}(\texttt{R}|\texttt{L})$. The confidence shows the conditional probability that an object includes itemset R, given that it includes itemset L. The value of confidence is a real number between 0 and 1.0.

$$\texttt{conf}(\texttt{L} \rightarrow \texttt{R}) = \texttt{P}(\texttt{R}|\texttt{L}) = \frac{\texttt{P}(\texttt{L} \cup \texttt{R})}{\texttt{P}(\texttt{L})} = \frac{\texttt{supp}(\texttt{L} \cup \texttt{R})}{\texttt{supp}(\texttt{L})} = \frac{\texttt{rsupp}(\texttt{L} \cup \texttt{R})}{\texttt{rsupp}(\texttt{L})}$$

Interest (Lift)

The interest [IBM98] (or lift) measures the degree of compatibility of L and R, i.e. the simultaneous occurrences of both events L and R. Possible values: $[0, +\infty[$. If $\texttt{lift} = 1$, then L and R are independent. The closer is the value of lift to 0, the more L and R are incompatible. The larger the value of lift than 1, L and R are more dependent. The lift is symmetric, i.e. $\texttt{lift}(\texttt{A} \rightarrow \texttt{B}) = \texttt{lift}(\texttt{B} \rightarrow \texttt{A})$.

$$\texttt{lift}(\texttt{L} \rightarrow \texttt{R}) = \frac{\texttt{P}(\texttt{L} \cup \texttt{R})}{\texttt{P}(\texttt{L}) \times \texttt{P}(\texttt{R})} = \frac{\texttt{conf}(\texttt{L} \rightarrow \texttt{R})}{\texttt{rsupp}(\texttt{R})} = \frac{\texttt{rsupp}(\texttt{L} \cup \texttt{R})}{\texttt{rsupp}(\texttt{L}) \times \texttt{rsupp}(\texttt{R})}$$

Conviction

The conviction [BMUT97] measures the deviation of the rule $\mathtt{A} \longrightarrow \mathtt{B}$ from the rule $\mathtt{A} \longrightarrow \neg\mathtt{B}$, or, in other words, how high is the degree of implication of the rule $\mathtt{A} \longrightarrow \neg\mathtt{B}$. Possible values: $[\mathtt{0}, +\infty[$. If $\mathtt{conv} > \mathtt{1}$, then L and R are dependent. When $\mathtt{conv} = \mathtt{1}$ then L and R are completely unrelated. A value in the interval $[\mathtt{0}, \mathtt{1}[$ signifies that L and R are not dependent. The value of conviction is not defined in the case of exact rules.

$$\mathtt{conv}(\mathtt{L} \longrightarrow \mathtt{R}) = \frac{\mathtt{P(L)} \times \mathtt{P}(\neg\mathtt{R})}{\mathtt{P}(\mathtt{L} \cup \neg\mathtt{R})} = \frac{\mathtt{rsupp(L)} \times \mathtt{rsupp}(\neg\mathtt{R})}{\mathtt{rsupp(L)} - \mathtt{rsupp}(\mathtt{L} \cup \mathtt{R})} = \frac{\mathtt{rsupp(L)} \times (\mathtt{1} - \mathtt{rsupp(R)})}{\mathtt{rsupp(L)} - \mathtt{rsupp}(\mathtt{L} \cup \mathtt{R})}$$

Note that in the formula $\mathtt{P}(\mathtt{L} \cup \neg\mathtt{R})$ is the probability of having L but not having R at the same time. In other words, this is the proportion of counter examples of the rule $\mathtt{L} \longrightarrow \mathtt{R}$.

Dependency

The dependency measures the degree of independence between the events L and R, i.e. the fact that the occurrence of the event L is or is not dependent on the occurrence of the event R. Possible values: $[\mathtt{0}, \mathtt{1}[$. A value closer to 0 means that L and R are independent. A value closer to 1 means that L and R are more dependent.

$$\mathtt{dep}(\mathtt{L} \longrightarrow \mathtt{R}) = |\mathtt{P(R|L)} - \mathtt{P(R)}| = |\mathtt{conf}(\mathtt{L} \longrightarrow \mathtt{R}) - \mathtt{rsupp(R)}|$$

Novelty

The novelty [PS91] is used to quantify the correlation between attributes in a rule. Possible values: $]-\mathtt{1}, \mathtt{1}[$. When $\mathtt{nov} = \mathtt{0}$, then L and R are statistically independent and the rule is not interesting. When $\mathtt{nov} > \mathtt{0}$ (resp. $\mathtt{nov} < \mathtt{0}$), then L is positively (resp. negatively) correlated to R. The value of novelty is close to $-\mathtt{1}$ in the case of rules with weak support, i.e. where $\mathtt{P}(\mathtt{L} \cup \mathtt{R}) \approx \mathtt{0}$. The novelty is symmetric, i.e. $\mathtt{nov}(\mathtt{A} \longrightarrow \mathtt{B}) = \mathtt{nov}(\mathtt{B} \longrightarrow \mathtt{A})$.

$$\mathtt{nov}(\mathtt{L} \longrightarrow \mathtt{R}) = \mathtt{P}(\mathtt{L} \cup \mathtt{R}) - \mathtt{P(L)} \times \mathtt{P(R)} = \mathtt{rsupp}(\mathtt{L} \cup \mathtt{R}) - (\mathtt{rsupp(L)} \times \mathtt{rsupp(R)})$$

Satisfaction

The possible values of the satisfaction fall in the interval $]-\infty, \mathtt{1}]$. If the rule is exact, then $\mathtt{sat} = \mathtt{1}$. Otherwise, the value of satisfaction is calculated by the formula shown below. If $\mathtt{sat} = \mathtt{0}$, then L and R are independent. The less the value of novelty and the more the value of satisfaction is, the more interesting a rule is considered.

$$\mathtt{sat}(\mathtt{L} \longrightarrow \mathtt{R}) = \frac{\mathtt{conv}(\mathtt{L} \longrightarrow \mathtt{R}) - \mathtt{1}}{\mathtt{conv}(\mathtt{L} \longrightarrow \mathtt{R})}$$

4.4.2 Combining Quality Measures

In [CNT03], Cherfi *et al.* propose an algorithm that allows one to combine the previously presented quality measures. The idea is the following. They take a set of association rules, and investigating their lift values, they choose the most interesting rules (in the case of lift a higher value indicates a more significant rule). From these rules, by their conviction, they choose again the rules that seem to be the most interesting. They continue so with each interestingness measures until they get a subset of rules that are significant from the point of view of all quality measures. Of course, this process requires the help of an analyst who can decide at each step what percentage of rules should be filtered.

Measure	Interval	Preferred value	Value of independency	Reference value(s)	Symmetric?
$\mathtt{lift}(\mathtt{L} \to \mathtt{R})$	$[0,+\infty[$	$\nearrow$	1	$=0$, incompatible	yes
$\mathtt{conv}(\mathtt{L} \to \mathtt{R})$	$[0,+\infty[$	$\nearrow$	1	>1, dependent $[0,1[$, not dep.	no
$\mathtt{dep}(\mathtt{L} \to \mathtt{R})$	$[0,1[$	$\nearrow$	0	≈ 1, dependent	no
$\mathtt{nov}(\mathtt{L} \to \mathtt{R})$	$]-1,1[$	$\searrow$	0	≈ -1, weak support	yes
$\mathtt{sat}(\mathtt{L} \to \mathtt{R})$	$]-\infty,1]$	$\nearrow$	0	$=1$, exact rule	no

Table 4.2: Characteristics of the different quality measures.

4.4.3 Example

Let us calculate all the quality measures of the approximate rule $\mathtt{B} \to \mathtt{CE}$ and the exact rule $\mathtt{AB} \Rightarrow \mathtt{E}$. Both of these rules are extracted from dataset D (Table 3.1).

$\mathtt{B} \to \mathtt{CE}$ ($\mathtt{supp} = 3$; $\mathtt{rsupp} = 3/5 = 0.6$ (60%); $\mathtt{suppLeft} = 4$; $\mathtt{rsuppLeft} = 4/5 = 0.8$ (80%); $\mathtt{suppRight} = 3$; $\mathtt{rsuppRight} = 3/5 = 0.6$ (60%); $\mathtt{conf} = \frac{3}{4} = 0.75$ (75%); $\mathtt{lift} = \frac{0.6}{0.8\times0.6} = 1.25$; $\mathtt{conv} = \frac{0.8\times(1-0.6)}{0.8-0.6} = 1.6$; $\mathtt{dep} = |0.75 - 0.6| = 0.15$; $\mathtt{nov} = 0.6 - (0.8 \times 0.6) = 0.12$; $\mathtt{sat} = \frac{1.6-1}{1.6} = 0.375$)

$\mathtt{AB} \Rightarrow \mathtt{E}$ ($\mathtt{supp} = 3$; $\mathtt{rsupp} = 3/5 = 0.6$ (60%); $\mathtt{suppLeft} = 3$; $\mathtt{rsuppLeft} = 3/5 = 0.6$ (60%); $\mathtt{suppRight} = 4$; $\mathtt{rsuppRight} = 4/5 = 0.8$ (80%); $\mathtt{conf} = 1.0$ (100%); $\mathtt{lift} = \frac{0.6}{0.6\times0.8} = 1.25$; $\mathtt{conv} = \mathtt{NOT\_DEFINED}$; $\mathtt{dep} = |1.0 - 0.8| = 0.2$; $\mathtt{nov} = 0.6 - (0.6 \times 0.8) = 0.12$; $\mathtt{sat} = 1.0$)

4.5 Efficient Support Derivation of Frequent Itemsets

In general, from an itemset P_1 one can generate association rules of the form $P_2 \to (P_1 \setminus P_2)$, where $P_2 \subset P_1$. To calculate interestingness measures, the support of *both* sides must be known. Considering a frequent association rule r: $P_1 \to P_2$, both P_1 and P_2 are frequent itemsets, since all subsets of a frequent itemset are frequent (Property 3.1). Calculating the quality measures of all association rules is a trivial task because the support of all frequent itemsets is known. The only problem may be the memory usage, knowing that the number of frequent itemsets is usually very large.

Calculating the quality measures of closed and minimal non-redundant association rules might result in a low performance rate. Previously we have seen that FCIs are a very useful representation of FIs. The number of FCIs is usually much less than the number of FIs, and using FCIs the support of any FI can be properly derived (Property 4.1). This task can be realized efficiently with the trie data structure (see Appendix C.3). Trie provides an efficient solution, but when the trie contains *many elements* and the trie is consulted *lots of times*, its efficiency can quickly degrade. This is especially true when computing other quality measures, not only support and confidence. Recall that support and confidence require the lookup of the support of the left side only, while the other measures require the support of both sides.

Here we present an *efficient* solution for seriously reducing the expensive trie accesses. Our solution is based on the observation that the support of a large number of itemsets is asked *lots of* times. Actually, this is not surprising, knowing that a huge number of association rules share the same antecedents and/or consequents. Our idea: once the support of an itemset is retrieved from the trie, store the itemset-support pair temporarily in a cache. Next time, when the support of the same itemset is asked, first look it up in the cache. If it is there, then its stored support value is sent back. If it is not in the cache, then we access the trie. The cache is implemented as a hash structure, allowing a *very quick look-up* of itemsets. Figure 4.2 shows the difference between the two approaches.

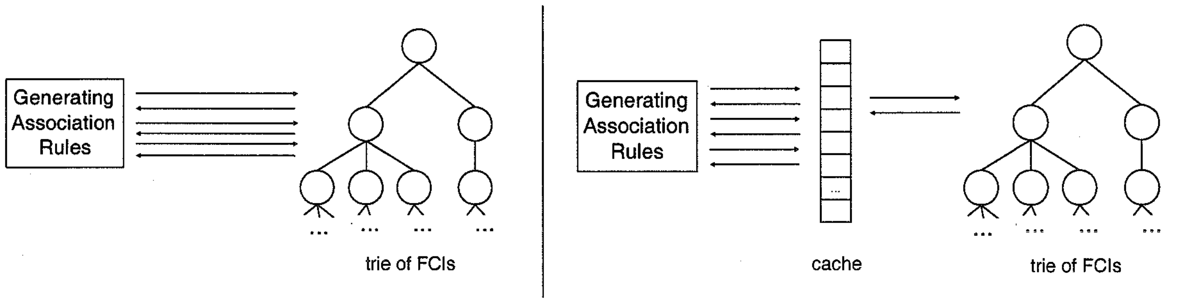


Figure 4.2: Non-optimized and optimized support derivation of frequent itemsets.

The Algorithm. The cache optimization can be added transparently to the previously presented association rule mining algorithms. Simply the `getSupportOf` function (Algorithm 11) must be extended as shown in Algorithm 14. (The extension of Algorithm 13 can be done similarly.) Additional lines are marked with an asterisk. As for the cache, it is a simple hash structure containing itemset-support pairs. The chosen hash function must provide a uniform distribution of elements in the hash. When an itemset is looked for and it is in the hash, then its associated support is returned. When an itemset is not cached, it is looked up in the trie and then stored in the cache, even if its slot is occupied. This means that an itemset can be overwritten in the cache. This way the cache is kept small and simple. Experiments show that this provides a very efficient solution.

Algorithm 14 (getSupportOf function with cache):

```
Input:   it – an arbitrary itemset (closed or non-closed)
Output   support value of it

   1)  if it is in the list of closed itemsets then return it.support; // i.e. it is a closed itemset
   2)  // else, if it is a non-closed frequent itemset
 * 3)  cachedItemset ← cache.get(it); // is it cached?
 * 4)  if (cachedItemset != null) return cachedItemset.support;
   5)  // else, if it is not cached
   6)  s ← getSmallestSupersetOf(it); // support of it is equal to s.support
 * 7)  cache.register(it, s.support); // register it with its support value
   8)  return s.support;
```

Experimental Results. In the experiments, we compared the extraction of closed (Table 4.3) and minimal non-redundant (Table 4.4) association rules with and without cache optimization.

The indicated time values show the time of rule generation (the extraction time of the necessary itemsets is not included). The tables contain the following information: *min_supp* and *min_conf* thresholds; number of generated rules; number of trie accesses without cache; time of rule generation without cache; number of trie accesses with cache; time of rule generation with cache. In all cases, support values of *both* sides were calculated.

As we can see, the generation of $\mathcal{MNR}$ requires much less trie access. It has two reasons. First, by definition, the left side of an $\mathcal{MNR}$ rule is a generator, thus its support is known. Secondly, beside FCIs, FGs are also available. Thus, if an itemset is non-closed, it can still be a generator, which means that its support is available. However, in the case of $\mathcal{CR}$s, when an itemset is non-closed, the trie has to be consulted in order to derive its support.

Using the cache, the number of trie accesses can be seriously reduced in all cases, which results in an improved performance. This performance gain is especially spectacular in the case of closed rules. It is worth noting that $\mathcal{CR}$s can be generated more efficiently than $\mathcal{MNR}$s in sparse datasets (T20I6D100K). Experiments clearly show the usefulness of a cache. Another advantage is that this extension can be easily implemented and added to other rule mining algorithms.

4.6 Experimental Results

Here we compare $\mathcal{AR}$, $\mathcal{CR}$ and $\mathcal{MNR}$. Experiments were carried out on three different databases, namely T20I6D100K, C20D10K and MUSHROOMS. Detailed description of these datasets and the test environment can be found in Appendix A. It has to be noted that T20I6D100K is a sparse, weakly correlated dataset imitating market basket data, while the other two datasets are dense and highly correlated. Table B.5 contains additional information about the proportion of the number of FCIs to the number of FIs. It can be seen that weakly correlated data contain few frequent itemsets, even at low minimum support values, and almost all frequent itemsets are closed. On the contrary, in the case of highly correlated data the difference between the number of frequent itemsets and frequent closed itemsets is significant.

4.6.1 Number of Rules

Table 4.5 shows the following information: minimum support and confidence; number of all association rules; number of closed rules; number of rules in the generic basis; number of rules in the informative basis; number of rules in the reduced informative basis; number of minimal non-redundant association rules; number of reduced minimal non-redundant association rules. We attempted to choose significant *min_supp* and *min_conf* thresholds as observed in other papers for similar experiments.

In T20I6D100K almost all frequent itemsets are closed, thus the number of all rules and the number of closed association rules is almost equal. For the other two datasets that are dense and highly correlated, the reduction of the number of rules in the Closed Rules is considerable.

From the family of minimal non-redundant association rules, the most significant sets are $\mathcal{MNR}$ and $\mathcal{RMNR}$, because they are lossless, sound and informative representations of *all* association rules [Kry02]. The size of the $\mathcal{MNR}$ set is almost equal to the size of $\mathcal{AR}$ in sparse datasets, but in dense datasets $\mathcal{MNR}$ produces much less rules. Removing the transitive relations, the number of rules can be further reduced in all cases ($\mathcal{RMNR}$).

dataset (min_supp)	min_conf	# of rules ($\mathcal{CR}$)	# of trie acc. − cache	time − cache	# of trie acc. + cache	time + cache
T20I6D100K (0.5%)	90%	726,459	10,465	164.48	2,034	120.30
	70%	956,083	11,017	201.15	2,068	152.31
	50%	1,044,086	11,502	216.20	2,099	167.07
	30%	1,073,114	11,559	222.59	2,099	170.06
C20D10K (30%)	90%	47,289	80,719	16.82	41,784	12.49
	70%	91,953	148,884	31.79	63,451	21.10
	50%	114,245	179,343	38.43	67,224	24.24
	30%	138,750	216,014	45.89	70,161	27.36
MUSHROOMS (30%)	90%	5,571	8,306	2.23	3,941	1.49
	70%	11,709	17,219	4.57	5,850	2.44
	50%	16,306	23,777	6.36	6,239	2.98
	30%	20,120	28,786	7.63	6,457	3.31

Table 4.3: Generating $\mathcal{CR}$ without (−) and with (+) cache.

dataset (min_supp)	min_conf	# of rules ($\mathcal{MNR}$)	# of trie acc. − cache	time − cache	# of trie acc. + cache	time + cache
T20I6D100K (0.5%)	90%	721,948	1,814	404.22	525	394.14
	70%	951,572	2,208	436.99	593	428.59
	50%	1,039,575	2,528	451.93	644	441.52
	30%	1,068,603	2,568	457.89	653	449.47
C20D10K (30%)	90%	9,221	436	1.78	54	1.68
	70%	19,866	567	2.85	94	2.77
	50%	25,525	1,298	3.48	228	3.35
	30%	31,775	2,497	4.24	579	4.04
MUSHROOMS (30%)	90%	1,496	295	0.67	67	0.54
	70%	3,505	1,119	0.95	288	0.78
	50%	5,226	1,942	1.34	590	1.00
	30%	7,115	2,893	1.63	1,219	1.28

Table 4.4: Generating $\mathcal{MNR}$ without (−) and with (+) cache.

4.6.2 Execution Times of Rule Generation

Figure 4.6 shows for each dataset the execution times of the computation of all, closed and minimal non-redundant association rules. For the extraction of the necessary itemsets we used the multifunctional *Zart* algorithm (Section 3.3.1) that can generate all kinds of association rules indicated in Table 4.5. Figure 4.6 does not include the extraction time of itemsets, it only shows the time of rule generation.

Comparing $\mathcal{AR}$ and $\mathcal{CR}$

For datasets with much less frequent closed itemsets (C20D10K, MUSHROOMS), the generation of closed rules is more efficient than finding all association rules. As seen before, we need to look up the closed supersets of frequent itemsets very often when extracting closed rules. For this procedure we use the trie data structure that shows its advantage on dense, highly correlated

dataset (min_supp)	min_conf	$\mathcal{AR}$	$\mathcal{CR}$	$\mathcal{GB}$	$\mathcal{IB}$	$\mathcal{RIB}$	$\mathcal{MNR}$ ($\mathcal{GB} \cup \mathcal{IB}$)	$\mathcal{RMNR}$ ($\mathcal{GB} \cup \mathcal{RIB}$)
D (40%)	50%	50	30	8	17	13	25	21
	90%	752,715	726,459		721,716	91,422	721,948	91,654
T20I6D100K	70%	986,058	956,083	232	951,340	98,097	951,572	98,329
(0.5%)	50%	1,076,555	1,044,086		1,039,343	101,360	1,039,575	101,592
	30%	1,107,258	1,073,114		1,068,371	102,980	1,068,603	103,212
	90%	140,651	47,289		8,254	2,784	9,221	3,751
C20D10K	70%	248,105	91,953	967	18,899	3,682	19,866	4,649
(30%)	50%	297,741	114,245		24,558	3,789	25,525	4,756
	30%	386,252	138,750		30,808	4,073	31,775	5,040
	90%	20,453	5,571		952	682	1,496	1,226
MUSHROOMS	70%	45,147	11,709	544	2,961	1,221	3,505	1,765
(30%)	50%	64,179	16,306		4,682	1,481	5,226	2,025
	30%	78,888	21,120		6,571	1,578	7,115	2,122

Table 4.5: Comparing sizes of different sets of association rules.

dataset (min_supp)	min_conf	$\mathcal{AR}$	$\mathcal{CR}$	$\mathcal{MNR}$
	90%	114.43	120.30	394.14
T20I6D100K	70%	147.69	152.31	428.59
(0.5%)	50%	165.48	167.07	441.52
	30%	169.66	170.06	449.47
	90%	15.72	12.49	1.68
C20D10K	70%	26.98	21.10	2.77
(30%)	50%	34.74	24.24	3.35
	30%	41.40	27.36	4.04
	90%	1.93	1.49	0.54
MUSHROOMS	70%	3.99	2.44	0.78
(30%)	50%	5.63	2.98	1.00
	30%	6.75	3.31	1.28

Table 4.6: Execution times of rule generation.

datasets. On the contrary, when almost all frequent itemsets are closed (T20I6D100K), the high number of superset operations cause that all association rules can be extracted faster.

Comparing $\mathcal{CR}$ and $\mathcal{MNR}$

As we have seen, $\mathcal{CR}$ is a maximal set of closed association rules, i.e. it contains *all* closed association rules. As a consequence, we cannot say that this basis is minimal, or non-redundant, but by all means it is a smaller set than $\mathcal{AR}$, especially in the case of dense, highly correlated datasets. Moreover, $\mathcal{CR}$ is a framework for some other bases. For instance, minimal non-redundant association rules are also closed association rules, since by definition the union of the antecedent and the consequent of such a rule forms a frequent closed itemset. Thus, $\mathcal{MNR}$ is a special subset of $\mathcal{CR}$, which could also be defined the following way:

Definition 4.8 *Let CR be the set of Closed Rules. The set of minimal non-redundant association* *rules is:*

$$\mathcal{MNR} = \{r : P_1 \rightarrow P_2 \mid r \in CR \wedge P_1 \textit{ is a frequent generator}\} \ .$$

This is equivalent to the following definition:

$$\mathcal{MNR} = \{r : P_1 \rightarrow P_2 \mid (P_1 \cup P_2) \in FC \wedge P_1 \textit{ is a minimal generator}\} \ ,$$

where FC stands for the set of frequent closed itemsets.

Experimental results show that $\mathcal{CR}$ can be generated more efficiently than $\mathcal{MNR}$ on sparse datasets. However, on dense datasets $\mathcal{MNR}$ can be extracted much more efficiently.

4.7 Related Work

Beside $\mathcal{CR}$ and $\mathcal{MNR}$, there are several other concise representations of association rules proposed, e.g. representative rules ($\mathcal{RR}$) [Kry98], Duquennes-Guigues basis ($\mathcal{DG}$) [GD86], Luxenburger basis ($\mathcal{LB}$) [Lux91], proper basis ($\mathcal{PB}$), structural basis ($\mathcal{SB}$) [PBTL99a], etc. A very good comparative study of these bases can be found in [Kry02].

All valid exact association rules can be derived from $\mathcal{DG}$, and all valid approximate association rules can be derived from $\mathcal{LB}$. The exact association rules are derived from $\mathcal{DG}$ by applying Armstrong's axioms. $\mathcal{RR}$ is a subset of $\mathcal{MNR}$, but unlike $\mathcal{MNR}$, this rule representation is not informative (although it gives a pessimistic estimation of support and confidence values).

4.8 Conclusion

In this chapter we presented a new basis for association rules called Closed Rules ($\mathcal{CR}$). This basis contains all valid association rules that can be generated from frequent closed itemsets. $\mathcal{CR}$ is a lossless representation of all association rules. Regarding the number of rules, our basis is between all association rules ($\mathcal{AR}$) and minimal non-redundant association rules ($\mathcal{MNR}$), filling a gap between them. The new basis provides a framework for some other bases. We have shown that $\mathcal{MNR}$ is a subset of $\mathcal{CR}$. The number of extracted rules is less than the number of all rules, especially in the case of dense, highly correlated data when the number of frequent itemsets is much more than the number of frequent closed itemsets. $\mathcal{CR}$ contains more rules than $\mathcal{MNR}$, but for the extraction of closed association rules we *only* need frequent closed itemsets, nothing else. On the contrary, the extraction of minimal non-redundant association rules needs much more computation since frequent generators also have to be extracted and assigned to their closures.

As a summary, we can say that $\mathcal{CR}$ is a good alternative for all association rules. The number of generated rules can be much less, and beside frequent closed itemsets nothing else is required.

Chapter 5

Rare Itemsets and Rare Association Rules

In this chapter we address the extraction of rare itemsets and the generation of rare association rules. This is one of the most important part of this thesis work. Until now, studies in data mining have mainly concentrated on frequent itemsets and generation of association rules from them. The first algorithm to find frequent itemsets was *Apriori*, which has been followed by numerous other algorithms. Most of these algorithms are extensions and optimizations of *Apriori*. Their common property is that they all extract frequent itemsets or a subset of frequent itemsets (frequent closed itemsets, maximal frequent itemsets, frequent generators). In this chapter we investigate the complement of frequent itemsets, namely the rare (or non-frequent) itemsets. In the literature, the problem of rare itemset mining and the generation of rare association rules has not yet been studied in detail, though such itemsets also contain important information just as frequent itemsets do. A particularly relevant field for rare itemsets is medical diagnosis. For example it may be that in a large group of patients diagnosed with the same sickness, a few patients exhibit unusual symptoms. It is important for the doctor to take this fact into consideration.

The chapter is organized as follows. In Section 5.1 we show how to extract all rare itemsets from a dataset. This section is based on [SMP$^+$06]. In Section 5.2 we first define the *border* between frequent and rare itemsets, and then study the properties of the border. Section 5.3 details how to generate rare association rules from rare itemsets.

5.1 Rare Itemsets

As we noted before, research has concentrated on finding *frequent* association rules (i.e. association rules with sufficiently high support and confidence). To find them, frequent itemsets from the dataset must be extracted first. The problem of frequent itemset extraction was initially a sub-problem of association rule mining [AMS$^+$96], but later it turned out to be useful also in different areas, such as sequential pattern mining [AS95], spatial association rule mining [KH95], , cyclic association rule mining [ORS98], , negative association rule mining [SON98], , frequent closed itemset search [PBTL99c, STB$^+$02, ZH02, WHP03], maximal frequent itemset search [Bay98, LK98, AAP00, GZ01], discovery of disjunction-free sets [BR01], etc.

5.1.1 Contribution and Motivations

We present a new method for finding rare itemsets. The first part of our method identifies a minimal generator set called *minimal rare itemsets.* In the second part, these itemsets are used to restore all rare itemsets. To the best of our knowledge there is no algorithm yet designed specifically for extracting all rare itemsets.

The discovery of rare itemsets may be particularly useful in medicine and biology. We describe hereafter some potential applications of rare itemset extraction in these domains. For example, we may be interested in identifying the cause of cardiovascular diseases (`CVD`) for a given database of medical records. A frequent association rule such as "{`elevated cholesterol level`} $\Rightarrow$ {`CVD`}" may validate the hypothesis that people having a high cholesterol level are at high risk for `CVD`. On the other hand, if we have sufficiently high number of vegetarian people in our database, then a rare association rule "{`vegetarian`} $\Rightarrow$ {`CVD`}" may validate the hypothesis that vegetarian people have a low `CVD` risk. In this case, the itemsets {`vegetarian`} and {`CVD`} are both frequent, but the itemset {`vegetarian`, `CVD`} is rare. To take another example, this time from the field of pharmacovigilance (a field of pharmacology dedicated to the detection, survey and study of adverse drug effects), given a database of adverse drug effects, rare itemset extraction enables a more efficient way to associate drugs with adverse effects. Thus, withdrawal or continuance of a given drug can be decided (e.g. the lipid-lowering drug cerivastatin was withdrawn in August 2001), and thereby fatal accidents may be avoided. Finally, as a third example, rare itemset extraction may be useful for the data mining of healthy cohorts. For example, the real-world healthy cohort STANISLAS cohort [SVH+98, MNSVS05a] is composed of a thousand presumably healthy French families. Its main objective is to investigate the impact of genetic and environmental factors on diversity in cardiovascular risk factors. Interesting information to extract from this database for the expert of the domain includes the profiles which associate genetic data with extreme or borderline values of biological parameters. However, such types of associations are rather rare in healthy cohorts. That is the reason why rare itemset extraction could be very helpful to the expert.

5.1.2 Basic Concepts

Below we present the basic concepts of rare itemsets. We use definitions that were given in Section 3.1.

Definition 5.1 (rare itemset) *An itemset P is called* rare *(or* not frequent*) if its support is not more than a given* maximum support *(denoted by* max_supp*), i.e. $supp(P) \leq max\_supp$.*

This definition allows an interval between *min_supp* and *max_supp.* We work with a special case, i.e. no interval between *max_supp* and *min_supp.* That is, we consider an itemset to be rare if it is not frequent. It means one single border between frequent and rare itemsets. Section 5.2 contains a detailed study of this border. In the case of a single border, knowing the *min_supp* value, *max_supp* can be easily computed. If *min_supp* is given in absolute value, then $max\_supp = min\_supp - 1$. If *min_supp* is given in relative value, then $max\_supp = min\_supp - 1/|O|$, where $|O|$ is the number of objects in the dataset. The task of rare itemset mining consists of generating all itemsets (with their supports) with supports less than or equal to a specified threshold *max_supp.*

5.1.3 A Lattice-Based Approach for Itemset Enumeration

Before presenting our algorithms for finding rare itemsets, we investigate our method from a lattice-based point of view. Figure 5.1 shows the powerset lattice $P(D)$ of the set of items in our example database D (Table 3.1). The set of all rare itemsets forms a join semilattice since it is closed under the join operation, i.e., for any rare itemsets X and Y, $X \cup Y$ is also rare. On the other hand, it does not form a meet semilattice, since if X and Y are rare, it does not imply $X \cap Y$ is rare. Note that frequent itemsets form a meet semilattice, i.e. for any two frequent itemsets, X and Y, $X \cap Y$ is also frequent. In the examples we define $min\_supp = 3$, which means that $max\_supp = 2$.

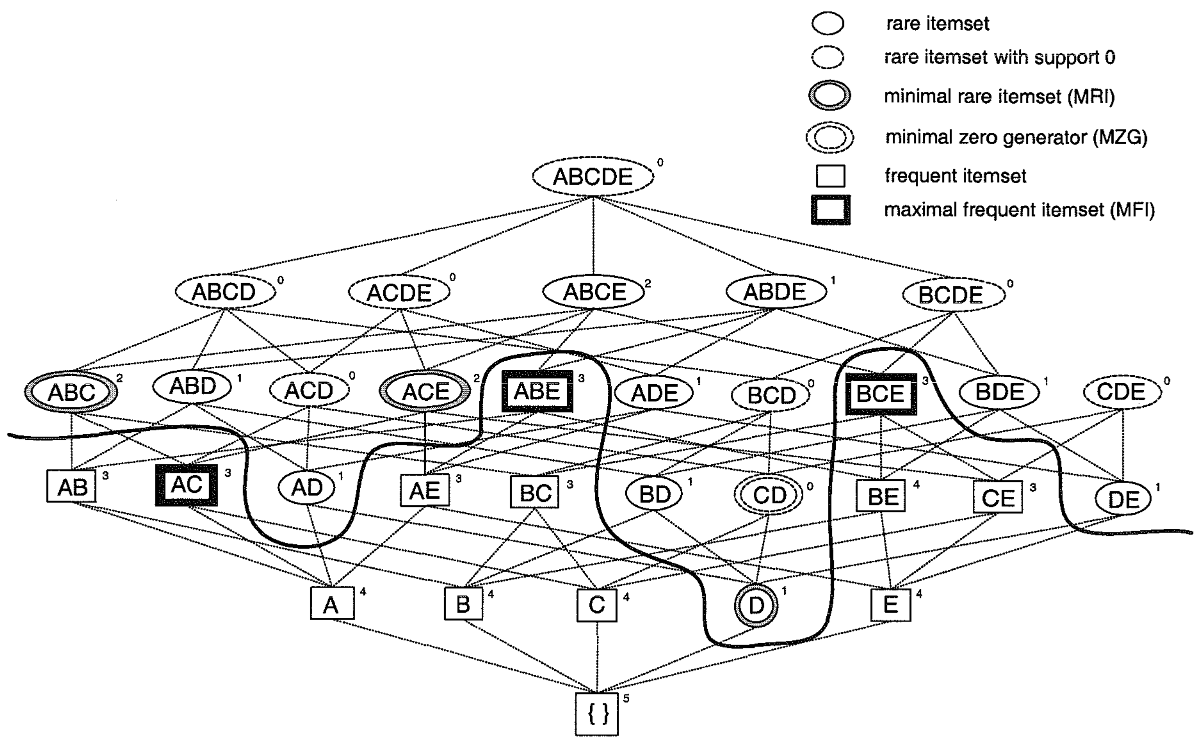


Figure 5.1: Powerset lattice of dataset D (Table 3.1).

Itemsets can be grouped in two sets: frequent and rare itemsets. Between the two sets a border can be drawn (see Section 5.2 for more details on the concept of border). At the bottom of the lattice we find the smallest itemset, the empty set. At each level there are itemsets of the same length. At the top of the lattice we find the largest itemset, which contains all the attributes. In Figure 5.1, support values are indicated in the top right-hand corner of itemsets.

Both sets (frequent and rare itemsets) have a minimal generator subset.[24] In the case of frequent itemsets this subset is called maximal frequent itemsets. Here we recall the definition of MFIs from Section 3.1:

An itemset is called a maximal frequent itemset *(MFI) if it is frequent (and thus all its*

[24]Under a "generator set" of frequent (resp. rare) itemsets we mean here a set of itemsets from which all other frequent (resp. rare) itemsets can be restored.

subsets are frequent), and all its proper supersets are not frequent.[25]

These itemsets are called *maximal* because they have no frequent supersets. On the other hand, the set of these itemsets is a *minimal* set, i.e. the smallest set from which all frequent itemsets can be restored.[26]

As the complementary of MFIs, we can define the minimal rare itemsets in the following way.

Definition 5.2 (minimal rare itemset) *An itemset is called a minimal rare itemset (*MRI*) if it is rare (and thus all its supersets are rare), and all its proper subsets are not rare.*

Note that *minimality* is used here in the sense that MRIs have no proper rare subsets, i.e. all their proper subsets are on the other (positive) side of the border.

This definition can be rewritten formally in the following way. Recall that A denotes the set of attributes in a dataset. The set of minimal rare itemsets is:

$$MRI = \{X \subseteq A \mid \forall Y \subset X,\ supp(X) < min\_supp \leq supp(Y)\} \ .$$

Such itemsets form a minimal set, i.e. the smallest set from which all rare itemsets can be restored. For restoring all frequent itemsets, we first take all possible subsets of MFIs, and then with one database pass count their supports. Similarly, for restoring all rare itemsets, we first generate all possible supersets of MRIs, and then with one database pass count their supports.

Rare itemsets can be distinguished into two kinds of subsets: **(a)** rare itemsets with support 0, and **(b)** rare itemsets with support greater than 0. This distinction is important, because the total number of rare itemsets is usually very high, and thus we are more interested in working with rare itemsets whose support exceeds 0.

Definition 5.3 *An itemset is called a* zero itemset *if its support is 0, otherwise it is called a* non-zero itemset.[27]

In this part of our work we concentrate on finding non-zero rare itemsets. However, it must be noted that zero itemsets can also be interesting in some cases. For instance, a zero itemset Z provides the information that items of Z *never* occur together in the database.

In the case of all rare itemsets, we have already described the minimal rare itemsets. For zero itemsets, a similar minimal generator subset can also be defined:

Definition 5.4 *An itemset is a* minimal zero generator *(MZG), if it is a zero itemset (thus all its supersets are zero itemsets), and all its proper subsets are non-zero itemsets.*

In Figure 5.1 there is one minimal zero generator: {CD}. MZGs are a lossless condensed representation of zero itemsets, i.e. from MZGs all zero itemsets can be restored with their proper support values (which are always 0). For this we only need to generate all possible supersets of MZGs using the attributes of the dataset.

As mentioned before, we do not restore zero itemsets because of their high number. To avoid zero itemsets, we will use MZGs. The second part of our method restores all non-zero rare itemsets from MRIs in a levelwise manner. If a candidate has an MZG subset, then this candidate is surely a zero itemset, thus it can be pruned. That is, using the MZGs, we can reduce the search space during the restoration of all rare itemsets.

[25]Note that maximal frequent itemsets are closed itemsets ($MFI \subseteq FCI \subseteq FI$).

[26]They could also be called *largest* frequent itemsets, because they do not have a larger frequent superset.

[27]The concept of "zero itemset" is not to be confused with the concept of "empty set".

Proposition 5.1 *All zero itemsets are in the same unique equivalence class called the* zero equivalence class. *If this class is not empty, then (1) the generators (see Def. 3.2) of this class are the minimal zero generators (MZGs); and (2) the closure of this class is the* largest itemset, *i.e. the itemset that contains all the attributes of the dataset.*

Proof.
(1) If there exist a zero itemset, then the largest itemset is a zero itemset as well by the anti-monotonocity property (see Property 3.2). Thus, a zero itemset is a subset of the largest itemset, and since their support is the same, by Lemma 3.2 they are in the same equivalence class. That is, all zero itemsets are in the same unique equivalence class whose closure is the largest itemset.
(2) Minimal zero generators (MZGs) are the generators of the zero equivalence class because by Def. 5.4 they have no proper subset with the same support. □

Corollary 5.1 *If an object includes all the attributes of the dataset*[28]*, then there exist no zero itemsets. As a consequence, there are no minimal zero generators, and there is no zero equivalence class either.*

5.1.4 Finding Rare Itemsets

In this section we present the two parts of our method for finding rare itemsets. The first part only finds minimal rare itemsets, while the second part restores all non-zero rare itemsets from the set of minimal rare itemsets.

Finding Minimal Rare Itemsets

It may be surprising, but the easiest way to find minimal rare itemsets is to use the well-known *Apriori* algorithm [MT96]. *Apriori* is based on two principles (see Properties 3.1 and 3.2). *Apriori* is designed to find all frequent itemsets, but as a "side effect" it also explores the minimal rare itemsets. When *Apriori* finds a rare itemset, it will not generate later any of its supersets because they are surely not frequent. Since *Apriori* explores the itemset lattice level by level from the bottom up, it will count the support of the minimal rare itemsets. These itemsets are pruned, and later the algorithm notices if a candidate has a minimal rare subset (actually *Apriori* checks if all $(k-1)$-long subsets of a k-candidate are frequent. If one of them is not frequent, then the candidate is surely rare. But it also means that the candidate has a minimal rare subset). Thanks to this pruning technique, *Apriori* can significantly reduce the search space in the itemset lattice.

In order to save minimal rare itemsets, *Apriori* needs just a slight modification. If the support of a candidate is less than the minimum support, then instead of deleting it we will save it in the set of minimal rare itemsets (see *Apriori-Rare*, Algorithm 15).

`SupportCount` method: counts the support of the candidate itemsets. This requires one database pass.

`Apriori-Gen` function: using frequent k-long itemsets it generates potentially frequent $(k+1)$-long candidates. Potentially frequent means that the candidates have no rare subset, i.e. they have no minimal rare subset. Including a rare itemset means being rare (by Property 3.2). For a detailed description of this function see Algorithm 23.

[28]In other words, the input dataset –considered as a binary matrix– contains a full line (a line full of 1s).

```
Algorithm 15 (Apriori-Rare):

Description:  modification of Apriori to find minimal rare itemsets (MRIs)
Input:        dataset + min_supp
Output:       all frequent itemsets + minimal rare itemsets

 1)  $C_1 \leftarrow \{$1-itemsets$\}$;
 2)  $i \leftarrow 1$;
 3)  while $(C_i \neq \emptyset)$
 4)  {
 5)      SupportCount$(C_i)$;
 6)      $R_i \leftarrow \{r \in C_i \mid \text{support}(r) < min\_supp\}$; // $R$ – for rare itemsets
 7)      $F_i \leftarrow \{f \in C_i \mid \text{support}(f) \geq min\_supp\}$; // $F$ – for frequent itemsets
 8)      $C_{i+1} \leftarrow$ Apriori-Gen$(F_i)$; // $C$ – for candidates
 9)      $i \leftarrow i + 1$;
10)  }
11)  $I_{MR} \leftarrow \bigcup R_i$; // minimal rare itemsets
12)  $I_F \leftarrow \bigcup F_i$; // frequent itemsets
```

The execution of the algorithm on dataset D (Table 3.1) with minimum support 3 (60%)[29] is illustrated in Table 8.4. Taking the union of the R_i tables the algorithm finds the minimal rare itemsets (collected in Table 5.2).

In the next subsection we show how to reconstruct the supersets of MRIs (i.e. how to reconstruct all rare itemsets) avoiding zero itemsets.

Restoring All Rare Itemsets

All rare itemsets are restored from minimal rare itemsets. For this we need to generate all possible supersets of MRIs. The minimal zero generators are used to filter zero itemsets during the generation of supersets. This way the search space can be significantly reduced. In this subsection we present a prototype algorithm for this task called *Arima*[30] (A Rare Itemset Miner Algorithm, see Algorithm 16).

`SupportCount` method: counts the support of the candidate itemsets. This requires one database pass.

The execution of the algorithm on dataset D (Table 3.1) with minimum support 3 (60%)[31] is illustrated in Table 5.3.

The algorithm first takes the smallest MRI, {D}, which is rare, thus it is copied to R_1. Its 2-long supersets are generated and stored in C_2 ({AD}, {BD}, {CD} and {DE}). With one database pass their supports can be counted. Since {CD} is a zero itemset, it is copied to the MZG list. Non-zero itemsets are stored in R_2. For each rare itemset in R_2, all its supersets are generated. For instance, from {AD} we can generate the following candidates: {ABD}, {ACD} and {ADE}. If a candidate has an MZG subset, then the candidate is surely a zero itemset and can be pruned ({ACD}). Potentially non-zero candidates are stored in C_3. Duplicates are not allowed in C_i tables. From MRIs the 3-long itemsets are added to C_3 ({ABC} and {ACE}). The

[29]This is equivalent to maximum support 2 (40%).
[30]Not to be confused with the ARIMA model methodology (Auto Regressive Integrated Moving Average).
[31]This is equivalent to maximum support 2 (40%).

C_1	supp
{A}	4
{B}	4
{C}	4
{D}	1
{E}	4

R_1	supp
{D}	1

F_1	supp
{A}	4
{B}	4
{C}	4
{E}	4

C_2	supp
{AB}	3
{AC}	3
{AE}	3
{BC}	3
{BE}	4
{CE}	3

R_2	supp
$\emptyset$	

F_2	supp
{AB}	3
{AC}	3
{AE}	3
{BC}	3
{BE}	4
{CE}	3

C_3	supp
{ABC}	2
{ABE}	3
{ACE}	2
{BCE}	3

R_3	supp
{ABC}	2
{ACE}	2

F_3	supp
{ABE}	3
{BCE}	3

C_4	supp
$\emptyset$	

Table 5.1: Execution of the *Apriori-Rare* algorithm with $min\_supp = 3$ ($max\_supp = 2$).

set	supp
{D}	1
{ABC}	2
{ACE}	2

Table 5.2: Minimal rare itemsets found in dataset D.

Algorithm 16 (Arima):

Description: restores all (non-zero) RIs from MRIs
Input: dataset + MRIs
Output: all rare itemsets + MZGs

```
1)  $MZG \leftarrow \emptyset$;
2)  $S \leftarrow$ {all attributes in $D$};
3)  $i \leftarrow$ {length of smallest itemset in $MRI$};
4)  $C_i \leftarrow$ {$i$-long itemsets in $MRI$}; // i.e. smallest itemsets in $MRI$
5)  $MZG \leftarrow MZG \cup \{z \in C_i \mid \text{support}(z)= 0\}$;
6)  $R_i \leftarrow \{r \in C_i \mid \text{support}(r) > 0\}$;
7)  while ($R_i \neq \emptyset$)
8)  {
9)      loop over the elements of $R_i$ ($r$)
10)     {
11)         $Cand \leftarrow$ {all possible supersets of $r$ using $S$}; // no duplicates are allowed
12)         loop over the elements of $Cand$ ($c$)
13)         {
14)             if $c$ has a proper subset in $MZG$ (i.e. if $c$ is a superset
                of a min. zero gen.), then delete $c$ from $Cand$;
15)         }
16)         $C_{i+1} \leftarrow C_{i+1} \cup Cand$; // no duplicates are allowed
17)         $Cand \leftarrow \emptyset$; // re-initializing $Cand$
18)     }
19)     SupportCount($C_{i+1}$);
20)     $C_{i+1} \leftarrow C_{i+1} \cup$ {$(i+1)$-long itemsets in $MRI$};
21)     $MZG \leftarrow MZG \cup \{z \in C_{i+1} \mid \text{support}(z)= 0\}$;
22)     $R_{i+1} \leftarrow \{r \in C_{i+1} \mid \text{support}(r) > 0\}$;
23)     $i \leftarrow i + 1$;
24) }
25) $I_R \leftarrow \bigcup R_i$; // rare itemsets
```

algorithm stops when the R_i table is empty. The union of the R_i tables gives all non-zero rare itemsets. At the end we also have all MZGs collected, thus if one needs all zero itemsets, this list can be used to restore them. The process is similar; we would need to generate all possible supersets of MZGs. In our case we are only interested in non-zero itemsets, but it is also possible to work with zero itemsets.

5.2 Border Study

If we consider an itemset to be rare if it is not frequent, then there exists a single border between frequent and rare itemsets in the powerset lattice. Frequent itemsets are said to be on the *positive side* of the border, while rare (or not frequent) itemsets are on the *negative side* of the border. The theory of the border was introduced by Mannila and Toivonen in [MT97]. The border was also studied and discussed by Boulicaut *et al.* in [BBR03, CRB05]. There also exist studies on

$MZG = \emptyset$
S = {A, B, C, D, E}
MRI = {D(1), ABC(2), ACE(2)}
$i = 1$

C_1	supp
{D}	1

$MZG_{before} = \emptyset$
$MZG_{after} = \emptyset$

R_1	supp
{D}	1

C_2	supp
{AD}	1
{BD}	1
{CD}	0
{DE}	1

$MZG_{before} = \emptyset$
MZG_{after} ={CD}

R_2	supp
{AD}	1
{BD}	1
{DE}	1

C_3	supp
{ABD}	1
{ADE}	1
{BDE}	1
{ABC}	2
{ACE}	2

MZG_{before} ={CD}
MZG_{after} ={CD}

R_3	supp
{ABD}	1
{ADE}	1
{BDE}	1
{ABC}	2
{ACE}	2

C_4	supp
{ABDE}	1
{ABCE}	2

MZG_{before} ={CD}
MZG_{after} ={CD}

R_4	supp
{ABDE}	1
{ABCE}	2

C_5	supp
$\emptyset$	

MZG_{before} ={CD}
MZG_{after} ={CD}

R_5	supp
$\emptyset$	

Table 5.3: Execution of the *Arima* algorithm with $min\_supp = 3$ ($max\_supp = 2$).

the complexity of finding maximal frequent and minimal infrequent itemsets [BGKM02, STT98].

Given an arbitrary dataset and a user-defined minimum support threshold, let $\mathcal{F}$ be the set of frequent itemsets in the dataset.

Definition 5.5 (positive border) *The positive border of frequent itemsets (or simply* positive border*) is a set that consists of frequent itemsets such that all their proper supersets are not frequent. The positive border is denoted by* $Bd^+(\mathcal{F})$.

Definition 5.6 (negative border) *The negative border of frequent itemsets (or simply* negative border*) is a set that consists of not frequent itemsets such that all their proper subsets are frequent. The negative border is denoted by* $Bd^-(\mathcal{F})$.

From the definitions above, it is clear that the positive border is equivalent to the set of maximal frequent itemsets (see Def. 5.1), and the negative border is equivalent to the set of minimal rare itemsets (see Def. 5.2).

Definition 5.7 (border) *The border of frequent itemsets (or simply* border*) is the union of the positive and negative borders, i.e.* $Bd(\mathcal{F}) = Bd^+(\mathcal{F}) \cup Bd^-(\mathcal{F})$.

Example. Consider the dataset D (Table 3.1) with $min\_supp = 3$ (see Figure 5.1). Then
$\mathcal{F} = \{\emptyset, A, B, C, E, AB, AC, AE, BC, BE, CE, ABE, BCE\}$,
$Bd^+(\mathcal{F}) = \{AC, ABE, BCE\}$,
$Bd^-(\mathcal{F}) = \{D, ABC, ACE\}$. □

Table 5.4 contains some statistics about border sizes. It shows the name of the database, the minimum support values, the number of frequent itemsets, the size of the positive border and the size of the negative border. As can be seen, the size of the negative border can be significantly larger than the positive border, especially in the case of sparse datasets.

By [MT96], the negative border defines which sets are frequent and which are not. Given a set X, if X is a superset of some set in the negative border, then X is not frequent, otherwise it is frequent (by Property 3.2).

Note that its dual is also true, i.e. the positive border also defines which sets are frequent and which are not. Given a set Y, if Y is a subset of some set in the positive border, then Y is frequent, otherwise it is not frequent (by Property 3.1).

In [MT96] it is also stated that "algorithms that compute the collection of frequent sets also compute the frequencies of sets in the negative border, as the negative border consists of exactly those sets which, on the basis of other information, could be frequent, and whose frequency should therefore be checked". In Section 5.1 we have used *Apriori* to compute the negative border (*Apriori-Rare*), but this idea could be applied to other levelwise algorithms too, e.g. to *Pascal* or to *Zart*.

5.2.1 Minimal Rare Generators

Definition 5.8 (minimal rare generator) *An itemset is called a* minimal rare generator *(MRG) if it is a rare generator, and all its proper subsets are not rare.*

Note that *minimality* is used here in the sense that MRGs have no proper rare subsets, i.e. all their proper subsets are on the other (positive) side of the border.

This definition can be rewritten formally in the following way. Let G be the set of generators. The set of minimal rare generators is:

$$MRG = \{X \in G \mid \forall Y \subset X,\ supp(X) < min\_supp \leq supp(Y)\}\ .$$

Corollary 5.2 *All proper supersets of an MRG are rare (by Property 3.2).*

Corollary 5.3 *All proper subsets of an MRG are frequent generators (by Property 3.4).*

Proposition 5.2 *An itemset is a minimal rare itemset if and only if it is a minimal rare generator.*

Proof.
(1) *An MRI is an MRG.* We prove it by contradiction. A generator, by definition, has no proper subset with the same support. Suppose that an MRI is not a generator. In this case, it has a proper subset with the same support. Since MRIs are rare, this subset is also rare, but by definition all proper subsets of an MRI are frequent. This conclusion contradicts the assumption.
(2) *An MRG is an MRI.* The proof follows by Def. 5.8 and Corollaries 5.2 and 5.3. □

By Def. 5.8 and Prop. 5.2 it can be seen that MRGs (and thus MRIs) have a *double minimality.* First, they are minimal *globally*, i.e. among rare itemsets because they have no proper rare subsets. Secondly, they are minimal *locally*, i.e. in their equivalence classes because they have no proper subsets with the same support.

Generators Representation

Kryszkiewicz has shown in [Kry01] that generators can constitute a concise lossless representation of frequent itemsets. Minimal rare generators are called *negative generator border* in her work, denoted by GBd^-. Kryszkiewicz's *generators representation* requires:

- frequent generators (FGs) with their support values
- minimal rare generators (MRGs)
- the set I_D of items that occur in the given database.

Given an arbitrary itemset X, it can be decided with the generators representation whether X is frequent or not. If it is frequent then its proper support value can be derived. The method is the following:

- if $\neg(X \subseteq I_D)$ or $(\exists Z \in MRG \mid Z \subseteq X)$, then X is not frequent;
- else X is frequent and $supp(X) = min(\{supp(Y)\} \mid Y \in FG \wedge Y \subseteq X)$.

Example. Let us consider dataset D (Table 3.1) with $min\_supp = 3$ (see Figure 5.1). Then
$FG = \{A, B, C, E, AB, AC, AE, BC, CE\}$,
$MRG = \{D, ABC, ACE\}$.
The itemset $ABDE$ has an MRG subset, D, thus $ABDE$ is not frequent. The itemset ABE has no MRG subset, thus it is frequent. Its frequent generator subsets are: A, B, E, AB and AE, of which AE (or AB) has the smallest support, i.e. $supp(ABE) = supp(AE) = 3$. $\square$

Comparing the Generators Representation with the Frequent Closed Itemset Representation

Previously, we have presented another concise lossless representation of frequent itemsets, namely the frequent closed itemsets representation (see Property 4.2). In our work we have used the FCI representation because of the following reasons. First, this representation only requires the FCIs. Secondly, the FCI representation is *much more* condensed than the generators representation (see Appendix F) because of the following reasons.

The number of FCIs is less than or equal to the number of frequent generators (by the definition of equivalence classes), i.e. $|FCI| \leq |FG|$. The generators representation requires the MRGs too, and the number of MRGs can be quite high (see Table 5.4 for a comparison). If MRGs are added to FGs, we get the following: $|FCI| \ll |FG| + |MRG|$.

dataset	min_supp	$\|\mathcal{F}\|$	$\|Bd^+(\mathcal{F})\|$	$\|Bd^-(\mathcal{F})\|$
T20I6D100K	10%	7	7	907
	5%	99	99	5,645
	2%	378	365	68,411
	1%	1,534	914	169,819
	0.75%	4,710	1,791	211,578
	0.5%	26,836	4,520	268,915
	0.25%	155,163	12,467	537,765
T25I10D10K	10%	20	20	1,099
	5%	142	142	10,798
	2%	533	492	106,931
	1%	2,893	1,979	223,262
	0.75%	17,073	4,230	279,363
	0.5%	302,284	11,990	413,827
	0.25%	2,055,355	33,946	1,190,931
C20D10K	10%	89,883	152	901
	5%	352,611	388	2,002
	2%	1,741,883	1,589	7,735
	1%	6,194,967	3,794	18,666
	0.75%	9,004,043	5,417	24,641
	0.5%	15,602,883	8,846	37,816
	0.25%	40,450,371	18,604	69,104
C73D10K	95%	1,007	18	1,622
	90%	13,463	81	1,701
	85%	46,575	45	1,652
	80%	109,159	137	1,802
	75%	235,271	181	1,939
	70%	572,087	588	2,727
	65%	1,544,691	988	3,675
MUSHROOMS	50%	163	17	147
	40%	505	39	254
	30%	2,587	89	409
	20%	53,337	245	1,004
	10%	600,817	895	3,077
	5%	4,137,547	2,475	8,806
	1%	92,894,869	10,563	41,557

Table 5.4: Border sizes.

5.3 Rare Association Rules

5.3.1 Introduction

Conventional data mining techniques are only designed to find frequent itemsets in a database. Although it is very useful when one wants to find regularities in a database to help the prediction task, in certain cases one may be more interested in irregularities. As frequent itemsets are usually consistent with the expectations of experts, a large number of frequent association rules with high confidence and support are not interesting. On the other hand, rare itemsets are unknown, unexpected or contradictory to what the user believes. Therefore, they are novel and potentially more interesting for the user than frequent itemsets [LLFH99].

Rare cases are often of special interest. This is especially true in the context of data mining, where one often wants to uncover patterns that may be hidden in massive amounts of data. Rare cases deserve special attention because they address significant problems for data mining algorithms. Rare cases are often more interesting than typical ones [Wei04]. For instance, in medical diagnosis, unusual symptoms often help research more than regular or expected symptoms.

In order to find such rare rules using conventional frequent itemset mining algorithms like *Apriori*, the minimum support must be set very low, drastically increasing the runtime of the algorithm. Moreover, when the minimum support is set low, *Apriori* produces a huge number of frequent itemsets. This is also known as the *rare item problem* [LHM99, CDF$^+$01, YHHR03]. If some data occur rarely but on occurring they appear together in high proportion, the support is low and the confidence is high. Though this case is worthy to be discovered, with the existing frequent association rule techniques these rules remain hidden. In theory, frequent itemset mining algorithms are capable of finding rare associations, but in practice they become intractable if the minimum support is set low enough. As these algorithms are inadequate for finding rare associations, new algorithms are needed.

5.3.2 Basic Concepts

Below we present the basic concepts of rare association rules. We rely on definitions given in Section 5.1.2. An association rule is called *rare* if its support is not more than a given *maximum support.* Since we use a single border, it means that a rule is rare if its support is less than a given *minimum support.* A rare association rule r is *valid* if r is confident, i.e. $supp(r) < min\_supp$ and $conf(r) \geq min\_conf$. In the rest of the thesis, by "rare association rules" we mean *valid* rare association rules. In this part of our work we are interested in finding valid rare association rules, i.e. rules with low support and high confidence.

A rare itemset is a *perfectly rare itemset*, if all its subsets are rare itemsets, otherwise it is an *imperfectly rare itemset.* A rule r: $A \rightarrow B$ is a *perfectly rare rule* if $A \cup B$ is a perfectly rare itemset. A rule r: $C \rightarrow D$ is an *imperfectly rare rule* if $C \cup D$ is an imperfectly rare itemset [KR05].[32]

5.3.3 Related Work

Although the rare itemset mining problem has not yet been studied in much detail, there have been some approaches for solving the "rare item problem". We shall discuss the *MSapriori* (Multiple Supports Apriori), the *RSAA* (Relative Support Apriori Algorithm), the *Apriori-Inverse* and the *MIISR* (Mining Interesting Imperfectly Sporadic Rules) approaches.

[32]Note that instead of "rare rules" Koh and Rountree use the terminology "sporadic rules" in [KR05].

With the ***MSapriori*** algorithm, Liu *et al.* [LHM99] try to overcome the rare item problem by changing the definition of minimum support. In their extended model, each item in the database can have a *minimum item support* (MIS) given by the user. This way, the user expresses different support requirements for different rules. Let MIS(i) denote the MIS value of item i. The minimum support of a rule is the lowest MIS value among the items in the rule. The rule $\{a_1, a_2, \ldots, a_k\} \rightarrow \{a_{k+1}, \ldots, a_r\}$ satisfies the minimum support condition if the support of the rule[33] in the data is greater than or equal to min(MIS(a_1),MIS(a_2),...,MIS(a_r)). With minimum itemset supports they can change the minimum support value of rules dynamically to obtain higher minimum supports for rules that only involve frequent items, and lower minimum supports for rules that involve less frequent items. In this way, *MSapriori* tries to solve the problem that arises by adopting the uniform support in *Apriori*. The MIS values are calculated by the following formula:

$$\begin{aligned} \mathrm{MIS}(i) &= \mathrm{M}(i), \text{ if } \mathrm{M}(i)>\mathrm{LS} \\ &= \mathrm{LS}, \text{ otherwise} \\ \mathrm{M(i)} &= \beta \times \mathrm{f}(i), \text{ where } 0 \leq \beta \leq 1 \ . \end{aligned}$$

The user must specify two parameters: **(1)** LS, the lowest minimum item support allowed, and **(2)** β, a parameter that controls how the MIS values for items should be related to their frequencies. The actual frequency of an item, f(i), is expressed in percentage of the database size.[34]

MSapriori is a method to discover the association rules using a different MIS value as the support for each item. Actually, *MSapriori* finds frequent association rules, and since it dynamically changes the minimum support values, it identifies some less frequent rules. However, the actual criterion of discovery is determined by the user's value of β rather than the frequency of each item. Since this method is based on *Apriori*, it has a similar drawback: finding rare association rules requires the use of a low *min_supp* value, and as a result, the number of rules generated can be enormous, with only a small number being significant rare rules.

The ***RSAA*** algorithm [YHHR03] generates rules which involve significant rare itemsets. This technique uses *relative support*: for any dataset, and with the support of item i represented as *supp*(i), relative support is defined[35] as:

$$\begin{aligned} \mathrm{RSupp}\{i_1, i_2, \ldots, i_k\} = \max(& supp(i_1, i_2, \ldots, i_k)/supp(i_1), \\ & supp(i_1, i_2, \ldots, i_k)/supp(i_2), \\ & \ldots, \\ & supp(i_1, i_2, \ldots, i_k)/supp(i_k)) \ . \end{aligned}$$

This algorithm increases the support threshold for items that have low frequency and decreases the support threshold for items that have high frequency. Like *Apriori* and *MSapriori*, *RSAA* is exhaustive in its generation of rules, so it spends time looking for rules which are not rare. If the minimum-allowable relative support value is set close to zero, then *RSAA* takes a similar amount of time to that taken by *Apriori* to generate low-support rules.

The ***Apriori-Inverse*** [KR05] is a levelwise, bottom-up algorithm. It uses two borders. The first threshold *max_supp* is used to find rare itemsets, and the second threshold, the absolute

[33]The definition of the support of a rule is not changed in this model, i.e. $supp(A \rightarrow B) = supp(A \cup B)$, where $A \cup B$ is the itemset that contains all items of A and B.

[34]Example. Consider three items, A, B and C, where f(A)=1%, f(B)=2% and f(C)=10%. If LS=2% and β=0.3, then MIS(A)=2%, MIS(B)=2% and MIS(C)=3%.

[35]This definition of "relative support" is different from ours given in Section 3.1.

minimum support is used to eliminate itemsets that are considered to be *too* rare. First, it identifies 1-long *rare* itemsets and stores them in the table R_1. From this point onwards, it works like *Apriori*. Since R_1 only contains rare itemsets, when the algorithm generates their 2-long, 3-long, etc. supersets, they all will be rare according to the anti-monotonocity property (Property 3.2). Moreover, they all will be *perfectly rare* because all their subsets are rare. This is guaranteed by the fact that R_1 only contains rare items. At the end, taking the union of the R_i tables, the algorithm finds all perfectly rare itemsets. Then, association rules are formed in the usual way.

It is important to note that *Apriori-Inverse* does not find *all* rare itemsets, but only a *subset* of rare itemsets. This subset contains all perfectly rare itemsets, i.e. rare itemsets such that all their subsets are rare. For instance, in dataset D (Table 3.1) by $max\_supp = 3$, *Apriori-Inverse* would only find the itemset {D}, from which no association rule can be generated.

Koh and Rountree note that it would be desirable to be able to generate imperfectly rare rules as well. For this task they propose slight modifications of *Apriori-Inverse* to find at least some imperfectly rare rules: specifically, those that contain subsets that have support just a little higher than max_supp.

Apriori-Inverse is not able to find imperfectly rare rules because it never considers itemsets that have support above max_supp. *Apriori* will miss these rules, because the support of the itemsets forming these rules is low. *Apriori-Inverse* will miss them as well, because the support for the individual items is too high. Therefore, both algorithms will miss rules of the form $AB \rightarrow C$, where A and B are individually frequent, but AB is rare and C is rare. For instance, this is the situation when two symptoms –both of which commonly occur alone– appear together only rarely; but when they do, the combination indicates a rare and serious disease with high confidence. In biomedicine, this type of rule is considered to be very important.

With *Apriori*, the min_supp must be set very low in order to find rare itemsets. This results in drastically increased runtime of the algorithm due to the combinatorial explosion in the number of frequent itemsets. *Apriori-Inverse* suffers from the same problem in reverse: max_supp has to be set so high that it qualifies too many itemsets as rare.

Koh *et al.* in [KRO06] propose a modification of *Apriori-Inverse* called ***MIISR*** to find some imperfectly rare rules using item constraints: they capture rules with a single-item consequent below the max_supp, and then these items are considered to be the only possible consequents for all rules that will be generated. Thus, they generate such rare association rules where the consequent is always a perfectly rare itemset. Though *MIISR* can find some imperfectly rare rules, it has the same problem as *Apriori-Inverse*: in order to find rare itemsets, the max_supp has to be set so high that it qualifies too many itemsets as rare.

5.3.4 Contribution and Motivations

In this part of our work we are interested in finding valid rare association rules, i.e. rules with low support and high confidence. In Section 5.1 we presented a method to restore all rare itemsets from the set of minimal rare itemsets. Once we have all rare itemsets, in theory it is possible to generate all valid rare association rules. However, this method has two drawbacks. First, the restoration of all rare itemsets is a very memory-extensive step due to the huge number of rare itemsets. Secondly, even if we manage to restore all rare itemsets, the number of generated rules would be even more. We should face the same problem as in the case of $\mathcal{AR}$ rules: we have a huge number of rules of which many are redundant and not at all interesting.

Among frequent association rules, there are some special subsets, called bases, from which all other frequent association rules can be restored with a proper inference mechanism. Previously we have seen that the minimal non-redundant association rules ($\mathcal{MNR}$) are a lossless, sound and informative representation of all valid association rules. Moreover, these rules are considered to be very interesting because they allow one to deduce maximum information with a minimal hypothesis.

Our work is motivated by the long-standing open question of devising an efficient algorithm for finding rules that have very high confidence, but for which there is weak support. Thus, the aim of this part of our research is twofold. First, we want to find valid rare association rules efficiently. Secondly, we want to find directly an *interesting* subset of rare association rules, something like the $\mathcal{MNR}$ rules for frequent rules. We present a method that first identifies a minimal set of rare itemsets called minimal rare generators (MRGs). Then, we calculate their corresponding rare equivalence classes, and finally we generate *exact* rare association rules. We call these rules "MRG rules" because their antecedents are minimal rare generators. We demonstrate that MRG rules can always be extracted. Even if the *min_supp* value is set very high, one can find MRG rules with very low support. We also provide a methodology for mining rare association rules.

5.3.5 Finding MRG Association Rules

We became interested in rare association rules while working on a real-life biomedical database called STANISLAS cohort (see Chapter 6 for more details on this database). Our expert was interested in finding relations that are true in the case of a very small number of individuals, say even as small as one or two patients. This required finding association rules with an extremely low support. We also had to face the rare item problem [LHM99]. In order to find rules with very low support, we made several experiments trying to lower the *min_supp* as low as possible in an iterative way. We also realized that conventional frequent itemset mining algorithms have a limit on how low *min_supp* can be set. We call this absolute limit the *barrier.*

The **barrier** *is the* absolute minimum support *value that is still manageable for the given frequent itemset mining algorithm in the given computing environment.*

Naturally, the exact position (value) of the barrier depends on several *computing environment variables*, e.g.:

- the database (size, density, highly- or weakly-correlated, etc.)
- the platform (characteristics of the machine that is used for the calculation (CPU, RAM))
- the software (efficient / less efficient implementation).

For instance, suppose that in one environment one can lower the *min_supp* to 80%. Using the same database on a different platform it may be possible that we are able to descend to 60% or even lower. But the essential point is that with conventional techniques there is *always* a barrier that cannot be crossed, and it is almost certain that we cannot lower the *min_supp* to 1.[36] What is on the other side of the barrier? What kind of information is hidden from us? These are the questions that we pose.

[36]If *min_supp* = 1 (absolute value), then all itemsets are frequent, i.e. by this value one can have all itemsets whose support exceeds 0.

Breaking the Barrier

How could we break the barrier? How to extract interesting association rules from the negative side of the barrier? The key algorithm that can help us is *Apriori-Rare* that we presented in Section 5.1.4 (Algorithm 15). *Apriori* finds frequent itemsets, but as a "side effect" it also explores the so-called minimal rare itemsets (MRIs). *Apriori-Rare* keeps these itemsets instead of deleting them. In Section 5.2.1 we show that minimal rare itemsets are rare generators (Proposition 5.2). The idea is the following: find the closures of these rare generators so as to obtain their equivalence classes. Once rare equivalence classes are explored, it is possible to generate rare association rules in a way very similar to that of finding (frequent) minimal non-redundant association rules (Section 4.3). We call these rules "MRG rules" because their antecedents are minimal rare generators. We group these steps together in an algorithm called *BtB* (Breaking the Barrier).

Note that we are interested in rules whose support is very low but greater than 0. Since *Apriori-Rare* can find MRIs with support 0 too, these itemsets are pruned.

MRG Rules

As we will see later, two kinds of MRG rules can be distinguished, namely exact and approximate rules. In our experiments we are more interested in rare association rules with low support and *very high* confidence. The exact MRG rules can thus be characterized as:

$$r\colon P_1 \Rightarrow P_2 \setminus P_1, \quad \text{where} \quad \begin{array}{l} P_1 \text{ is a minimal rare generator} \\ P_1 \cup (P_2 \setminus P_1) = P_2 \text{ is a rare closed itemset} \\ P_1 \subset P_2 \\ r \text{ is a rare association rule} \\ conf(r) = 1.0 \\ P_1 \text{ is rare} \\ P_2 \setminus P_1 \text{ is rare or frequent.} \end{array}$$

Since a generator is a minimal subset of its closure with the same support, these rules allow us to deduce maximum information with a minimal hypothesis, just as the $\mathcal{MNR}$ rules. Furthermore, the MRG rules are non-redundant, because for rules with the same support and same confidence, these rules contain the most information.

Example. Figure 5.2 shows all the equivalence classes of dataset D (Table 3.1). Support values are depicted above to the right of equivalence classes. Itemsets with the same support are grouped together in the same level. Levels are separated by borders that are defined by different *min_supp* values. Next to each *min_supp* value, the corresponding minimal rare generators are also shown. For instance, if *min_supp* = 4 then there exist 5 frequent itemsets (A, C, B, E, BE) and 6 minimal rare generators (D, AB, AC, AE, BC, CE).

Suppose that the barrier is at *min_supp* = 4. In this case, using *Apriori*, the less frequent association rules have support 4. Our method allows us to find rare association rules *below* this barrier. With *Apriori-Rare*, the following MRGs are found: D, AB, AC, AE, BC and CE. Calculating their closures, four rare equivalence classes are explored, as shown in Table 5.5. Note that *not all* rare equivalence classes are found. For instance, the class whose closure is $ABCE$ is not found because its generators are *not* MRGs, i.e. it is not true for ABC and ACE that all their proper subsets are frequent itemsets.

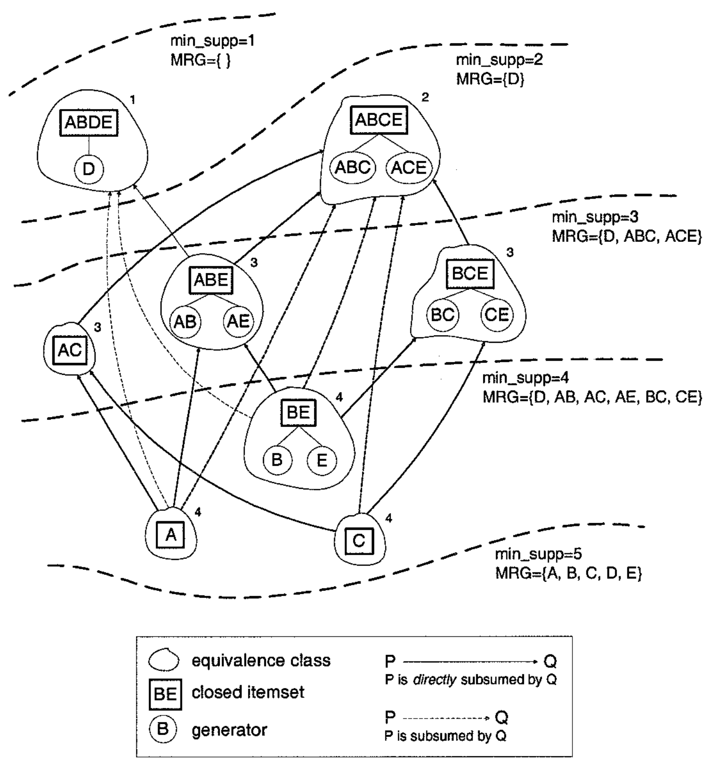


Figure 5.2: Rare equivalence classes found by *BtB* at different *min_supp* values.

closure	support	generators
ABDE	1	D
AC	3	AC
ABE	3	AB, AE
BCE	3	BC, CE

Table 5.5: Rare equivalence classes found by *BtB* at $min\_supp = 4$.

5.3.6 Generating MRG Rules

Once the rare equivalence classes are found (see Table 5.5), the rule generation method is basically the same as in the case of $\mathcal{MNR}$ rules (see Algorithm 12 in Section 4.3). Two kinds of MRG rules can be distinguished, *exact* and *approximate* rules. Whereas approximate rules are generated from two different equivalence classes, exact rules are always extracted within the same equivalence class.

Finding exact MRG rules. So, exact MRG rules are extracted within the same equivalence class. Such rules can only be extracted from complex classes (see Def. 3.4). Table 5.6 shows which exact MRG rules can be extracted from Table 5.5.

Finding approximate MRG rules. Approximate MRG rules are extracted from classes whose closures are comparable with respect to set inclusion. The algorithm is the following: let P_1 be an MRG and let $\gamma(P_1)$ denote the closure of P_1. Find all proper supersets P_2 of $\gamma(P_1)$ among the closures of the found rare equivalence classes. Then add r: $P_1 \to P_2 \setminus P_1$ to the set

rule	support	confidence
$D \Rightarrow ABE$	1	1.0
$AB \Rightarrow E$	3	1.0
$AE \Rightarrow B$	3	1.0
$BC \Rightarrow E$	3	1.0
$CE \Rightarrow B$	3	1.0

Table 5.6: Exact MRG rules found in dataset D (Table 3.1) by $min\_supp = 4$.

rule	support	confidence
$AB \rightarrow DE$	1	$1/3 = 0.33$
$AE \rightarrow BD$	1	$1/3 = 0.33$

Table 5.7: Approximate MRG rules found in dataset D (Table 3.1) by $min\_supp = 4$.

of approximate MRG rules. For instance, Table 5.7 shows which approximate MRG rules can be extracted from Table 5.5.

Note that we are just interested in rare rules that have *high confidence.* As for approximate rules with low confidence, their importance is doubtful. There is a high chance that the itemsets in these rules only appear together by chance. Because of this reason, in our experiments we concentrate rather on *exact* MRG rules.

The Algorithm

Algorithm 17 presents the pseudo code of the *BtB* (Breaking the Barrier) algorithm. *BtB* consists of three main steps. First, we assemble the set of minimal rare generators. For this task we use the *Apriori-Rare* algorithm that finds the set of minimal rare itemsets. Proposition 5.2 says that MRIs are MRGs. Then, the algorithm calculates the closures of MRGs to produce their equivalence classes. The closure of an itemset X is equal to the intersections of the corresponding itemsets of objects that include X. For instance, AB of dataset D (Table 3.1) is included in the 1st, 3rd and 5th objects, thus $\gamma(AB) = ABDE \cap ABCE \cap ABCE = ABE$. Finally, MRG rules are generated from the found rare equivalence classes, as explained before.

Algorithm 17 (BtB):

```
Description:   finds MRG rules

 1)  find the set of minimal rare generators (MRGs);
 2)  find the closures of MRGs;
 3)  generate MRG rules from the found rare equivalence classes;
```

5.3.7 Calculating Other Interestingness Measures

The support and confidence values of the extracted MRG rules can be easily computed, as shown in Tables 5.6 and 5.7. However, to calculate other interestingness measures (see Section 4.4), the support of the *right side* must be known too. There are two possibilities: **(1)** the right side is frequent, or **(2)** the right side is rare.

Frequent right side. The general form of an MRG rule r is: $P_1 \rightarrow P_2 \setminus P_1$, where P_2 is a rare closed itemset. The right side, $P_2 \setminus P_1$, is a subset of P_2, thus it can be frequent. To derive its support, we use the FCI representation of frequent itemsets (see Property 4.1).

Recall that the support of an arbitrary frequent itemset is equal to the support of its smallest frequent closed superset (Property 4.1). This task can be done very efficiently with the trie data structure (see Appendix C.3).

Rare right side. If the right side is rare, there are two further possibilities. Either it is among the rare itemsets that were discovered by *BtB*, or it is not. In the first case its support is known. In the second case, however, its support is not known because it is in an undiscovered rare equivalence class. To find out its support, we are obliged to scan the database.

Note that some association rules may be completely trivial: for instance, when the antecedent is rare but the consequent has support of 100%. Other quality measures can help to find these cases, e.g. the lift. Thus, to filter trivial rules, it is highly recommended to derive the support of the consequents too.

In order to reduce the number of trie and database accesses, we use our cache optimization (see Section 4.5) in all cases.

5.3.8 A Methodology for Finding Rare Association Rules

The algorithm *BtB* is designed to find rare association rules where conventional methods fail. In our experiments, frequent itemset mining algorithms perform quite well with sparse datasets, i.e. the *min_supp* can be set relatively low. However, these algorithms have serious problems with dense datasets. For these cases we suggest using *BtB*. The first step is finding the exact position of the barrier. For this purpose, we propose an iterative automated method. There are two limitations: space limit and time limit. Space limit is determined by the main memory of the platform, while time limit is set by the user (for instance 2 hours). First the *min_supp* is set very high, e.g. 90%. If *Apriori* can find all FIs within the limits, *min_supp* is lowered. The process continues until *Apriori* fails (for two possible reasons: **(1)** not enough memory, or **(2)** too long running time). The barrier is determined by the last successful *min_supp* value. At this point, we change to *BtB*. As *BtB* calls *Apriori-Rare*, a slight modification of *Apriori*, it is guaranteed that *Apriori-Rare* will not fail with this *min_supp* value if *Apriori* did not fail. Our global data mining methodology (see Section 6.2) can also be applied to rare association rules.

To sum up, first we start with *Apriori*, and we lower the minimum support in an iterative way. When *Apriori* fails, we change to *BtB* in order to find rare association rules below the barrier.

5.3.9 A Classification of Association Rules by the Frequency of the Antecedent and Consequent

Association rules[37], by the frequency of the left and right sides, can be grouped in four classes:

1. r̲are → r̲are (i.e. a rare itemset implies a rare itemset)
2. r̲are → f̲requent (i.e. a rare itemset implies a frequent itemset)

[37] Any kind of association rules, either frequent or rare.

3. frequent → rare (i.e. a frequent itemset implies a rare itemset)
4. frequent → frequent (i.e. a frequent itemset implies a frequent itemset).

From now on, we will refer to these rules as rule types RR, RF, FR and FF respectively.

Frequent association rules. All frequent association rules are in type FF. The reason is the following. An association rule r: $X \rightarrow Y \backslash X$ $(X \subset Y)$ is frequent if the union of the left and right sides is frequent. By the downward closure property (Property 3.1), any subset of a frequent itemset is frequent. Thus, X and $Y \setminus X$ are frequent.

Rare association rules. Rare association rules can be in types RR, RF or FR. However, MRG rules can only be in types RR or RF because by definition, their antecedents are minimal *rare* generators. In our work, we have not yet worked with rules of type FR, but they can also be generated easily. Once we have the frequent equivalence classes and the rare equivalence classes (that are found by using the MRGs), the rule generation process is similar to the generation of $\mathcal{MNR}$ rules. For instance, from dataset D (Table 3.1), the following FR rules are extracted (see also Figure 5.2):

FR rule	support	confidence
$A \rightarrow BDE$	1	$1/4 = 0.25$
$B \rightarrow CE$	3	$3/4 = 0.75$
$B \rightarrow AE$	3	$3/4 = 0.75$
$B \rightarrow ADE$	1	$1/4 = 0.25$
$E \rightarrow BC$	3	$3/4 = 0.75$
$E \rightarrow AB$	3	$3/4 = 0.75$
$E \rightarrow ABD$	1	$1/4 = 0.25$

Table 5.8: FR rules in dataset D (Table 3.1) by $min\_supp = 4$.

The FR rules are generated by using a frequent and a rare equivalence class, thus they also have the property that the antecedent is a (frequent) generator and the union of the antecedent and consequent is a (rare) closed itemset. For finding rare equivalence classes, we have used the MRGs, as in the case of "exact MRG rules".

Property 5.1 *Association rules $X \rightarrow Y \setminus X$ $(X \subset Y)$, where X is frequent and $Y \setminus X$ is rare, are approximate rules.*

Property 5.1 states that FR rules are approximate rules.

Proof. Since the rule r: $X \rightarrow Y \backslash X$ $(X \subset Y)$ is an FR rule, X is frequent and $Y \backslash X$ is rare. As $Y \backslash X$ is rare, Y is rare too (by Property 3.2). As Y is rare and X is frequent, $supp(Y) < supp(X)$. Thus, the confidence of r is: $supp(X \cup (Y \setminus X))/supp(X) = supp(Y)/supp(X) < 1$. □

5.3.10 Experimental Results

We evaluated *BtB* on five different datasets. Table 5.10 shows the different steps of finding exact MRG rules. It shows the name of the dataset, the minimum support value, the number of frequent and frequent closed itemsets, the total number of minimal rare generators (zero itemsets included)[38], the number of non-zero MRGs, the number of found rare equivalence classes (support > 0), the number of complex rare equivalence classes (support > 0), the number of generated exact MRG rules, the number of these rules in types RR and RF, and the time of rule generation. The number of FIs is only indicated to show the combinatorial explosion of FIs as the *min_supp* is lowered. The headers of the most important columns are printed in bold, which are: **(1)** FCIs (they are only needed if we are interested in knowing the support of the consequents, as it was the case in our experiments), **(2)** MRGs whose support exceeds 0, and **(3)** complex rare equivalence classes that are found by using non-zero MRGs.

Example. The first row of Table 5.10 contains information about dataset D (Table 3.1). If *min_supp* $= 4$, i.e. 80% (see also Figure 5.2), then we can extract 5 FIs of which 3 are closed. *Apriori-Rare* finds 6 MRGs, none of them have support 0. Calculating their closures we have 4 rare equivalence classes. One of them is simple, but 3 of them are complex, thus they are appropriate for generating exact MRG rules. Among the 5 exact rules, there is 1 of type RR and 4 of type RF. The rule generation process, including the derivation of the support of the consequents, required 0.03 seconds.

For each dataset, the *min_supp* value was lowered in an iterative way in order to find the barrier. For each dataset (except for D), the lowest *min_supp* value indicates the barrier, i.e. the lowest value before hitting the space limit of the platform or the time constraint of 10,000 seconds.

Distribution of MRG Rules

As can be seen in Table 5.10, the most challenging database is the C73D10K, where *min_supp* could only be lowered to 65%. With *BtB* we could find 3,675 MRG rules *below* this threshold. Table 5.9 shows some statistics about the distribution of these rules. The rule with the smallest support has support 1 (0.01%), while the rule with the largest support has support 6,499 (64.99%). As indicated, several rules are found that can be considered to be very rare (interval]0%–10%[). Note that altogether we have found 61 rules with support 1 (0.01%).

	C73D10K *min_supp* = 65%
smallest support	1 (0.01%)
largest support	6,499 (64.99%)
]0%–10%[	1,447
[10%–20%[	34
[20%–30%[	33
[30%–40%[	16
[40%–50%[	35
[50%–60%[	120
[60%–65%[	1,990

Table 5.9: Distribution of MRG rules.

[38]See Def. 5.3.

dataset	min_supp	# FIs	# **FCIs**	# MRGs (all)	# **MRGs (non-zero)**	# rare eq. classes (non-zero)	# **rare eq. classes (non-zero, complex)**	# MRG rules (exact)	# RR rules (exact)	# RF rules (exact)	time of rule generation (sec)
D	80%	5	3	6	6	4	3	5	1	4	0.03
T20I6D100K	10%	7	7	907	907	907	27	27	27	0	2.40
	5%	99	99	5,645	5,645	5,645	27	27	24	3	2.10
	2%	378	378	68,411	68,411	68,411	27	27	19	8	1.65
	1%	1,534	1,534	169,819	169,819	169,819	796	796	92	704	7.26
	0.75%	4,710	4,710	211,578	211,561	211,557	4,049	4,053	1,609	2,444	79.40
	0.5%	26,836	26,208	268,915	268,589	268,446	16,100	16,243	6,059	10,184	385.38
	0.25%	155,163	149,217	537,765	534,088	531,555	43,458	45,991	17,038	28,953	1828.28
T25I10D10K	10%	20	20	1,099	1,099	1,098	63	64	62	2	0.66
	5%	142	142	10,798	10,798	10,797	63	64	60	4	0.61
	2%	533	533	106,931	106,661	106,427	5,914	6,148	3,673	2,475	31.76
	1%	2,893	2,676	223,262	214,004	200,126	35,242	49,120	34,827	14,293	322.62
	0.75%	17,073	7,841	279,363	260,958	236,011	46,422	71,369	51,061	20,308	529.03
	0.5%	302,284	52,033	413,827	376,107	332,478	66,450	110,079	72,807	37,272	1103.92
C20D10K	10%	89,883	8,777	901	837	778	778	837	42	795	0.72
	5%	352,611	21,213	2,002	1,867	1,682	1,682	1,867	48	1,819	1.26
	2%	1,741,883	50,729	7,735	7,065	6,014	6,014	7,065	209	6,856	6.77
	1%	6,194,967	85,608	18,666	15,433	12,485	12,485	15,433	461	14,972	20.08
	0.75%	9,004,043	103,892	24,641	21,065	16,652	16,652	21,065	604	20,461	34.61
	0.5%	15,602,883	132,952	37,816	33,266	25,165	25,165	33,266	1,084	32,182	71.47
	0.25%	40,450,371	193,448	69,104	62,173	41,915	41,915	62,173	3,681	58,492	284.47
C73D10K	95%	1,007	93	1,622	1,622	1,570	1,570	1,622	1,400	222	15.52
	90%	13,463	942	1,701	1,701	1,625	1,625	1,701	1,368	333	16.39
	85%	46,575	2,359	1,652	1,652	1,591	1,591	1,652	1,347	305	16.15
	80%	109,159	4,262	1,802	1,802	1,698	1,698	1,802	1,320	482	17.21
	75%	235,271	9,367	1,939	1,939	1,794	1,794	1,939	1,265	674	22.12
	70%	572,087	19,501	2,727	2,727	2,365	2,365	2,727	1,155	1,572	30.48
	65%	1,544,691	47,491	3,675	3,675	2,953	2,953	3,675	1,078	2,597	53.69
MUSHROOMS	50%	163	45	147	147	139	139	147	68	79	0.28
	40%	505	124	254	251	234	234	251	66	185	0.32
	30%	2,587	425	409	402	361	361	402	76	326	0.44
	20%	53,337	1,169	1,004	959	853	853	959	124	835	0.71
	10%	600,817	4,850	3,077	2,916	2,324	2,324	2,916	187	2,729	1.50
	5%	4,137,547	12,789	8,806	7,963	5,430	5,430	7,963	549	7,414	5.43
	1%	92,894,869	52,708	41,557	37,034	16,799	16,799	37,034	1,239	35,795	57.80

Table 5.10: Steps taken to find MRG association rules.

5.3.11 Conclusion

Frequent association rule mining has been studied extensively in the past. The model used in all these studies, however, has always been the same, i.e. finding all rules that satisfy user-specified *min_supp* and *min_conf* constraints. However, most rules with high support are obvious and well-known, and it is the rules of low support that provide interesting new insights.

In this part of our work we presented a method to extract interesting rare association rules that remain hidden with a conventional frequent itemset mining algorithm. These rules, called "MRG rules", are very similar to frequent minimal non-redundant association rules ($\mathcal{MNR}$) in the sense that the antecedent is a generator and the union of the antecedent and consequent is a closed itemset. The main difference is that these rules are generated from *rare* equivalence classes, and the antecedent is not a "simple" rare generator but a minimal rare generator (MRG) with the additional property that all its proper subsets are frequent. Another advantage of our method is that MRG rules can be found in all cases. We also presented a methodology for mining rare association rules. The idea is to start working with *Apriori* and decrease the *min_supp* value in an iterative way to as low as possible so that the absolute minimum support called the *barrier*, can be found. When *Apriori* fails, we change to *BtB*, and it is capable of finding MRG rules below this threshold. The MRG rules, similarly to the $\mathcal{MNR}$ rules, are interesting because they allow the deduction of maximum information with a minimal hypothesis. Furthermore, they are non-redundant because these rules contain the most information among rules with the same support and same confidence.

There are several bases defined for frequent itemsets. We are curious to know if similar bases can be defined for rare association rules either. Exact MRG rules represent a subset of all exact rare association rules. It is natural to ask if there is a way to restore all exact rare association rules. Since all rare itemsets can be restored from MRGs, as we showed with the *Arima* algorithm, we suppose that the answer is positive. With all this, can we say that the exact MRG rules are a condensed representation of all exact rare rules? Kryszkiewicz states in [Kry02] that a frequent rule representation should be lossless, sound and informative. How can these definitions be applied to rare association rules?

As can be observed, rare association rules raise lots of questions. As a future perspective, we plan to continue studying these rules. We also plan to describe experiments done on the real-world data of the STANISLAS cohort, in order to provide a concrete example of this promising field of KDD.

Chapter 6

A Data Mining Methodology and the Coron Toolkit

6.1 An Overview of the Coron System

In this thesis, we have studied and designed an intelligent information system that allows one to **(i)** extract knowledge pieces from datasets (extract frequent itemsets and association rules), and to **(ii)** organize the extracted knowledge units (classification and visualization with formal concept analysis). We have designed a domain independent, multi-purpose data mining platform called CORON[39], which incorporates a rich collection of data mining algorithms, and allows a number of auxiliary operations. At present, CORON appears to be an original working platform, integrating efficient algorithms for both itemset and association rule extraction. CORON also provides support for preparing and filtering data, and, for interpreting the extracted units of knowledge.

In our case, the extracted knowledge units are association rules. At the present time, finding association rules is one of the most important tasks in data mining. Association rules allow one to reveal "hidden" relationships in a dataset. Finding association rules requires first the extraction of frequent itemsets.

Most experiments with CORON were performed on a real-life biomedical dataset called STANISLAS cohort. During these experiments, I have worked together with one of my colleagues, Sandy Maumus[40], who was an expert on this dataset. During the course of these experiments, we realized that we needed a **(1)** methodology for mining, and **(2)** a tool for implementing the methodology. This inter-disciplinary collaboration led to the development of a global data mining methodology, and to the development of the CORON platform (see Appendix G for a detailed user guide).

Currently, there exist several freely available data mining algorithms and tools. For instance, the goal of the FIMI workshops[41] is to develop more and more efficient algorithms in three categories: **(1)** frequent itemsets (FI) extraction, **(2)** frequent closed itemsets (FCI) extraction, and **(3)** maximal frequent itemsets (MFI) extraction. However, they tend to overlook one thing: the *motivation* to look for these itemsets. After having found them, what can be done with them? In Chapter 4 it is shown that extracting FIs, FCIs, or MFIs only is not enough to generate really useful association rules (see also the *Zart* algorithm in Section 3.3.1). The FIMI algorithms may

[39]`http://coron.loria.fr`

[40]Sandy MAUMUS defended her PhD on 15 November 2005.

[41]`http://fimi.cs.helsinki.fi/`

be very efficient, but they are not always suitable for our needs. Furthermore, these algorithms are independent, i.e. they are not grouped together in a unified software platform. We also did experiments with other toolkits, like WEKA[42]. WEKA covers a wide range of machine learning tasks, but it is not really suitable for finding association rules. The reason is that it provides only one algorithm for this task, the *Apriori* algorithm. *Apriori* finds FIs only, and is not efficient for large, dense datasets.

Because of all these reasons, we decided to group the most important algorithms into a software toolkit that is aimed at data mining. We also decided to build a methodology and a platform that implements this methodology in its entirety. Another advantage of the platform is that it includes the auxiliary operations that are often missing in the implementations of single algorithms, like filtering and pre-processing the dataset, or post-processing the found association rules. Of course, the usage of the methodology and the platform is not narrowed to one kind of dataset only, i.e. they can be generalized to arbitrary datasets.

Extraction of Frequent Itemsets and Association Rules with Coron

CORON is an integrated software platform including components for finding frequent (closed) itemsets in binary contexts and for extracting different kinds of association rules, i.e. key operations that can be required in practical applications. CORON is designed to cover a wide range of basic tasks of symbolic data mining, including pre-processing the dataset, extracting frequent itemsets, generating association rules using these itemsets, and filtering the rules according to different conditions. CORON offers an open architecture and generic implementations which ease its adaptation to the particular application domain and problem setting. The platform supports several data formats (context types), like binary contexts or relational context families. In addition, a rich set of algorithmic methods for symbolic data mining is included in the system's architecture. CORON is designed in parallel with the development of the GALICIA platform[43] for formal concept analysis experiments.

Extracting itemsets. The central part of CORON called Coron-base contains a rich set of algorithms. The currently implemented algorithms for extracting frequent (closed) itemsets are the following: *Apriori*, *Apriori-Close*, *Close*, *Titanic*, *Pascal*, *Pascal*$^+$, *Zart*, *Eclat*, *Eclat-Z*, *Charm* and *Charm-MFI*. Beside frequent itemsets, Coron-base contains algorithms for extracting rare itemsets too, such as *Apriori-Rare*, *Arima* and *BtB*.

Extracting association rules. A module of the CORON system called ASSRULEX (Association Rule eXtractor) is specialized to work with the previously found itemsets to generate different sets of frequent association rules, namely **(1)** all valid rules, **(2)** closed rules and **(3)** the family of minimal non-redundant rules ($\mathcal{GB}$, $\mathcal{IB}$, $\mathcal{RIB}$, $\mathcal{MNR}$, $\mathcal{RMNR}$). ASSRULEX also supports the extraction of rare association rules, i.e. finding exact MRG rules.

The chapter is organized as follows. Section 6.2 details the global data mining methodology and the different modules of the CORON system. Furthermore, we present a methodology for finding rare itemsets that can suggest new hypotheses to be tested and show new research directions. Section 6.3 gives an introduction to mining biological data and describes our real-world dataset, the STANISLAS cohort. Major results, obtained from the application of the methodology on the

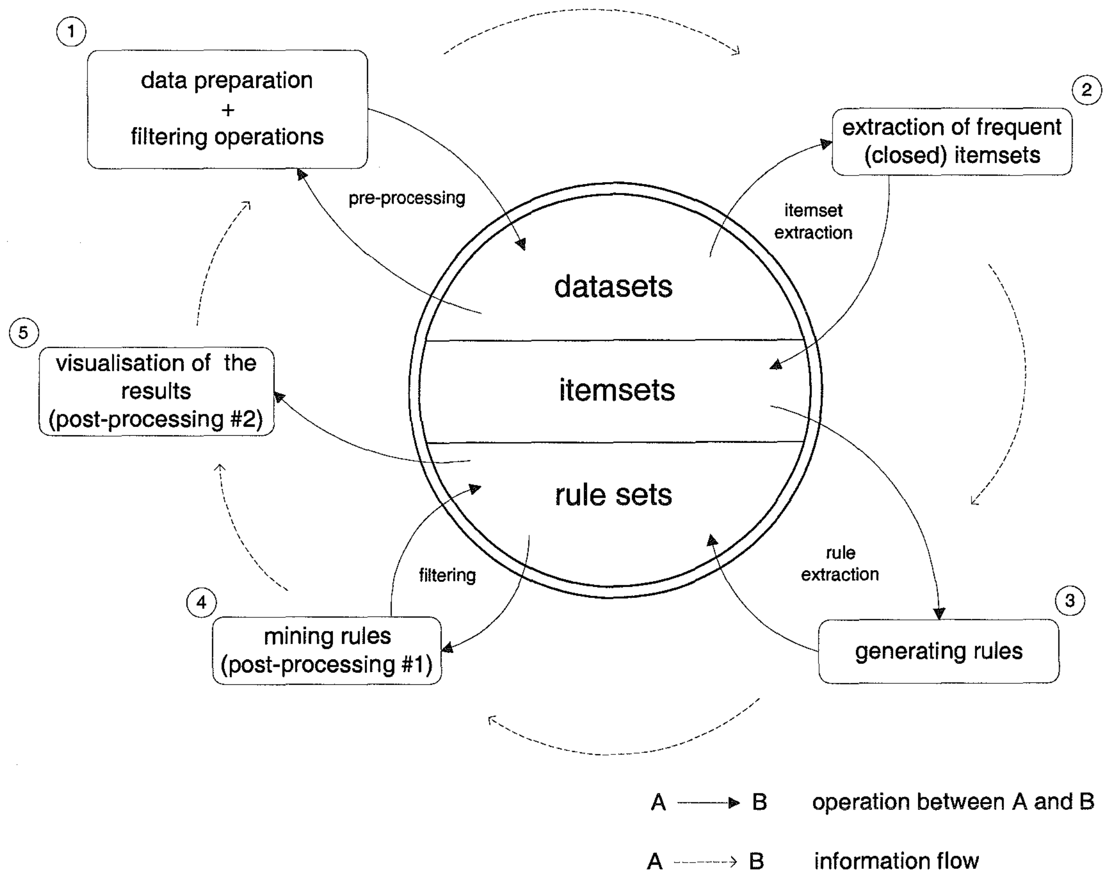
[42]`http://www.cs.waikato.ac.nz/~ml/weka/`
[43]`http://galicia.sourceforge.net`

STANISLAS cohort using association rule extraction, are summarized at the end of Section 6.3, together with our conclusions and perspectives on this real-life database. Other uses of the CORON system are presented in Section 6.4.

6.2 A Global Data Mining Methodology

The methodology was initially designed for mining biological cohorts, but it is generalizable to *any* kind of database. It is important to notice that the whole process is guided by an expert, who is a specialist of the domain related to the database. Her role may be crucial, especially for selecting the data and for interpreting the extracted units (for fully becoming knowledge units). CORON (which is described below) is designed to satisfy the present methodology and offers all the tools that are necessary for its application in a single platform. Nevertheless, the user who prefers to test other tools than CORON can also apply the proposed global data mining methodology.

The methodology consists of the following steps: **(1)** Definition of the study framework, **(2)** Iterative step: data preparation and cleaning, pre-processing step, processing step, post-processing step; Validation of the results and Generation of new research hypotheses; Feedback on the experiment. The life-cycle of the methodology is shown in Figure 6.1.



1
data preparation + filtering operations
pre-processing
2
extraction of frequent (closed) itemsets
itemset extraction
datasets
itemsets
rule sets
5
visualisation of the results (post-processing #2)
filtering
rule extraction
4
mining rules (post-processing #1)
3
generating rules
A → B operation between A and B
A ⇢ B information flow

Figure 6.1: Life-cycle of the mining methodology.

6.2.1 Definition of the Study Framework

The expert of the domain defines a specific field for the analysis (called hereafter "framework"). Thus, she may choose on what type of data she wants to work: biological data, genetic data, or both; unrelated individuals or families; focus on a special metabolic network or on a particular syndrome.

6.2.2 Iterative Step

Data preparation and cleaning. Data cleaning is necessary. This step includes the detection and the possible removal of incomplete and out-of-range values. Moreover, several actions for converting the data can be done at this step:

- *Addition/creation* of new attributes for helping the extraction of association rules by combining attributes (intersection, union and complementary).
- *Deletion* of attributes that are not interesting in the chosen biological framework. This option is close to the projections described below.
- *Discretization*: transform continuous data into Boolean values, for instance by using a threshold defined in the literature, or by separating values of each continuous variable into quartiles.

Data filtering (pre-processing). Several actions can be carried out that correspond to operations found in set theory: complement, union and intersection (with operations of additions and projections).

- *Apply projections*:
    - On the rows: i.e. selecting individuals with one or more attributes specified by the expert.
    - On the columns: i.e. selecting (or deleting) some attributes.
- *Consider the complement* of a set of individuals satisfying a rule, defined by the set of individuals who do not satisfy this rule.

Thus, the output of the filtering process is considered as a new dataset on which data mining procedures can be applied again.

Applying the data mining procedure. We used symbolic data mining methods, in particular, frequent itemset search and association rule extraction. With the help of the expert, the necessary thresholds values can be set for quality indices such as the minimum support and the minimum confidence for generating frequent itemsets and association rules, respectively. As the process is iterative and interactive, the expert can change these thresholds during a next iteration to carry out new experiments.

Post-processing. By filtering and visualizing the rules, we can find those rules that contain the most interesting attributes. If a less relevant attribute is always present in the rules, it can be considered as "noise", and can be deleted from the input dataset. This means, we extract association rules again from the modified dataset. The iterative step can be repeated until the most relevant rules are found. The expert's interpretation is mobilized both for rule mining and result visualization. In the rule mining step, the expert can also make several choices:

- *Choosing rules with a specific form* (e.g. selecting rules that only have one attribute on their left side).
- *Selecting rules with an attribute of interest* from the expert's point of view, on the left hand side, on the right hand side, or on both sides.
- *Classifying the extracted rules* in ascending (or descending) order according to their support or confidence values (or even according to other statistical values [CNT05]).
- *Selecting rules* with a support belonging to a given interval $[a, b]$; returning rules with a support less than (or more than) or equal to a given value c. These selections can also be applied with the other statistical indices cited above.

The rule mining step may be dependent on numerical measures, e.g. support and confidence, or on domain knowledge as shown in some experiments [JCK+04].

Visualization of the results. A visualization method adapted to symbolic data mining method procedure has to be chosen. For frequent itemset search leading to the extraction of less frequent itemsets, concept lattices may be used beneficially [Jay03, VW94, Col01].

Validation of the results and generation of new research hypotheses. The evaluation of the rules can be done either by statistical tests or numerical data analysis, i.e. component analysis. The generated results allow the expert to suggest new directions of research. Accordingly, these new hypotheses are tested by new experiments managed at the biological level, like genetic epidemiological studies or wet laboratory experiments.

6.2.3 Using the Methodology with Rare Association Rules

In Section 5.3 we presented a method for extracting association rules with such low support values that cannot be handled with conventional frequent itemset mining algorithms. We are only interested in those rare rules that have very high confidence values. An iterative and automated method for finding such rules is detailed in Section 5.3.8. We recall here the basic principles. First, we start mining with a frequent itemset mining algorithm, say *Apriori*. Iteratively, the minimum support threshold is set as low as possible. When *Apriori* fails, i.e. the *barrier* is reached, we change to *Apriori-Rare*. *Apriori-Rare* identifies the so-called minimal rare generators (MRGs). Then, using MRGs it is possible to generate rare association rules (we call them "MRG rules").

The global data mining methodology, as shown in Figure 6.1, can be applied in the same way to rare association rules as it is applied to frequent rules.

6.3 Experiments on a Real-Life Biological Database

6.3.1 Introduction to Mining Biological Data

A cohort study consists of following a defined population during a period of time, and of recording different data concerning this population [BHC+95]. Data from cohorts show a high rate of complexity. First, these data are longitudinal ones, i.e. they can vary in time. Secondly, these studies involve a large number of individuals, and collect a large quantity of different parameters [MCHSV02]. Moreover, these data are of different types: quantitative, qualitative, textual, binary, etc. Finally, when the recording terminates, some values of variables may be missing or noisy, leading to incomplete databases. All these characteristics show that data from a cohort study can be considered as complex data. Therefore, the exploitation of such data is a non-trivial task.

An example of the complexity met in cohort studies is given by the STANISLAS cohort. This ten-year study is designed to study cardiovascular risk factors. It involves presumably healthy French families (familial data, with subjects of different ages), and records different types of data (clinical, biological, environmental and genetic data) at three points of health examination, separated from each other by a five-year interval (longitudinal data).

The experts involved in the study of the STANISLAS cohort are specialists of the cardiovascular domain, and they are interested in finding associations relating one or more genetic features (polymorphisms) to biological cardiovascular risk factors. To satisfy this objective, many types of statistical analysis are classically used by biologists. However, such analyses are valid only if an *a priori* hypothesis to be tested is proposed. Facing the amount and the complexity of the data, experts need alternative methods that could help them to foresee new hypotheses.

Symbolic data mining methods could play role of these alternative methods. They have already been used in biology for different purposes [BBJ+02, CCH03, CH03]. In this chapter, we present an experiment for evaluating symbolic data mining methods, and in particular frequent itemset search and association rule extraction in the context of the STANISLAS cohort. We have used the *Zart* algorithm that is implemented in CORON [SN05]. In the case of the STANISLAS cohort, the objective is to discover frequent itemsets and association rules linking biological risk factors and genetic polymorphisms. As a genetic polymorphism is defined as a variation in the DNA sequence that occurs in at least one percent of the population, it is easily understandable that the frequency of the different genetic variants is relatively low in the STANISLAS cohort, given that it is based on a presumably healthy population. Therefore, rather than analyzing frequent itemsets in the present study, we are much more interested in rare (or infrequent) itemsets.

6.3.2 Studied Population: the STANISLAS Cohort

The STANISLAS cohort is a ten-year family study whose main objective is to investigate the impact of genetic and environmental factors on variability of cardiovascular risk factors [MCHSV02, SVH+98]. Recruitment of families living in Northeastern France started in 1993 (t_0) with an invitation for a health examination in the "Centre de Médecine Préventive at Vandœuvre-lès-Nancy", France. At this time, 1006 presumably healthy families (4295 individuals) have been recruited, satisfying the following selection criteria: families of French origin, consisting of two parents, and at least two biological children aged of 4 or more, with members of the family free from serious and/or chronic illnesses. All participants gave their written informed consent. This cohort was approved by the Local Ethics Committee of Nancy, France. In 1998 (t_{+5}), the participants returned for the second health screening with a participation rate of 75%. A third examination is currently under development.

The collected data are of four types: **(1)** Clinical data (e.g. morphometric measures like size, weight, blood pressure); **(2)** Environmental data (life habits, physical activity, medical history, drug intake); **(3)** Biological data (different dosages like glucose, cholesterol, blood count, ...); **(4)** Genetic data: genotypes are determined for each individual based on data stemming from 116 genetic polymorphisms (or SNPs for Single Nucleotide Polymorphisms) that correspond to different metabolic processes involved in cardiovascular diseases.

Thus, it may be noticed that the data in the STANISLAS cohort have many facets and possibly a large range of values, explaining why they are considered complex.

6.3.3 Experiments and Results

The two experiments that are described hereafter were performed on the data of the STANISLAS cohort. This part has been done in collaboration with Sandy Maumus and is dependent on biological knowledge. The interpretation of the biological results have been kept even if they are hard to understand for a computer scientist.

Data preparation. All the data of the STANISLAS cohort are recorded in a Microsoft Access database. Biological, clinical and environmental data can have continuous or discrete values whereas genetic data have discrete values. As the *Zart* algorithm only accepts discrete values, the biological data have been discretized. Two types of discretization are tested. For the first one, each continuous variable is converted into four discrete attributes by using the thresholds given by the calculation of quartiles (this was used in both experiments). For the second type of discretization, each continuous variable is converted into a discrete one using the NCEP thresholds. Indeed, according to NCEP ATP criteria [oH01], an individual has metabolic syndrome if he or she has three or more of the following criteria: waist circumference > 102 cm in men and > 88 cm in women; triglyceride levels ≥ 1.70 mmol/l; HDL cholesterol concentration < 1.04 mmol/l in men and < 1.30 mmol/l in women; blood pressure $\geq 130/85$ mmHg; and fasting glucose value ≥ 6.1 mmol/l. Therefore, the biological data have been converted into the following discrete ones: obesity (increased waist circumference), hypertriglyceridemia, hypoHDLemia, hypertension (increased blood pressure), and hyperglycemia. Moreover, variables have been added for computational reasons in the second experiment: the variable metabolic syndrome ("MS") defined by NCEP criteria; the variable "non MS" that takes the value 1 for a given individual if he does not present the metabolic syndrome; the variables "normal glucose", "normal HDL", "normal blood pressure", "normal triglycerides", "normal waist circumference", that correspond to the negation of the discrete variables hyperglycemia, hypoHDLemia, hypertension, hypertriglyceridemia, and obesity, respectively.

Genetic data of the STANISLAS cohort are recorded in Boolean format in a Microsoft Access database, where each genetic polymorphism is represented by its alleles. A genetic polymorphism can be noted A/a where A is the frequent allele and a is the rare allele. A gene can exist under different forms called alleles. The differences between these gene alleles involve sequence variations. For a given gene, the genotype corresponds to the combination of two alleles. An individual is homozygous for a gene whenever he has two similar alleles for this gene. Inversely, an individual is heterozygous for a gene whenever he has two different alleles for this gene. Two data conversions for the genetic polymorphisms have been tested: **(1)** presentation of the polymorphisms by genotypes: AA, Aa and aa; **(2)** presentation of the polymorphisms by alleles AA and a, where a represents the regrouping of Aa and aa genotypes. In both experiments, the data preparation was done keeping in mind that the main objective here is to extract rare patterns associating biological cardiovascular risk factors and genetic polymorphisms.

Projections. The expert may use horizontal and vertical projections on the discretized database composed of biological and genetic data. Horizontal projections allow the selection of individuals having a particular attribute of interest (e.g. a specific genotype, obesity, or the attribute corresponding to the highest quartile for triglyceride levels). Vertical projections allow the selection of some particular attributes (e.g. rare genotypes or rare alleles).

First Experiment: Detection of Interactions Involving Lipids

The chosen framework (framework #1) is the lipid metabolism. The aim was to detect potential interactions between genetic polymorphisms and biological variables involving lipids. For this specific study, the expert has chosen to work on clinical data (age, sex, BMI[44]), biological data (low-density cholesterol concentration), genetic data (genetic polymorphisms of APOE: APOE codons 112/158 and APOB: APOB Thr71Ile) and environmental data (oral contraceptive intake, smoking and alcohol consumption). Thus, the studied population consisted of 1552 individuals (772 men and 780 women) selected from the STANISLAS cohort.

The use of the *Zart* algorithm on this population led us to generate thousands of rules. Here we present results obtained by performing two different horizontal projections that the expert was interested in. The first projection led to work on a small subset of women having the $\varepsilon2/\varepsilon2$ genotype. An interesting rule from the expert's viewpoint is *R1* (see Table 6.1). *R1* can be interpreted as follows: "A woman being APOE $\varepsilon2/\varepsilon2$ has low LDL-C concentration and is homozygous for the wild type of APOB Thr71Ile gene polymorphism". This rule brings two units of information. First, a well-known result in the domain of lipid research: an individual carrying the $\varepsilon2/\varepsilon2$ genotype for the APOE gene has lower LDL-C concentration. Secondly, a new result: each one of the 780 women being APOE $\varepsilon2/\varepsilon2$ is also APOB 71 Thr/Thr. This result has been validated by a statistical test. Indeed, the APOB Thr71Ile distribution was significantly different according to APOE genotypes in women ($p = 0.016$ in women, $p = 0.428$ in men, Kruskal-Wallis test).

Therefore, we found an interaction between two gene polymorphisms (APOB Thr71Ile and APOE codons 112/158) located on different chromosomes (APOB gene is located at chromosome 2p24 and APOE gene at chromosome 19q13.2), which could suggest a genetic protective profile.

The second projection performed on the studied population generated a subset of 162 individuals with Ile/Ile genotype for the APOB Thr71Ile polymorphism. An interesting extracted rule was *R2* (Table 6.1). This rule can be interpreted as follows: "twelve of the 162 individuals being APOB 71 Ile/Ile are men with a high body mass index and with an LDL-C concentration superior to the NCEP norm [BR95]". This rule is very interesting, because it provides a new hypothesis that did not appear neither in our previous articles, nor in the literature, at least to the best of our knowledge. This new hypothesis, i.e. a potential interaction between body mass index and the APOB Thr71Ile polymorphism, was tested by statistical analysis (general linear regression model). A significant interaction was found between body mass index and APOB Thr71Ile polymorphism in men only ($p = 0.009$ and $p = 0.844$ in men and women respectively). As a consequence, the APOB 71 Ile/Ile genotype was related to higher LDL-C concentration in only men with overweight (BMI > 25 kg/m^2; $p = 0.002$).

Second Experiment: Extraction of a Genetic Profile Characterizing the Metabolic Syndrome

The framework chosen for this experiment is the genetics of the metabolic syndrome (framework #2). The metabolic syndrome is defined as a cluster of synergistically interacting cardiovascular risk factors. It is mainly characterized by insulin resistance, glucose intolerance, dyslipidemia, hypertension, and obesity. To study the genetics of the metabolic syndrome in the STANISLAS cohort, the following variables have been selected: **(1)** biological attributes represented by gender (man or woman), and the five biological variables used by the

[44]BMI (body mass index) is an index that estimates the body mass of an individual. An individual with a BMI between 25 and 30 is considered as overweighted. An individual with a BMI more than 30 is considered as obese.

NCEP-ATPIII [oH01] for defining metabolic syndrome, i.e. plasma glucose concentration, blood pressure, triglyceride concentration, HDL-cholesterol level and waist circumference; **(2)** the 101 genetic polymorphisms available in the STANISLAS database. Three hundred and eight adults have been genotyped for all the genetic polymorphisms. Therefore, we have worked on a starting database of 308 individuals and 235 attributes. Here, the expert is interested in extracting profiles showing the co-occurrence of properties related to metabolic syndrome with specific genetic polymorphisms.

One example of projection that concretely illustrates the extraction of new biological hypothesis is given below. The aim of the experiment is to characterize the genetic profile of individuals of the dataset presenting metabolic syndrome. Here, genetic polymorphisms are divided into their genotypes. A first horizontal projection is done to obtain metabolic syndrome individuals. Nine individuals are kept. Only exact rules (with minimal confidence equal to 100%) are conserved. The nine individuals verify the rule *R3*. In this rule, all the genotypes involved are homozygous for the frequent allele except one that is an heterozygous genotype: APOB 71Thr/Ile. A vertical projection is applied on the entire dataset on the following attributes: the APOB 71Thr/Ile genetic polymorphism, hyperglycemia, hypertriglyceridemia, hypoHDLemia, hypertension, obesity and man and woman. In order to extract association rules involving rare itemsets, the minimal support has to be lowered. As the number of generated rules is high, the set of extracted rules is mined for selecting rules with the attribute metabolic syndrome in the left or in the right hand side. An interesting extracted rule is *R4*, that can be interpreted as "an individual presenting the metabolic syndrome is heterozygous for the APOB 71Thr/Ile polymorphism". This rule has been verified and validated using statistical tests, allowing to conclude that the repartition of genotypes of the APOB71 polymorphism is significantly different if an individual presents metabolic syndrome or not (statistical validation on another data set of 740 individuals, Chi square test, $p = 0.03$). This validation suggests a new biological hypothesis: a subject possessing the rare allele for the APOB 71Thr/Ile polymorphism presents more frequently the metabolic syndrome.

Discussion

The results that are given above show that the mining of the STANISLAS cohort is not a trivial work. Many rules may be extracted that are not necessarily interesting: actually, only a few rules are indeed interesting for the expert of the domain. Moreover, it is very important that the expert is able to reiterate the mining process as many times as needed. Hence, the data preparation and the parameter adjustment of the mining system have to be simple and straightforward operations. In these conditions, the expert may obtain some substantial results.

Here, we have shown how to obtain rules with a special form, i.e. a minimal antecedent and a maximal consequent, in the same way as the experience related in [CH03]. Furthermore, the extracted rules that have been judged as interesting by the expert have given new biological insights, and suggested some fruitful research directions. Thus, it is worth continuing the experience. Even though association rule extraction has been rarely used in biology, there exist some studies [BBJ+02, CH03, QTDDB02, STVN04, SBMP01]. Nevertheless, there are not many studies investigating real-world data such as the data of the STANISLAS cohort. Creighton and Hanash [CH03] studied gene expression data of yeasts obtained from a public database, by using the *Apriori* algorithm [AMS+96]. Moreover, these authors focused on rules whose left hand side contains a single item. This study is close to our work. On the one hand it is close from the methodological point of view, because both *Apriori* and *Zart* are levelwise algorithms. On the other hand, both studies focus on special association rules. In the study of Quentin-Trautvetter *et al.* [QTDDB02], the CBA software (Classification Based on Association) is used for association rule extraction. As the authors are faced with a too large number of generated rules and thus unable to detect interesting rules, they select a limited number of attributes. This is related to our approach using vertical and horizontal projections which are described in the methodology.

Conclusion

This methodology shows that the preparation of the data, and the interpretation of the results, as well as the iterations of the mining process, are of first importance for a mining experimentation in biology. Moreover, the role of the expert of the domain is central and crucial: the whole data mining process is led under the supervision of the expert.

Two experiments involving real-world data of a cohort population have been performed here. The first one led to the detection of interactions involving lipid genes and the second one enabled the extraction of a genetic profile characterizing the metabolic syndrome in the studied population. These two experiments have given a number of results that may be very interesting. The methodology and the associated tools, as well as the behavior of the expert regarding the mining system still have to be improved. This is particularly true for the preparation of data, and for the visualization of results, where some visualization modules still have to be tested.

6.4 Other Uses of the Coron System

Here we demonstrate some other uses of the CORON system. This section is organized as follows. Section 6.4.1 presents the CABAMAKA system for case-based reasoning. Section 6.4.2 shows the collaboration between the GALICIA and the CORON platforms. We also present shortly how CORON is being used for text mining experiments.

Text Mining with Coron

A part of the ongoing research in our team concerns text mining using the CORON system. Results of this experiment will be reported in the future. This project aims at defining a methodology to detect adverse drug effects through databases or texts. In fact, there exist databases where doctors record adverse effects due to drug absorption. The problem is to generate an alert as soon as there are enough "reliable" conditions saying that one or more drugs are responsible for these adverse effects. Current tools use numerical and statistic approaches. We want to apply association rule extraction and lattice based classification to this problem.

6.4.1 The Cabamaka System

This subsection is based on [dBL+06]. This part has been done in collaboration with Mathieu d'Aquin, Fadi Badra, Sandrine Lafrogne, Jean Lieber and Amedeo Napoli.

Introduction

Case-based reasoning (CBR [RS89]) aims at solving a target problem thanks to a case base. A case represents a previously solved problem. A CBR system selects a case from the case base and then adapts the associated solution, requiring domain-dependent knowledge for adaptation. The goal of adaptation knowledge acquisition (AKA) is to extract this knowledge. The system CABAMAKA applies principles of knowledge discovery from databases (ECBD) to AKA. The originality of CABAMAKA lies essentially in the approach to AKA that uses a powerful learning technique

# rule	extracted rule	support (n/%)	confidence (%)
R1	$\{\text{Woman} = 1\} \Rightarrow \{\text{LDL} - \text{C} \leq 3.16,\ \text{APOE}\_\varepsilon 2\varepsilon 2,\ \text{APOB71}\_\text{ThrThr}\}$	5/71.4	100
R2	$\{\text{Man} = 1,\ \text{BMI} \geq 26.5,\ \text{LDL} - \text{C} \geq 4.44\} \Rightarrow \{\text{OC} = 0,\ \text{APOB71}\_\text{IleIle}\}$	12/7.4	100
R3	{} ⇒ {MS, APOAI_121GG, APOAIV_347ThrThr, APOAIV_360GluGlu, ADRB3_64TrpTrp, NOS3 − 948AA, ANP_7ValVal, ENaCa_493TrpTrp, FII_20210GG, IL4R_478SerSer, ADRB2_164ThrThr, CCR3_39ProPro, APOB_71ThrIle, LPL_291AsnAsn, FV_506ArgArg, SELE_554LeuLeu}	9/100	100
R4	{MS} ⇒ {APOB_71ThrIle}	9/3	100

Table 6.1: Extracted association rules that are interesting from the expert's point of view. LDL-C: concentration in low-density cholesterol; BMI: body mass index; OC: oral contraceptive intake; MS: metabolic syndrome.

that is guided by a domain expert, according to the spirit of ECBD. We propose an original and working approach to AKA, based on ECBDtechniques.

CBR and adaptation. A case in a given CBR application is usually represented by a pair (pb, Sol(pb)) where pb represents a problem statement and Sol(pb) is a solution of pb. CBR relies on the *source cases* (srce, Sol(srce)) that constitute the *case base* CB. In a particular CBR session, the problem to be solved is called *target problem*, denoted by tgt. A case-based inference associates to tgt a solution Sol(tgt), with respect to the case base CB and to additional knowledge bases, in particular $\mathcal{O}$, the *domain ontology* that usually introduces the concepts and terms used to represent the cases.

A classical decomposition of CBR consists in the steps of retrieval and adaptation. *Retrieval* selects (srce, Sol(srce)) $\in$ CB such that srce is judged to be similar to tgt. The goal of adaptation is to solve tgt by modifying Sol(srce) accordingly.

The work presented hereafter is based on the following model of adaptation, similar to *transformational analogy* [Car83]:

① (srce, tgt) $\mapsto$ Δpb, where Δpb encodes the similarities and dissimilarities of the problems srce and tgt.

② (Δpb, AK) $\mapsto$ Δsol, where AK is the adaptation knowledge and where Δsol encodes the similarities and dissimilarities of Sol(srce) and the forthcoming Sol(tgt).

③ (Sol(srce), Δsol) $\mapsto$ Sol(tgt), Sol(srce) is modified into Sol(tgt) according to Δsol.

Adaptation is generally supposed to be domain-dependent in the sense that it relies on domain-specific adaptation knowledge. Therefore, this knowledge has to be acquired. This is the purpose of *adaptation knowledge acquisition* (AKA).

A related work in AKA. The idea of the research presented in [HK96] is to exploit the variations between source cases to learn adaptation rules. These rules compute variations on solutions from variations on problems. More precisely, ordered pairs (srce-case$_1$, srce-case$_2$) of similar source cases are formed. Then, for each of these pairs, the variations between the problems srce$_1$ and srce$_2$ and the solutions Sol(srce$_1$) and Sol(srce$_2$) are represented (Δpb and Δsol). Finally, the adaptation rules are learned, using as training set the set of the input-output pairs (Δpb, Δsol). The experiments have shown that the CBR system using the adaptation knowledge acquired from the automatic system of AKA shows a better performance compared to the CBR system working without adaptation. This research has strongly influenced our work that is globally based on similar ideas.

Cabamaka

Principles. CABAMAKA deals with case base mining for AKA. Although the main ideas underlying CABAMAKA are shared with those presented in [HK96], the followings are original ones. The adaptation knowledge that is mined has to be validated by experts and has to be associated with explanations that make it understandable by the user. In this way, CABAMAKA may be considered as a semi-automated (or interactive) learning system. Another difference with [HK96] lies in the volume of the cases that are examined: given a case base CB where $|\text{CB}| = n$, the CABAMAKA system takes into account every ordered pair (srce-case$_1$, srce-case$_2$) with srce-case$_1$ $\neq$ srce-case$_2$ (whereas in [HK96], only the pairs of *similar* source cases are considered, according to a fixed criterion). Thus, the CABAMAKA system has to cope with $n(n-1)$

pairs, a rather large number of elements, since in our application $n \simeq 750$. ($n(n-1) \simeq 5 \cdot 10^5$). This is why efficient techniques of knowledge discovery from databases (ECBD [Dun03]) have been chosen for this system.

Principles of ECBD. The goal of ECBDis to discover knowledge from databases, with the supervision of an analyst (expert of the domain). A ECBDsession usually relies on three main steps: data preparation, data-mining and interpretation.

Data preparation is based on formatting and filtering operations. The formatting operations transform the data into a form allowing the application of the chosen data-mining operations. The filtering operations are used for removing noisy data and for focusing the data-mining operation on special subsets of objects and/or attributes.

Data-mining methods are applied to extract pieces of information from the data. These pieces of information have some regular properties allowing their extraction. For example, *Charm* [ZH02] is a data-mining algorithm that performs efficiently the extraction of *frequent closed itemsets* (FCIs). The CABAMAKA system uses our *Charm* implementation, which is part of the CORON platform.

Interpretation aims at interpretating the output of data-mining i.e. the FCIs in the present case, with the help of an analyst. In this way, the interpretation step produces new knowledge units (e.g. rules).

Formatting. The formatting step of CABAMAKA inputs the case base CB and outputs a set of transactions obtained from the pairs ($\texttt{srce-case}_1, \texttt{srce-case}_2$). It is composed of two substeps. During the first substep, each $\texttt{srce-case} = (\texttt{srce}, \texttt{Sol(srce)}) \in \texttt{CB}$ is formatted in two sets of boolean properties: $\Phi(\texttt{srce})$ and $\Phi(\texttt{Sol(srce)})$. The computation of $\Phi(\texttt{srce})$ consists in translating srce from the problem representation formalism to $2^{\mathcal{P}}$, $\mathcal{P}$ being a set of boolean properties. Possibly, some information may be lost during this translation, but this loss has to be minimized. Now, this translation formats an expression srce expressed in the framework of the domain ontology $\mathcal{O}$ to an expression $\Phi(\texttt{srce})$ that will be manipulated as data, i.e. without the use of a reasoning process. Therefore, in order to minimize the translation loss, it is assumed that if $p \in \Phi(\texttt{srce})$ and p entails q (given $\mathcal{O}$) then $q \in \Phi(\texttt{srce})$. In other words, $\Phi(\texttt{srce})$ is assumed to be deductively closed given $\mathcal{O}$ in the set $\mathcal{P}$. The same assumption is made for $\Phi(\texttt{Sol(srce)})$. How this first substep of formatting is computed in practice depends heavily on the representation formalism of the cases.

The second substep of formatting produces a transaction $T = \Phi((\texttt{srce-case}_1, \texttt{srce-case}_2))$ for each ordered pair of distinct source cases, based on the sets of items $\Phi(\texttt{srce}_1)$, $\Phi(\texttt{srce}_2)$, $\Phi(\texttt{Sol}(\texttt{srce}_1))$ and $\Phi(\texttt{Sol}(\texttt{srce}_2))$. Following the model of adaptation presented in introduction (items ①, ② and ③), T has to encode the properties of $\Delta\texttt{pb}$ and $\Delta\texttt{sol}$. $\Delta\texttt{pb}$ encodes the similarities and dissimilarities of $\texttt{srce}_1$ and $\texttt{srce}_2$, i.e.:

- The properties common to $\texttt{srce}_1$ and $\texttt{srce}_2$ (marked by "="),
- The properties of $\texttt{srce}_1$ that $\texttt{srce}_2$ does not share ("-") and
- The properties of $\texttt{srce}_2$ that $\texttt{srce}_1$ does not share ("+").

All these properties are related to problems and thus are marked by pb. $\Delta\texttt{sol}$ is computed in a similar way and $\Phi(T) = \Delta\texttt{pb} \cup \Delta\texttt{sol}$. For example,

$$\begin{array}{lll} \text{if} & \Phi(\texttt{srce}_1) = \{a, b, c\} & \Phi(\texttt{Sol}(\texttt{srce}_1)) = \{A, B\} \\ \text{and} & \Phi(\texttt{srce}_2) = \{b, c, d\} & \Phi(\texttt{Sol}(\texttt{srce}_2)) = \{B, C\} \\ & & \\ \text{then} & \multicolumn{2}{l}{T = \{a^-_{\texttt{pb}}, b^=_{\texttt{pb}}, c^=_{\texttt{pb}}, d^+_{\texttt{pb}}, A^-_{\texttt{sol}}, B^=_{\texttt{sol}}, C^+_{\texttt{sol}}\}} \end{array} \tag{6.1}$$

Mining. The extraction of frequent closed itemsets from the set of transactions is computed with the *Charm* algorithm. A transaction $T = \Phi((\texttt{srce-case}_1, \texttt{srce-case}_2))$ encodes a specific adaptation
$((\texttt{srce}_1, \texttt{Sol}(\texttt{srce}_1)), \texttt{srce}_2) \mapsto \texttt{Sol}(\texttt{srce}_2)$. An extracted FCI may be considered as a generalization of a set of transactions. For example, if $I_{ex} = \{a^-_{\texttt{pb}}, c^=_{\texttt{pb}}, d^+_{\texttt{pb}}, A^-_{\texttt{sol}}, B^=_{\texttt{sol}}, C^+_{\texttt{sol}}\}$ is an FCI, I_{ex} is a generalization of a subset of the transactions including the transaction T of equation (6.1): $I_{ex} \subseteq T$. The interpretation of this FCI as an adaptation rule is explained below.

Interpretation. The interpretation step is supervised by the analyst. The CABAMAKA system provides the analyst with the extracted FCIs and facilities for navigating among them. The analyst may select an FCI, say I, and interpret I as an adaptation rule. For example, the FCI I_{ex} may be interpreted in the following terms:

if a is a property of `srce` but is not a property of `tgt`,
c is a property of both `srce` and `tgt`,
d is not a property of `srce` but is a property of `tgt`,
A and B are properties of `Sol(srce)` and
C is not a property of `Sol(srce)`
then the properties of `Sol(tgt)` are
$\Phi(\texttt{Sol}(\texttt{tgt})) = (\Phi(\texttt{Sol}(\texttt{srce})) \setminus \{A\}) \cup \{C\}$.

This has to be translated as an adaptation rule r of the CBR system. Then the analyst corrects r and associates an explanation with it.

Implementation. The application domain of the CBR system we are developing is breast cancer treatment: in this application, a problem `pb` describes a class of patients with a set of attributes and associated constraints (holding on the age of the patient, the size and the localization of the tumor, etc.). A solution `Sol(pb)` of `pb` is a set of therapeutic decisions (in surgery, chemotherapy, etc.). The requested behavior of the CBR system is to provide a treatment and explanations on this treatment proposal. This is why the analyst is required to associate an explanation to a discovered adaptation rule.

The problems, solutions and the domain ontology of the application are represented in OWL DL (recommendation of the W3C).

Conclusion

The CABAMAKA system is inspired by the research presented in [HK96] and by the principles of ECBDfor the purpose of semi-automatic adaptation knowledge discovery. It has enabled to discover several useful adaptation rules for a medical CBR application. It has been designed to be reusable for other CBR applications: only a few modules of CABAMAKA are dependent on the formalism of the cases and of the domain ontology, and this formalism, OWL DL, is a well-known standard. One element of future work consists in searching for ways of simplifying the presentation of the numerous extracted FCIs to the analyst. This involves an organization of these FCIs for the purpose of navigation among them. Such an organization can be a hierarchy of FCIs according to their specificities or a clustering of the FCIs in themes.

6.4.2 The Galicia Platform

GALICIA [VGRR03] is intended as an integrated software platform that includes components for the key operations on lattices that might be required in practical applications or in more theoretically-oriented studies. Thus, the basic configuration of the platform performs major functions such as context input, lattice construction and visualization.

GALICIA is designed to cover the whole range of basic tasks that make up the complete life-cycle of a lattice, i.e. contexts are either loaded or created by means of a context editor; lattices are constructed and visualized; rearrangements of the context are performed to clarify the lattice structure; the resulting lattice is reduced to a suborder or decomposed into smaller lattices.

The intended impact of the platform is two-fold since it should support both applications of FCA and development of new lattice-based techniques. As an FCA-tool, GALICIA offers an open architecture and generic implementations which ease its adaptation to a particular application domain and problem settings. For example, a wide range of data formats (context types) are allowed in the platform. In addition, a rich set of algorithmic methods for lattice construction and maintenance is included in the system's architecture. Finally, the platform offers several visualization mechanisms including 2D and 3D graph drawing modes.

Our team Orpailleur has a collaboration with the developers of GALICIA at the University of Québec in Montreal. As a part of this collaboration, the GALICIA platform now includes an implementation of the *Titanic* algorithm done by the author of this thesis. *Titanic* allows constructing *iceberg* concept lattices.

Chapter 7

Conclusion and Perspectives

In this chapter we summarize the research contribution of the thesis, we give a synthesis of our work and point out directions for future work.

7.1 Conclusion

The main topic of this thesis is *knowledge discovery in databases* (KDD). More precisely, we have investigated two of the most important tasks of KDD today, namely itemset extraction and association rule generation. Throughout our work we have borne in mind that our goal is to find *interesting* association rules from various points of view: for efficient mining purposes, for minimizing the set of extracted rules and for finding intelligible (and easily interpretable) knowledge units. We have developed and adapted specific algorithms in order to achieve this goal.

The main contributions of this thesis are: **(1)** We have developed and adapted algorithms for finding minimal non-redundant association rules; **(2)** We have defined a new basis for association rules called Closed Rules; **(3)** We have investigated an important but relatively unexplored field of KDD namely the extraction of rare itemsets and rare association rules; **(4)** We have packaged our algorithms and a collection of other algorithms along with other auxiliary operations for KDD into a unified software toolkit called CORON.

7.1.1 Algorithms for Finding MNR Rules

In Chapter 3 we present two algorithms that we have specifically adapted to extract minimal non-redundant association rules ($\mathcal{MNR}$). This set of rules is a lossless, sound and an informative representation of all valid association rules. The first algorithm, *Zart*, is a practical extension of *Pascal*, which is probably the most efficient levelwise algorithm for finding frequent itemsets (FIs). In *Zart*, *Pascal* is integrated with *Apriori-Close* and with an extension of ours. In addition to *Pascal*'s capabilities, *Zart* identifies the set of frequent closed itemsets (FCIs) and associates their generators to them. We show that this extra output from *Zart* is essential for extracting $\mathcal{MNR}$ rules. In the second algorithm, *Eclat-Z*, we go further and we show how to generalize the idea in *Zart* for *any* frequent itemset mining algorithms. This way, arbitrary FI-miner algorithms can be extended in order to support the extraction of $\mathcal{MNR}$ rules. We present a general idea, and *Eclat-Z* is a concrete implementation of this concept. As its name indicates, we have made an extension of *Eclat*. *Eclat* is a vertical algorithm that identifies FIs very efficiently, but due to the depth-first search it produces FIs in an unordered way by length. *Zart* relies on the hypothesis that FIs are available in an ordered way by length. In *Eclat-Z* we solve this problem with a special file indexing technique. With this idea, arbitrary FI-miner algorithms can be turned into algorithms that find a useful and interesting subset of all valid association rules namely the minimal non-redundant association rules.

7.1.2 Closed Association Rules

In Chapter 4 we introduce a new basis called Closed Rules that we position between the set of all valid rules and the set of minimal non-redundant rules, filling a gap between them. In the case of dense and highly correlated datasets, the generation of closed rules is more efficient than finding all association rules. However, in sparse datasets, when almost all FIs are closed, all association rules can be extracted more efficiently. By all means, the set of Closed Rules is always smaller than the set of all valid association rules. As Closed Rules is a concise representation of all valid rules, and it only requires frequent closed itemsets, Closed Rules seems to be a good alternative to all valid association rules.

7.1.3 Rare Itemsets and Rare Association Rules

Chapter 5 is one of the most original part of this thesis work. In this chapter, we address the problems of extracting rare itemsets and generating rare association rules. In the literature, these problems have not yet been studied in detail, although rare itemsets can also contain important information just as frequent itemsets do. A particularly relevant field for rare itemsets is medical diagnosis.

In Chapter 5.1 we present a method for finding all *rare itemsets*. For this task we use the well-known *Apriori* algorithm. *Apriori* is known to find all FIs, but actually it also finds a special subset of rare itemsets, the minimal rare itemsets (MRIs). A slight modification of *Apriori*, which we call *Apriori-Rare*, retains MRIs instead of dropping them. We show how to restore all rare itemsets from MRIs while avoiding itemsets with support 0.

In Chapter 5.3 we go further by showing how to generate valid *rare association rules*. Our work is motivated by the long-standing open question of devising an efficient algorithm for finding rules with low support and very high confidence. In order to find such rules using conventional frequent itemset mining algorithms like *Apriori*, the minimum support must be set very low, which drastically increases the runtime of the algorithm. Moreover, when minimum support is set very low, *Apriori* produces a huge number of frequent itemsets. This is also known as the *rare item problem*. For this well-known problem we propose the following solution. In an iterative way, we lower the minimum support for *Apriori* until we reach the limit that is still manageable by the algorithm. At this point we change to *Apriori-Rare* that finds MRIs. We prove that MRIs are minimal rare generators (MRGs). By finding their closures, some rare equivalence classes are obtained from whom it is possible to extract exact rare association rules (we call these rules "exact MRG rules"). We also show how to extract approximate MRG rules; however their interestingness is doubtful. Thus, we concentrate more on exact rare rules. Furthermore, these rules are non-redundant because the antecedent is minimal and the consequent is maximal, implying that among rules with the same support and same confidence, these rules contain the most information. MRG rules can be found in all cases, even if the minimum support is set high.

We believe that the extraction of rare association rules is a very interesting and very promising field of KDD. Use of rare association rules can be advantageous in a variety of practical applications, be it marketing, business domains, telecommunications, or scientific fields such as biology, astronomy, medicine, etc. Being a relatively new field, it has not yet been studied in detail. Our approach is a first step in this direction. Rare itemsets and rare association rules raise lots of questions. There exist several bases for frequent association rules. Can we define similar bases for rare association rules too? The exact MRG rules are a subset of all exact rare association rules. Can we derive all these rare exact rules from the set of exact MRG rules? In our approach we consider one border only, i.e. for us an itemset is rare if it is not frequent. An interesting research direction is to work with two borders. While we use the "side product" of levelwise frequent itemset mining algorithms, namely the minimal rare itemsets, one may inquire if it possible to mine these rare itemsets directly, i.e. *without* first extracting frequent itemsets. Answering these questions requires further research.

7.1.4 The Coron Toolkit

All the algorithms presented in this thesis have been implemented in software. They have been grouped together in a unified software platform called CORON. CORON is a domain and platform independent, multi-purposed data mining toolkit, which incorporates not only a rich collection of data mining algorithms, but also allows a number of auxiliary operations. To the best of our knowledge, a data mining toolkit designed specifically for itemset extraction and association rule generation like CORON does not exist elsewhere. CORON also provides support for preparing and filtering data, and for interpreting the extracted units of knowledge.

Most of the experiments with CORON were performed on a real-life biomedical dataset called the STANISLAS cohort. During these experiments, we realized that we needed **(1)** a methodology for mining, and **(2)** a tool for implementing the methodology. Chapter 6 presents our global data mining methodology that can be generalized to arbitrary datasets. The methodology can be used for both frequent and rare association rules.

At the end of Chapter 6, besides the STANISLAS cohort, we present three other projects that use the CORON toolkit with success.

Future Plans for Improving Coron

While CORON is already a nearly comprehensive working platform that has exhibited its usefulness in varied projects, it can be ameliorated in some ways. We list below some ideas for its improvement.

At the moment, using CORON we can extract two concise representations of frequent association rules namely the Closed Rules and the minimal non-redundant association rules. We are interested in working with other bases too, such as the Duquennes-Guigues basis [GD86] for exact rules, the Luxenburger basis [Lux91] for approximate rules, or Kryszkiewicz's representative rules [Kry98]. All these bases are more concise than the $\mathcal{MNR}$ rules. However, they are not lossless, sound and informative at the same time.

Currently, CORON can only work with binary contexts. Many-valued contexts should also be supported in a future version.

CORON contains, among other algorithms, *Eclat* and *Charm*. Recently, Zaki proposed an optimization called diffsets that significantly reduces the memory footprint of these algorithms (see Section 3.2.2). The optimized versions are called *dEclat* and *dCharm*. We plan to add these optimized algorithms in CORON. Since they use much less memory, it is likely that they allow treating larger databases or smaller minimum support values on the same platform.

Currently, CORON does not have algorithms in the other category (see Section 3.2.4). Maybe the most well-known algorithm of this kind is *FP-growth* [HPY00], which uses a new data structure called FP-tree. This novel data structure is a compressed representation of all the transactions in the database. While *FP-growth* finds frequent itemsets, some of its modifications like *Closet* [PHM00] or *Closet*$^+$ [WHP03] are designed to extract frequent closed itemsets only.

Charm-MFI is a simple extension of *Charm* that filters MFIs among FCIs. We plan to compare it with some other algorithms that are specifically made for maximal frequent itemsets, e.g. *Max-Miner* [Bay98], *DepthProject* [AAP00], *GenMax* [GZ01], etc.

CORON is a project with one of the goals to integrate a large number of itemset mining algorithms. Since there is no *best* algorithm for arbitrary datasets, CORON offers the possibility for a user to test several algorithms and to choose the one that best suits his needs. However, a large list of algorithms could easily confuse users. As a consequence, we should make a thorough experimental study on a large set of different databases to give indications for a user which algorithm is suggested for a specific type of dataset. Or, CORON could also analyse the dataset and offer an algorithm that, beside the given parameters such as *min_supp*, could possibly give an optimal performance.

KDD refers to the overall process of discovering new, useful and understandable patterns in data. Developing efficient algorithms is just one of the steps of this process. For the visualization step, we use formal concept lattices. For constructing the lattices, first we find FCIs, then we calculate the order among concepts. However, there are more efficient ways to calculate the order; for instance, by using algorithms that explore the order parallely with the FCIs. Currently, CORON only has algorithms that extract itemsets and association rules, but in the future we plan to incorporate in it algorithms from the field of formal concept analysis too [KO01, KO02] in order to make its visualization module more efficient. As we have a collaboration with the developer team of the GALICIA project, it is very likely that the two platforms will borrow algorithms from one another, or maybe the two platforms will be merged together.

Another objective is to connect CORON with database management systems. For instance, Coron-base and ASSRULEX should be able to save their results directly in a DBMS system in order to facilitate data exchange with other projects that want to use the results produced by CORON.

7.2 Mid- and Long-Term Perspectives

The final component in any data mining algorithm is the data management strategy: the ways in which the data are stored, indexed and accessed. Most well-known data analysis algorithms have been developed with the assumption that the dataset can be accessed quickly and efficiently in the main memory (RAM). While memory technology has improved rapidly, there have been equally rapid improvements in secondary storage technologies. Many massive datasets do not fit in available RAM, thus new algorithms and/or new data management strategies are needed. Today, gigabyte or even terabyte databases are not uncommon. The ultimate goal is to be able to handle efficiently such large databases too.

Another challenge is the treatment of constantly evolving databases, e.g. records of telephone calls or electricity usage. Databases can have new records, new attributes or attributes of existing records can have new values. Previously extracted knowledge may need to be updated or it may even become invalid. This requires using incremental algorithms. Databases will continue to increase in size. It is generally agreed that such large databases can only be treated efficiently with parallel algorithms in a distributed environment. Parallel processing is ideally suited for addressing issues of scalability. Data mining algorithms should be integrated more with DBMS systems to provide common representation, storage and retrieval.

Combining the previously mentioned characteristics –DBMS integration with incremental parallel processing– could result in very powerful data mining solutions. It would be interesting to develop such combined solutions for mining rare association rules in large databases.

Chapter 8

Summaries in French and in Hungarian

8.1 Résumé étendu

8.1.1 Introduction

L'extraction de connaissances dans les bases de données

Le processus d'*extraction de connaissances dans les bases de données* a pour objectif d'extraire, à partir de grandes bases de données, des unités d'information pouvant être interprétées en tant que connaissances réutilisables. Ce processus repose sur trois étapes principales : la sélection et la préparation des données, la fouille de données et, finalement, l'interprétation des unités extraites.

L'extraction de connaissances dans les bases de données peut être vue de manière similaire à l'orpaillage : les pépites d'or recherchées sont dans ce cas les unités de connaissances et la rivière la base de données considérée. D'importants volumes de données — en particulier de documents — sont disponibles sans information a priori concernant leur usage. Une question fondamentale est de savoir si ces données contiennent quelque chose d'intéressant et de trouver des méthodes pour extraire ces "éléments d'intérêt". L'*extraction des connaissances dans les bases de données* — notée dans la suite ECBD — consiste à traiter d'importants volumes de données dans le but d'en extraire des unités de connaissances non triviales, potentiellement utiles, significatives et réutilisables. De façon générale, le processus d'ECBD est itératif et interactif. Il est contrôlé par un expert des données, appelé l'*analyste*, dont le rôle est de guider le processus d'extraction, sur la base de ses objectifs et de ses connaissances du domaine. L'analyste sélectionne et interprète un sous-ensemble des unités extraites pour construire des "modèles" qui seront dans la suite considérés comme des unités de connaissances auxquelles est associée un degré de plausibilité. Le processus d'ECBD repose sur trois étapes principales : **(i)** préparation : les sources de données sont préparées pour être traitées, **(ii)** fouille : elles sont alors fouillées et, **(iii)** interprétation : finalement, les unités d'information extraites sont interprétées pour devenir des unités de connaissances. Ces unités sont représentées dans un formalisme de représentation des connaissances, afin d'être utilisées au sein d'un système à base de connaissances. L'ECBD peut aussi être vue comme un processus permettant de passer des données aux informations, puis aux connaissances (voir la Figure 8.1), en considérant les définitions suivantes [SAA$^+$99, Wil02] :

Données. Les données sont les *signaux* non interprétés qui atteignent nos sens à chaque minute. Une lumière rouge, verte ou orange à un carrefour est un exemple de donnée. Les ordinateurs sont emplis de données : signaux codant des chaînes de caractères, des nombres, des caractères

Données (données brutes, bases de données)
↓ Compréhension du domaine
↓ Sélection de données (fenêtrage)
Données sélectionnées
↓ Nettoyage des données / Transformation des données
↓ Préparation de l'ensemble des données
Données préparées
↓ Processus de fouille de données (découverte de motifs)
↓ Méthodes numériques et symboliques d'ECBD
Motifs extraits
↓ Post-traitement des motifs extraits
↓ Interprétation / Évaluation
Unités de connaissances (pour des systèmes à base de connaissances)

Figure 8.1: La boucle d'ECBD : des données brutes aux unités de connaissance. L'objectif du processus d'ECBD est de sélectionner, de préparer et d'extraire des unités de connaissances depuis différents sources, puis de représenter les unités de connaissances dans des structures appropriées.

ou d'autres symboles qui sont traités en grande quantité, de façon mécanique et transparente.

Informations. Les informations sont des données associées à un *sens.* Pour un conducteur de voiture, la lumière rouge (le feux rouge) n'est pas seulement un signal d'une certaine couleur, mais aussi une indication signifiant qu'il doit s'arrêter. Par contre, un daltonien n'attachera probablement pas le même sens à une lumière rouge.

Connaissances. Les connaissances sont constituées de l'ensemble des données et des informations qu'une personne peut utiliser dans le but de réaliser certaines tâches et de créer de nouvelles informations. Les connaissances montrent une aptitude de *génération*, du fait qu'une des fonctions principales des connaissances est de produire de nouvelles connaissances.

Le processus d'ECBD est réalisé au sein d'un système d'ECBD qui se compose des éléments suivants : les bases de données, les modules de fouilles de données (symboliques et numériques) et les interfaces pour l'interaction avec le système, par exemple l'édition et la visualisation. De plus, le système d'ECBD peut tirer parti des connaissances du domaines, intégrées au sein d'une ontologie du domaine des données. Fermant la boucle, les unités de connaissances extraites par le système d'ECBD doivent être représentées dans un formalisme de représentation des connaissances adéquat et ainsi être intégrées au sein d'une ontologie afin d'être réutilisées pour la résolution de problèmes dans des domaines tels que l'agronomie, la biologie, la chimie, la médecine, etc.

Il existe plusieurs livres permettant de mieux comprendre les principes de l'ECBD et l'utilisation des méthodes d'ECBD, comme par exemple [FPSSU96, MBK98] et plus récemment [HK01, HMS01, Dun03], ainsi que [WF00] qui est associé au système Weka[45].

[45]`http://www.cs.waikato.ac.nz/~ml/weka/`

La fouille de données, étape centrale de l'ECBD

La fouille de données est l'étape centrale du processus d'ECBD. Le processus d'ECBD est constitué de plusieurs éléments : la sélection des données, le pré-traitement des données, leur éventuelle transformation, la mise en œuvre de la fouille pour extraire des motifs et des relations, et enfin, l'interprétation et l'examen des structures découvertes.

Les progrès concernant l'acquisition numérique de données et les technologies de stockage ont conduit à la création d'énormes bases de données. En conséquence, un intérêt croissant est apparu concernant la possibilité d'extraire des informations, qui pourrait être utiles. La discipline liée à cette tâche est connue sous le nom de *fouille de données.* Hand *et al.* définissent la fouille de données, dans [HMS01], de la façon suivante : "la fouille de données est l'analyse de grandes ensembles de données, afin d'y trouver des relations non suspectées et de résumer les données d'une nouvelle façon qui soit à la fois compréhensible et utile."

La fouille de données est typiquement appliquée à des données qui ont été collectées dans un autre but que celui de la fouille. Cela signifie que les objectifs de la fouille ne jouent pas de rôle dans la stratégie de collection de ces données. Pour cette raison, la fouille de données est souvent désignée comme une analyse "secondaire" de données.

La définition mentionne aussi que l'ensemble de données examiné en fouille de données est souvent grand. Quand d'importants volumes de données doivent être pris en compte, de nouveaux problèmes apparaissent. Certains de ces problèmes sont liés à des considérations fondamentales telles que le choix de la stratégie de gestion des données, la façon d'analyser les données en un temps raisonnable ou la façon de filtrer le bruit des données. Souvent, les données ne comprennent qu'une partie de la population, l'objectif pouvant être de *généraliser* à la population à partir de l'extrait. Par exemple, il peut être intéressant de prédire la façon la plus probable dont les futurs clients réaliseront leurs achats. Parfois, il peut être utile de résumer ou de *compresser* un important volume de données de façon à ce que le résultat soit plus compréhensible, sans notion de généralisation. Cela peut être le cas, par exemple, si l'on dispose des données d'un recensement complet pour un pays particulier ou une base de données enregistrant les détails de millions de transactions commerciales.

La fouille de données ne doit pas être vue comme un simple exercice réalisé une seule fois. D'énormes collections de données peuvent être analysées et examinées selon un nombre illimité de manières. Au fil du temps, de nouveaux types de structures ou de motifs peuvent attirer l'attention et devenir intéressants à rechercher dans les données.

La fouille de données a, pour de bonnes raisons, beaucoup attiré l'attention : c'est une nouvelle technologie, dédiée à de nouveaux problèmes, amenant potentiellement à des découvertes, utiles autant à des applications commerciales qu'à la recherche scientifique [HMS01].

Méthodes pour l'ECBD

Un exemple introductif. Tout d'abord, examinons ce que l'on peut attendre d'une application des méthodes de fouilles de données. Considérons un tableau binaire $\mathtt{M_{ij}}$, aussi appelé *contexte formel*, où les lignes représentent des *clients* et les colonnes les *produits* achetés par les clients (voir la Table 8.1) : $\mathtt{M_{ij}} = \mathtt{1}$ quand un client `i` a acheté un produit `j`. Dans le cas d'applications réelles, ce type de tableau peut contenir plusieurs milliers de colonnes et des millions de lignes... A partir de ce contexte formel, les unités suivantes peuvent être extraites :

Clients/Produits	chips	moutarde	saucisse	boissons	bière
C_1	1	0	0	0	1
C_2	1	1	1	1	1
C_3	1	0	1	0	0
C_4	0	0	1	0	1
C_5	0	1	1	1	1
C_6	1	1	1	0	1
C_7	1	0	1	1	1
C_8	1	1	1	0	0
C_9	1	0	0	1	0
C_{10}	0	1	1	0	1

Table 8.1: Un exemple de matrice binaire représentant des transactions entre des clients (C) et des produits (P).

- L'ensemble $\texttt{X} = \{\texttt{biere}, \texttt{saucisse}, \texttt{moutarde}\}$ apparaît avec une fréquence $\phi(\texttt{X}) = 4$, c'est-à-dire que quatre individus sur dix ont acheté ces produits en même temps. De la même façon, l'ensemble $\texttt{Y} = \{\texttt{biere}, \texttt{saucisse}\}$ apparaît avec une fréquence $\phi(\texttt{Y}) = 6$. L'ensemble X (respectivement Y) peut être interprété par le fait que 40% (respectivement 60%) des clients achètent les produits de l'ensemble X (respectivement de Y).

- De plus, la règle $\texttt{R} = \{\texttt{biere}, \texttt{saucisse} \rightarrow \texttt{moutarde}\}$ peut être extraite des ensembles X et Y ($\texttt{Y} \rightarrow \texttt{X} \setminus \texttt{Y}$, où $\texttt{X} \setminus \texttt{Y}$ correspond à l'ensemble X auquel est retiré Y), avec un indice de confiance de 0.66 (66.6%), indiquant que si un client achète de la biere et des saucisses, la probabilité que ce même client achète de la moutarde est de 0.66 (parmi 6 clients achetant de la bière et des saucisses, 4 clients achètent aussi de la moutarde).

  Du point de vue de l'analyste, les ensembles X et Y, ainsi que la règle R, peuvent être interprétés et validés comme des unités de connaissances extraites des données.

Méthodes de fouille de données. Le processus d'extraction est fondé sur des *méthodes de fouille de données* produisant des unités de connaissances à partir des données considérées. Les méthodes de fouille de données peuvent être soit symboliques soit numériques :

- Les méthodes symboliques incluent principalement : la classification par arbres de décision, la classification par treillis, la recherche de motifs fréquents et l'extraction de règles d'association, les méthodes d'apprentissage, comme l'induction, l'apprentissage à partir d'instances, et les méthodes utilisant la recherche d'information...

- Les méthodes numériques incluent principalement : les statistiques et l'analyse de données, les modèles de Markov cachés d'ordres 1 et 2 (conçus initialement pour la reconnaissance de formes), les réseaux bayésiens, les réseaux de neurones, les algorithmes génétiques...

Ces méthodes sont dépendantes des domaines de recherche en lien avec le processus d'ECBD [Man97] :

- *Statistiques et analyse de données* : le but est similaire, mais le processus d'ECBD nécessite dans la plupart des cas la combinaison de différentes méthodes, symboliques ou numériques, et des connaissances du domaine pour l'interprétation des unités extraites.

- *Gestion de bases de données* : les techniques de gestion de bases de données peuvent être utiles pour aider à résoudre les problèmes liés à la fouille de données, en utilisant par exemple des requêtes pour préparer les données à fouiller.

- *Apprentissage* : les méthodes d'apprentissage sont au centre du processus d'ECBD, même si les quantités de données impliquées, ainsi que les objectifs, c'est-à-dire l'utilisation des résultats pour la résolution de problèmes ou la prise de décisions, sont différents.

- *Représentation des connaissances et raisonnement* : le processus de fouille de données peut être guidé par un modèle — une ontologie du domaine — pour l'interprétation et la résolution de problèmes.

Le processus d'ECBD peut être considéré comme une sorte de "processus d'apprentissage supervisé" — considérant qu'un analyste contrôle et guide le processus d'ECBD. L'analyste peut s'appuyer sur ses propres connaissances, ainsi que sur des ontologies du domaine, pour produire une interprétation des résultats et les valider. De cette façon, les résultats du processus d'ECBD peuvent être réutilisés pour enrichir les ontologies existantes, montrant ainsi que la représentation des connaissances et l'ECBD sont deux tâches complémentaires : *pas de fouille de données sans connaissances du domaine !*

Vue globale de la thèse

Le sujet principal de cette thèse est *l'extraction de connaissances dans les bases de données* (ECBD). Plus précisément, nous avons étudié deux des plus importantes tâches d'ECBD actuelles, qui sont l'extraction de motifs et la génération de règles d'association. Tout au long de notre travail, notre objectif a été de trouver des règles d'associations *intéressantes* selon plusieurs points de vue : dans un but de fouille efficace, pour réduire au minimum l'ensemble des règles extraites et pour trouver des unités de connaissances intelligibles (et facilement interprétables). Pour atteindre ce but, nous avons développé et adapté des algorithmes spécifiques.

Les contributions principales de cette thèse sont : **(1)** nous avons développé et adapté des algorithmes pour trouver les règles d'association minimales non redondantes ; **(2)** nous avons défini une nouvelle base pour les règles d'associations appelée "règles fermées" ; **(3)** nous avons étudié un champ de l'ECBD important mais relativement peu étudié, à savoir l'extraction des motifs rares et des règles d'association rares ; **(4)** nous avons regroupé nos algorithmes et une collection d'autres algorithmes ainsi que d'autres opérations auxiliaires d'ECBD dans une boîte à outil logicielle appelée CORON.

Nous présentons maintenant un résumé un peu plus détaillé des points forts de la thèse.

Motifs fréquents. Dans le Chapitre 3 nous présentons deux algorithmes spécifiquement adaptés pour extraire des règles d'association minimales non-redondantes ($\mathcal{MNR}$). Cet ensemble de règles est obtenu sans perte d'information, et représente de manière informative toutes les règles d'association valides. Notre premier algorithme, *Zart*, est une extension de *Pascal* qui est probablement l'algorithme le plus efficace de recherche des motifs fréquents par niveau. En plus des capacités de *Pascal*, *Zart* est capable d'identifier l'ensemble des motifs fermés fréquents et de leurs associer leurs générateurs. Nous montrons que ces données supplémentaires fournies par *Zart* sont essentielles pour l'extraction de règles $\mathcal{MNR}$. Dans notre second algorithme, *Eclat-Z*, nous allons plus loin et montrons comment généraliser l'idée de *Zart* pour *n'importe quel* algorithme d'extraction des motifs fréquents. Il est ainsi possible d'étendre n'importe quel algorithme d'extraction des motifs fréquents afin de rajouter le support de l'extraction des règles $\mathcal{MNR}$.

Dans ce même chapitre nous proposons une extension simple de *Charm*, appelé *Charm-MFI*. *Charm-MFI* filtre les motifs fréquents maximaux parmi les motifs fréquents fermés. Cet algorithme a été utilisé pour étudier la frontière entre fréquents et non fréquents.

Règles d'association fréquentes. Le Chapitre 4 présente différents ensembles de règles d'association fréquentes, les règles valides et la famille des règles minimales non-redondantes. Nous introduisons aussi une nouvelle base, les règles fermées, qui se positionne entre les deux ensembles de règles d'association mentionnés précédemment, comblant l'écart entre ces deux ensembles. L'ensemble des règles fermées est une représentation concise de toutes les règles valides, pour lequel il y seulement besoin de calculer les motifs fermés fréquents. Les règles fermées semblent être une bonne alternative à l'ensemble des règles valides.

Dans la littérature, la plupart des algorithmes se concentrent seulement sur les valeurs de support et de confiance des règles. Nous montrons comment calculer d'autres indices statistiques qui recquièrent, en plus, la valeur du support pour la partie droite des règles. Dans notre travail, pour la représentation condensée des motifs fermés, nous utilisons des motifs fermés fréquents qui sont stockés dans une structure de données de type "trie". Dériver le support des deux parties des règles nécessite un nombre important d'opérations sur le "trie". Pour pallier ce problème, nous proposons un mécanisme de cache utilisant une table de hachage, qui se montre une solution efficace. L'utilisation d'un cache est avantageuse même si d'autres mesures d'intérêt ne sont pas requises et ainsi les techniques de cache peuvent aussi être incorporées dans les autres algorithmes.

Motifs rares et règles d'association rares. Le Chapitre 5 est un des chapitres les plus originaux de cette thèse. Dans ce chapitre, nous nous intéressons aux problèmes de l'extraction des motifs rares et de la génération de règles d'associations rares. Dans la littérature, ces problèmes n'ont jusqu'alors pas été étudiés en détail, bien que les motifs rares puissent contenir des informations importantes dans la même mesure que les motifs fréquents. En particulier, l'application aux diagnostics médicaux nous paraît un champ d'application particulièrement adapté.

Dans le Chapitre 5.1 nous présentons une méthode permettant de trouver tous les *motifs rares*. Pour ce faire, nous utilisons l'algorithme classique *Apriori*. *Apriori* est connu pour trouver l'ensemble des motifs fréquents, mais il trouve également un ensemble spécial des motifs rares : les motifs rares minimaux (MRM). Une légère modification d'*Apriori*, que nous appelons *Apriori-Rare*, permet ainsi de conserver l'ensemble des MRM. Nous montrons ensuite comment reconstruire l'ensemble de tous les motifs rares à partir des MRM tout en évitant les motifs de support 0.

Dans le Chapitre 5.3 nous allons plus loin en montrant comment générer des *règles d'association rares* valides. Notre travail est motivé par la question ouverte de longue date visant à construire un algorithme efficace pour la découverte de règles à support faible et confiance élevée. Afin de trouver de telles règles en utilisant des algorithmes conventionnels de recherche des motifs fréquents comme *Apriori*, le support minimal doit être fixé à un seuil très faible, ce qui augmente de manière drastique le temps d'exécution de l'algorithme. De plus, lorsque le support minimal est fixé très bas, *Apriori* produit un grand nombre des motifs fréquents. Ce problème est également connue sous le nom de *problème des motifs rares*. Pour résoudre ce problème bien connu, nous proposons une solution. Avec notre méthode nous pouvons extraire un ensemble des règles d'associations rares exactes (nous appelons de telles règles "règles MRG exactes"). Nous montrons également comment extraire des règles MRG approximatives ; néanmoins leur intérêt parait limité. De ce fait, nous nous concentrons plutôt sur les règles rares exactes. De plus, de telles règles sont non-redondantes car l'antécédent est minimal et le conséquent maximal, impliquant ainsi que parmi les règles de support et confiance identiques, celles-ci contiennent le plus d'information.

La boîte à outils Coron. Les algorithmes présentés dans cette thèse ont été implémentés et regroupés dans une plate-forme logicielle unifiée appelée CORON. CORON est une boîte à outils de fouille de données indépendante du domaine et de l'architecture utilisés. Non seulement CORON incorpore une riche collection d'algorithmes de fouille de données mais CORON permet également un grand nombre d'opérations auxiliaires. À notre connaissance, aucun autre logiciel n'a été conçu spécifiquement pour l'extraction de motifs et la génération de règles d'association. CORON fournit également un support pour la préparation, le filtrage des données ainsi que pour l'interprétation des unités de connaissances extraites.

La plupart des expériences avec CORON ont été réalisées sur de véritables bases de données biomédicales appelées la cohorte STANISLAS. Durant ces expériences, nous avons réalisé qu'il nous était nécessaire d'avoir **(1)** une méthodologie pour la fouille et **(2)** un outil permettant de l'implanter. Le Chapitre 6 présente notre méthodologie globale de fouille de données, pouvant être généralisée à n'importe quel ensemble des données. Cette méthodologie peut être utilisée pour les recherches des motifs aussi bien fréquents que rares.

A la fin du Chapitre 6, en plus de la cohorte STANISLAS, nous présentons trois autres projets ayant utilisé CORON avec succès.

Organisation de la thèse

Dans le **Chapitre 2**, nous commençons par présenter l'état de l'art. Nous nous focalisons principalement sur les méthodes d'ECBD symboliques s'appuyant sur l'opération de classification, la recherche de motifs fréquents et l'extraction de règles d'association. Nous montrons comment l'ensemble du processus, des données brutes aux unités de connaissances, est fondé sur le principe sous-jacent de classification.

Dans le **Chapitre 3**, nous approfondissons les concepts et algorithmes de recherche de motifs fréquents. Nous proposons des algorithmes que nous avons spécifiquement adaptés à nos besoins, c'est-à-dire pour trouver les règles d'association minimales non-redondantes et pour étudier la frontière. L'annexe B présente des algorithmes classiques en relation avec ce chapitre.

Dans le **Chapitre 4**, nous nous intéressons à l'extraction de différents ensembles de règles d'association fréquentes : les règles valides, les règles fermées et la famille des règles minimales non-redondantes.

Dans le **Chapitre 5**, nous traitons le problème de l'extraction de motifs rares et la génération de règles d'association rares. Ceci constitue l'une des parties les plus importantes de cette thèse.

Dans le **Chapitre 6**, nous présentons une méthodologie de fouille de données globale pour fouiller à la fois les règles rares et fréquentes. La deuxième partie du chapitre décrit la plate-forme CORON qui implémente cette méthodologie dans son intégralité.

Finalement, le **Chapitre 7** résume les principales contributions de la thèse et suggère des perspectives pour des travaux futurs.

Il doit être remarqué que les annexes sont aussi une part importante de la thèse. Parmi d'autres choses, nous presentons des algorithmes classiques, une structure de données efficace pour les motifs, différentes techniques d'optimisation et un guide d'utilisation détaillé de la plate-forme CORON.

8.1.2 Une méthodologie pour la fouille et la boîte à outils Coron

Les données biomédicales collectées dans les cohortes sont relativement complexes à analyser de par leur diversité et leur taille. Dans cette sous-section nous introduisons une méthodologie guidée par l'expert qui permet la fouille de données issues de cohortes. Les expériences ont été menées avec CORON, notre plate-forme qui propose différentes méthodes symboliques de fouille de données pour la recherche de motifs fréquents et l'extraction de règles d'association. Les données réelles utilisées proviennent de la cohorte STANISLAS, une étude sur dix ans qui collecte des données recueillies sur des familles françaises supposées saines. Dans le contexte de cette étude, l'expert est intéressé par des motifs et des règles associant des facteurs de risques cardiovasculaires et des polymorphismes génétiques. Les résultats obtenus montrent les capacités de la méthodologie de fouille de données dans la suggestion de nouvelles hypothèses, à tester ensuite par de nouvelles études en biologie.

Introduction

Une étude de cohorte consiste à suivre une population donnée pendant une période de temps et à collecter différentes données concernant cette population [BHC+95]. Les données issues de telles études présentent un degré élevé de complexité. D'une part, ces données sont des données longitudinales, c'est à dire qu'elles peuvent varier dans le temps. D'autre part, ces études recrutent un grand nombre d'individus, et recueillent une quantité importante de paramètres différents. De plus, ces données sont de différents types : quantitatives, qualitatives, textuelles, booléennes, etc. Enfin, le recueil des données étant fait, certaines valeurs de variables peuvent être manquantes ou bruitées, conduisant à des bases de données incomplètes. Toutes ces caractéristiques montrent que les données issues de cohortes peuvent être considérées comme des données complexes. Par conséquent, l'exploitation de telles données s'avère être une tâche non triviale.

La cohorte STANISLAS illustre la complexité des données de cohortes. C'est une étude sur dix ans qui a été mise en place afin d'étudier les facteurs de risque cardiovasculaire. Des familles d'origine française supposées saines ont été recrutées pour cette étude (données familiales, avec des sujets apparentés et non apparentés, de différentes tranches d'âges). Les données de cette cohorte sont de types variés (cliniques, biologiques, environnementales et génétiques) et ont été recueillies lors de trois visites séparées par cinq ans (données longitudinales).

Les experts impliqués dans l'étude de la cohorte STANISLAS sont des spécialistes du domaine cardiovasculaire et sont intéressés par la découverte de nouvelles associations liant un ou plusieurs polymorphismes génétiques à des facteurs de risque cardiovasculaire. Afin d'atteindre cet objectif, des analyses statistiques sont classiquement utilisées par les biologistes. Cependant, de telles analyses sont valides uniquement si une hypothèse a priori est donnée. Face à la quantité et à la complexité des données, les experts demandent des méthodes alternatives qui pourraient les aider à formuler de nouvelles hypothèses de recherche.

Les techniques symboliques de fouille de données pourraient être ces méthodes alternatives. D'ailleurs, certaines d'entre elles ont déjà été utilisées en biologie [BBJ+02, CCH03, CH03, QTDDB02, STVN04]. Dans cette sous-section, nous présentons une expérimentation des méthodes de recherche de motifs fréquents et d'extraction de règles d'association dans la base de données de la cohorte STANISLAS. Nous avons utilisé l'algorithme *Zart* qui est implémenté dans CORON, qui est présenté dans cette thèse [SN05]. Dans le cas de la cohorte STANISLAS, l'objectif est de découvrir des motifs fréquents et des règles d'association qui lient des variables biologiques et des polymorphismes génétiques. Un polymorphisme génétique correspond à une variation dans la séquence de l'ADN présente chez au moins 1% de la population, nous sommes donc intéressés par la recherche des motifs à support faible.

Nous présentons une méthodologie globale de fouille de données de cohortes, qui suggère de nouvelles hypothèses à tester en biologie et qui ouvrent la porte à de nouvelles directions de recherche. Après avoir présenté la méthodologie globale de fouille de données et la plate-forme CORON, nous décrivons brièvement les données réelles de la cohorte STANISLAS. Nous détaillons ensuite une expérimentation menée sur les données de la cohorte qui met en œvre la méthodologie, puis nous discutons les résultats obtenus et l'intérêt de la méthodologie. Enfin, nous terminons par nos conclusions et perspectives de travail.

La méthodologie globale de fouille de données

La méthodologie que nous décrivons a été initialement introduite dans [MNSVS05b] et a été mise au point pour fouiller les données issues de cohortes en biologie. Toutefois, cette méthodologie peut très facilement être adaptée à n'importe quel autre type de base de données. Une chose importante à remarquer est que tout le processus de fouille décrit dans cette méthodologie est guidé par l'expert, qui est un spécialiste du domaine d'analyse (la biologie en l'occurrence). Son rôle est crucial, en particulier dans la sélection des données et l'interprétation des résultats, qui permettront de transformer les modèles extraits en unités de connaissance. CORON est conçu pour correspondre à notre méthodologie et offre donc tous les outils nécessaires à sa mise en oeuvre dans une plate-forme unique. Cependant, si l'utilisateur préfère tester d'autres outils que ceux de CORON, il peut quand même s'appuyer sur la méthodologie que nous décrivons ci-après.

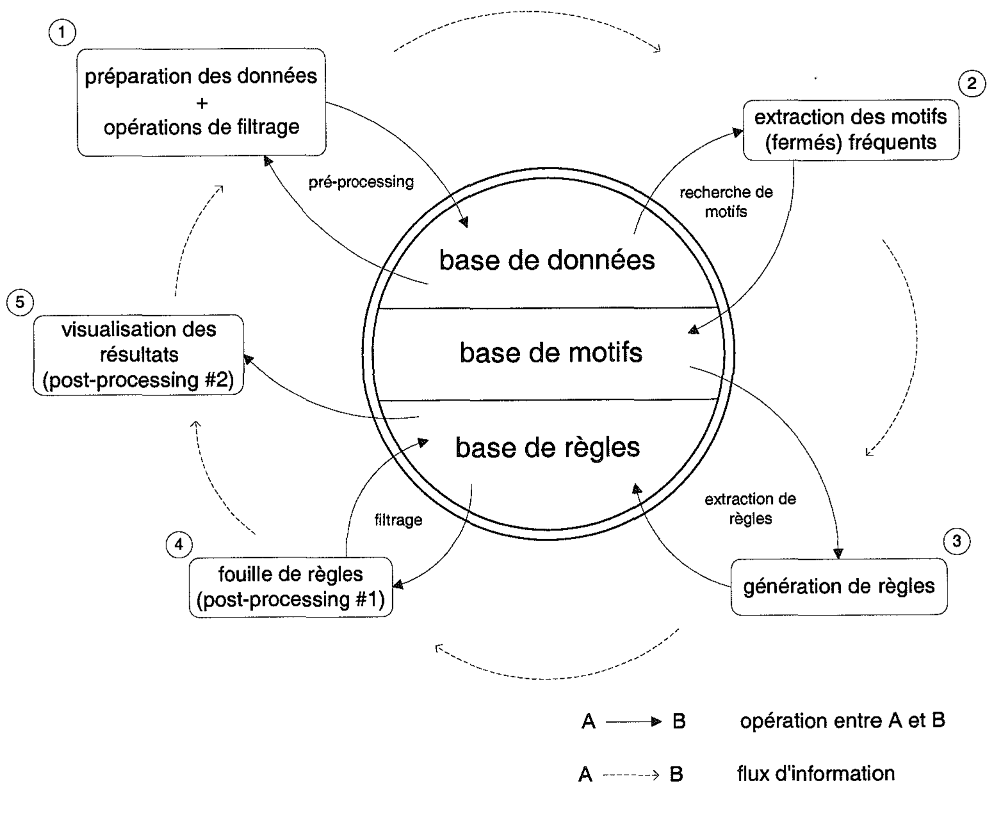

1
préparation des données + opérations de filtrage
pré-processing
2
extraction des motifs (fermés) fréquents
recherche de motifs
base de données
base de motifs
base de règles
5
visualisation des résultats (post-processing #2)
extraction de règles
3
génération de règles
filtrage
4
fouille de règles (post-processing #1)
A → B opération entre A et B
A ⇢ B flux d'information

Figure 8.2: Cycle de vie de la méthodologie de fouille symbolique de données.

La méthodologie se compose de cinq étapes résumées dans la Figure 8.2 :

(1) – Définition du cadre d'étude
(2) – Préparation et nettoyage des données
(3) – Étape itérative :
+ Pré-traitement : le filtrage des données
+ Application des procédures de fouille de données
+ Post-traitement (fouille de règles et visualisation des résultats)
(4) – Interprétation et validation des résultats
(5) – Génération de nouvelles hypothèses de recherche

CORON : une plate-forme dédiée à la mise en œuvre de la méthodologie

CORON [SN05] est une plate-forme logicielle intégrée qui fédère des modules dédiés à la recherche de motifs (fermés) fréquents dans des contextes binaires ainsi qu'à l'extraction de différents types de règles d'association, opérations clés pouvant être utilisées dans des applications pratiques. CORON est conçu pour couvrir un large éventail de tâches en fouille de données incluant le pré-traitement de l'ensemble des données, l'extraction de motifs fréquents, la génération de règles d'association à partir de ces motifs et le filtrage des règles selon différentes conditions. CORON offre une architecture ouverte ainsi que des implémentations génériques, ce qui facilite son adaptation à un domaine d'application particulier pour un problème spécifique. CORON s'accommode de différents formats de données, comme par exemple les données binaires ou les données Relational Context Family (ce format inclut un ensemble de contextes formels et un ensemble de relations inter-objets). De plus, de nombreux algorithmes de fouille symbolique de données sont inclus dans l'architecture du système. La conception de CORON est menée parallèlement au développement de GALICIA[46], une plate-forme mise en œuvre pour l'analyse de concepts formels.

Extraction de motifs (fermés) fréquents. La partie centrale de CORON (module Coron-base) contient un ensemble d'algorithmes d'extraction de motifs (fermés) fréquents. Les algorithmes actuellement implémentés sont les suivants : *Apriori*, *Apriori-Close*, *Close*, *Pascal*, *Titanic*, *Zart*, *Eclat*, *Charm* et *Arima*. *Apriori* [AMS$^+$96] est à la base de la plupart des algorithmes de recherche de motifs. C'est un algorithme de recherche par niveau qui permet d'extraire les motifs fréquents. *Apriori-Close* [PBTL99a] est une extension d'*Apriori* permettant le calcul simultané des motifs fréquents et fermés fréquents. *Close* [PBTL99c] recherche uniquement les motifs fermés fréquents et se montre plus efficace sur les bases de données denses. *Pascal* [BTP$^+$00a, BTP$^+$02] introduit la notion de motifs générateurs et démontre que le support de nombreux motifs fréquents peut être inferé par leur générateurs, réduisant ainsi de façon importante le nombre de passages dans la base de données. *Titanic* [STB$^+$02] utilise comme *Pascal* les motifs générateurs pour rechercher les motifs fermés fréquents. *Zart* [SNK05] est une extension de *Pascal* pouvant identifier en parallèle les motifs fermés fréquents et les générateurs des motifs fermés. Le résultat en sortie de *Zart* peut être utilisé directement pour trouver des règles d'association informatives. A la différence des algorithmes présentés ci-dessus et qui traitent la base de données de façon "horizontale" (priorité aux lignes), *Eclat* [ZPOL97] et *Charm* [ZH02] traitent la base de façon "verticale" (priorité aux colonnes). *Eclat* et *Charm* recherchent respectivement les motifs fréquents et les motifs fermés fréquents. Enfin depuis peu, CORON, grâce

[46]`http://galicia.sourceforge.net`

à l'algorithme *Arima*, permet également la recherche de motifs rares qui peuvent s'avérer très pertinents dans les domaines de la médecine et de la biologie [SMP+06].

Extraction de règles d'association. AssRuleX (Association Rule eXtractor) est un module de Coron qui utilise les motifs précédemment extraits pour générer les règles d'association correspondantes. Si l'on souhaite générer la totalité des règles d'association possibles à partir des motifs fréquents, le nombre de règles produites est très élevé et beaucoup de règles sont redondantes. Pour cette raison, il est préférable de travailler sur les règles d'association informatives définies dans [BTP+02]. Une règle d'association informative a la forme suivante : $P \rightarrow Q \setminus P$ où P est un générateur, Q est un motif fermé fréquent et P est un sous-ensemble propre de ce motif fermé fréquent ($P \subset Q$). Étant donné qu'un générateur est un sous-motif minimal du motif fermé de même support, les règles informatives permettent de déduire le maximum d'information avec une hypothèse minimale. Les règles d'association informatives forment un ensemble minimal de règles d'association non-redondantes où chaque règle possède un antécédent minimal et un conséquent maximal. Parmi les règles présentant un support et une confiance identiques, les règles d'association informatives sont celles qui contiennent le plus d'information et qui se révèlent être les plus utiles en pratique [Pas00b].

A côté des indices de support et de confiance qui sont classiquement donnés pour une règle d'association, Coron permet de calculer d'autres mesures de qualité qui ont été décrites dans [CNT05] : l'intérêt (ou lift), la conviction, la dépendance, la nouveauté et la satisfaction.

Grâce à Coron, il est à l'heure actuelle possible de générer l'ensemble des règles d'association, l'ensemble des règles d'association informatives, les règles d'association informatives réduites, les règles d'association informatives exactes, les règles d'association informatives approximatives et les règles d'association informatives approximatives réduites.

Pré-traitement des données. Il existe deux modules pour le pré-traitement des données dans Coron. Le premier module, Filter-DB, permet de filtrer les lignes et les colonnes de la base de données en entrée. Les différentes opérations de filtrage possibles correspondent aux opérations de la théorie des ensembles (complément, union et intersection) : application de restrictions et de projections **(i)** sur les lignes : sélection des individus possédant un ou plusieurs attributs choisis par l'expert, **(ii)** sur les colonnes : sélection (ou suppression) de certains attributs. Le résultat du filtrage est considéré comme un nouvel ensemble de données sur lequel les procédures de fouille de données sont à nouveau exécutées.

Le second module de pré-traitement est Filter-Compl, qui permet de trouver le complémentaire d'un ensemble d'objets satisfaisant une règle, défini par l'ensemble des objets qui ne satisfont pas cette règle.

Post-traitement des données. Le filtrage (ou fouille) des règles ainsi que leur visualisation permettent de repérer les règles mettant en jeu les attributs les plus intéressants pour l'expert. Deux modules sont disponibles dans Coron pour le post-traitement des données : RuleMiner et LeCo.

Grâce au sous-module Filter-Rules de RuleMiner, il est possible, à partir de l'ensemble des règles d'association générées avec AssRuleX, de conserver ou de supprimer selon le cas les règles satisfaisant ou non certains critères. Différents choix peuvent être faits à cette étape : **(i)** choix des règles possédant une forme particulière (par exemple, les règles ayant un antécédent unique), **(ii)** choix des règles possédant un attribut intéressant pour l'expert dans l'antécédent, dans le conséquent, ou indifféremment dans une des deux parties de la règle.

L'étape de fouille de règles peut être dépendante de certaines mesures numériques telles que le support ou la confiance, ou bien de la connaissance du domaine comme l'ont montré des expérimentations récentes [JCK$^+$04]. L'expert joue donc un rôle crucial dans l'analyse. Un autre sous-module de RULEMINER permet **(i)** le classement des règles extraites par ordre (dé)croissant selon les valeurs de leur support, de leur confiance, ou encore d'un autre indice statistique tels que ceux précédemment décrits, **(ii)** la sélection des règles dont le support appartient à un intervalle donné $[a,b]$; de retourner les règles dont le support est inférieur (ou supérieur) ou égal à une valeur c donnée. Ces sélections peuvent être faites avec les autres indices statistiques cités ci-dessus.

RULEMINER contient également un sous-module qui permet de spécifier une couleur pour chaque attribut. Ainsi, il est très facile pour l'utilisateur de suivre le ou les attributs qui l'intéressent.

En ce qui concerne la recherche de motifs fréquents qui est conduite pour trouver des motifs fréquents à support faible, les treillis de concepts peuvent s'avérer être un outil de choix [Jay03]. Avec le module LECO, CORON permet l'utilisation d'un outil de visualisation adapté aux méthodes symboliques de fouille de données. En effet, étant donné que les motifs fermés fréquents peuvent être visualisés avec les treillis de Galois [GW99], LECO utilise en entrée les motifs fermés fréquents qui sont les intensions des concepts formels. Après avoir calculé les extensions, LECO ordonne les concepts formels et peut ainsi construire le treillis de Galois.

Le contexte biologique : la cohorte STANISLAS

La cohorte STANISLAS (Suivi Temporaire Annuel Non Invasif de la Santé des Lorrains Assurés Sociaux) est une étude familiale qui a été lancée en 1993 au centre de Médecine Préventive de Vandœuvre-lès-Nancy. Son objectif principal est d'étudier le rôle et la contribution de facteurs génétiques et environnementaux sur la fonction cardiovasculaire [SVH$^+$98]. C'est une étude longitudinale sur dix ans, où des familles de la Meurthe-et-Moselle et des Vosges ont été invitées par la caisse primaire d'assurance maladie à venir passer un examen de santé tous les cinq ans. Lors du recrutement initial (1993–1995, t_0), les critères d'inclusion étaient les suivants : familles supposées saines, exemptes de maladies aiguës et/ou chroniques, composées de deux parents et d'au moins deux enfants biologiques de plus de six ans. 1006 familles (4295 sujets) ont ainsi pu être recrutées. Lors de la deuxième visite (1998–2000, t_{+5}), 75% des familles sont revenues. La troisième visite (2003–2005, t_{+10}) est actuellement en cours de réalisation. Les données collectées sont de différents types : **(1)** données cliniques (taille, poids, pression sanguine, ...) ; **(2)** données environnementales (habitudes de vie, activité physique, histoire médicale, prise de médicaments) ; **(3)** données biologiques (dosages du glucose, du cholestérol, de l'insuline, ...) ; **(4)** données génétiques : pour chaque individu sont déterminés les génotypes correspondant à 116 polymorphismes génétiques (ou SNPs pour Single Nucleotide Polymorphisms) correspondant à différents processus métaboliques impliqués dans les maladies cardiovasculaires [SAC$^+$99, CWS$^+$04].

Par conséquent, les données de la cohorte STANISLAS ont différents aspects à prendre en compte ainsi qu'un large éventail de valeurs possibles, ce qui explique leur complexité.

Le détail d'une expérimentation : mise en œuvre de la méthodologie sur les données de la cohorte STANISLAS

Définition du cadre d'étude. L'expert définit un domaine spécifique d'étude pour l'analyse. Dans le cas de la base "STANISLAS", il choisit de travailler sur des données biologiques, génétiques, ou les deux, de travailler sur des individus non apparentés ou sur des familles, de se concentrer sur un réseau métabolique ou bien sur un syndrome en particulier.

Le cadre d'étude choisi dans la présente expérimentation est la génétique du syndrome métabolique. Le syndrome métabolique regroupe un ensemble de facteurs de risque cardio-vasculaire et est essentiellement caractérisé par l'insulino-résistance, l'intolérance au glucose, la dyslipidémie, l'hypertension et l'obésité. Afin d'étudier la génétique du syndrome métabolique dans la cohorte STANISLAS, nous avons sélectionné les variables suivantes :
– les attributs biologiques représentés par le sexe (homme ou femme) et les cinq variables biologiques utilisées par le NCEP ATP-III pour définir le syndrome métabolique : concentration en glucose plasmatique, pression sanguine, concentration en triglycérides, concentration en HDL-cholestérol et tour de taille [oH01],
- tous les polymorphismes génétiques disponibles dans la base de données STANISLAS, ce qui représente 101 polymorphismes génétiques.

Trois cent huit adultes ont été génotypés pour tous les polymorphismes génétiques. La base de données de travail, notée "BDSM" sur laquelle les expérimentations ont été réalisées est donc composée de 308 individus et 235 attributs.

Préparation et nettoyage des données. Le nettoyage des données comprend tout d'abord la détection ainsi que la possibilité de suppression de valeurs incomplètes et/ou extrêmes. Par ailleurs, différentes actions de conversion des données peuvent également être réalisées à cette étape en fonction des besoins :
– *addition/création* de nouveaux attributs afin d'aider à l'extraction de règles d'association par combinaison d'attributs,
– *suppression* des attributs non pertinents dans le cadre d'étude choisi. Cette option est proche des projections et restrictions décrites plus haut,
– *discrétisation* : transformation des données continues en données booléennes, soit en utilisant un seuil ("valeur de référence") défini dans la littérature, soit en divisant chaque variable continue en quartiles.

Préparation des données biologiques de la base BDSM. Les données biologiques de la cohorte STANISLAS sont recueillies dans une base de données Microsoft Access. Toutes les données biologiques choisies pour l'expérimentation sont continues (mis à part le sexe qui est une donnée discrète). Étant donné que l'algorithme *Zart* accepte uniquement des données discrètes, les données biologiques ont été discrétisées. Deux types de discrétisation sont testées. Pour la première, chaque variable continue est discrétisée en utilisant les seuils proposés par le NCEP. En effet, si l'on se réfère aux critères du NCEP ATP-III [oH01], un individu présente un syndrome métabolique s'il possède au moins trois des cinq critères suivants : tour de taille $>$ 102 cm chez les hommes et $>$ 88 cm chez les femmes ; taux de triglycérides $\geq$ 1.70 mmol/l ; concentration en HDL-cholestérol $<$ 1.04 mmol/l chez les hommes et $<$ 1.30 mmol/l chez les femmes ; pression sanguine $\geq$ 130/85 mmHg ; concentration en glucose $\geq$ 6.1 mmol/l. Les données biologiques de départ ont dont été converties en cinq variables discrètes : obésité (tour de taille élevé), hypertriglycéridémie, hypoHDLémie, hypertension et hyperglycémie.

Pour le second type de discrétisation, chaque variable continue est divisée en quatre attributs discrets en utilisant des seuils obtenus par le calcul des quartiles des distributions des valeurs des variables. De plus, des variables ont été ajoutées pour des raisons de calcul : la variable syndrome métabolique ("SM") définie par les critères du NCEP, la variable "non SM" dont la valeur est égale à 1 si l'individu ne présente pas le syndrome métabolique, les variables "glucose normal", "HDL normal", "pression sanguine normale", "triglycérides normaux", "tour de taille normal", correspondant à la négation des variables discrètes hyperglycémie, hypoHDLémie, hypertension, hypertriglyceridémie et obésité.

Préparation des données génétiques de la base BDSM. Les données génétiques de la cohorte STANISLAS sont recueillies sous un format booléen dans une base de données Microsoft Access. Chaque polymorphisme génétique est représenté par ses allèles. Un polymorphisme génétique est noté A/a, A étant l'allèle fréquent et a l'allèle rare. Un gène peut exister sous différentes formes appelées allèles. Ces allèles diffèrent par des variations dans la séquence ADN. Pour un gène donné, le génotype correspond à la combinaison de deux allèles. Un individu est homozygote pour un gène s'il possède deux allèles identiques pour ce gène. Un individu est hétérozygote pour un gène s'il possède deux allèles différents pour ce gène. Deux modes de conversion différents ont été testés sur les données génétiques : **(1)** présentation des polymorphismes par génotypes : AA, Aa et aa; **(2)** présentation des polymorphismes par allèles AA et a, où a représente le regroupement des génotypes Aa et aa.

Étape itérative. Pour chaque itération, il y a une étape de pré-traitement (le filtrage des données), une étape de traitement (l'application des procédures de fouille de données), et une étape de post-traitement (fouille de règles, interprétation et visualisation des résultats). Nous pouvons réaliser toutes ces opérations avec la plate-forme CORON. Dans ce qui suit, nous décrivons une expérimentation particulière sur la base visant à caractériser le profil génétique associé au syndrome métabolique.

Pré-traitement des données de la base BDSM. Les polymorphismes génétiques sont codés par leur génotypes. Grâce à un premier filtrage, nous sélectionnons les individus présentant le syndrome métabolique dans la base BDSM, ce qui représente neuf individus. Ce chiffre peut paraître faible, mais il ne l'est pas tant que cela en regard du nombre global de 308 adultes examinés.

Application des procédures de fouille de données. L'utilisateur applique la recherche de motifs fréquents et l'extraction de règles d'association grâce au module Coron-base. A cette étape, l'expert fixe les valeurs de certains seuils, comme le support minimum et la confiance minimum pour la génération de motifs fréquents et de règles d'association. Comme la méthodologie globale décrite ici est un processus itératif et interactif, l'expert est amené à changer lors d'une itération les valeurs des seuils introduites et à mener de nouvelles expérimentations.

Dans notre analyse, seules les règles exactes sont conservées. Les neuf individus vérifient la règle *R1* (Table 8.2). Dans cette règle, tous les génotypes impliqués sont homozygotes pour l'allèle le plus fréquent, sauf un qui est hétérozygote, APOB 71Thr/Ile. Suite à ce résultat, la sélection des attributs suivants est réalisée sur la base de donnée BDSM complète : le polymorphisme APOB 71Thr/Ile, l'hyperglycémie, l'hypertriglycéridémie, l'hypoHDLémie, l'hypertension, l'obésité et le sexe. Le cadre d'étude choisi ici est le syndrome métabolique. Cet état est peu fréquent dans la cohorte STANISLAS, ceci s'expliquant par le fait que les individus qui la composent sont présumés sains. Ainsi, l'expert, qui est intéressé par la découverte de règles d'association mettant en jeu le syndrome métabolique, va orienter le processus de fouille en baissant le seuil du support minimum.

Post-traitement. Les étapes de post-traitement s'effectuent dans CORON avec le module RULEMINER. Comme le nombre de règles générées au cours de cette expérimentation est

ID règle	**Règle extraite**	**Support** ($n/\%$)	**Confiance** (%)
R1	{} ⇒ {SM, APOAI_121GG, APOAIV_347ThrThr, APOAIV_360GluGlu, ADRB3_64TrpTrp, NOS3 – 948AA, ANP_7ValVal, ENaCa_493TrpTrp, FII_20210GG, IL4R_478SerSer, ADRB2_164ThrThr, CCR3_39ProPro, APOB_71ThrIle, LPL_291AsnAsn, FV_506ArgArg, SELE_554LeuLeu}	9/100	100
R2	{SM} ⇒ {APOB_71ThrIle}	9/3	100

Table 8.2: Règles d'association extraites présentant un intérêt du point de vue de l'expert.

élevé, l'ensemble des règles extraites est lui-même fouillé afin de sélectionner les règles comportant l'attribut syndrome métabolique (SM) dans l'antécédent ou dans le conséquent. Une règle intéressante du point de vue de l'expert est *R2* (Table 8.2), car elle associe le syndrome métabolique à un génotype particulier dans notre base de données.

Interprétation et validation des résultats. *R2* s'interprète par : "présenter le syndrome métabolique implique pour un individu qu'il est hétérozygote pour le polymorphisme APOB 71Thr/Ile". Cette règle a été vérifiée et validée en utilisant des tests statistiques et a permis de conclure que la répartition des génotypes du polymorphisme de l'APOB71 diffère significativement selon qu'un individu présente ou non un syndrome métabolique (validation statistique sur une nouvelle base de données de 740 individus, test du χ^2, p=0.03).

Génération de nouvelles hypothèses de recherche. Les résultats générés permettent de suggérer de nouvelles directions de recherche. Ces nouvelles hypothèses sont testées par de nouvelles expérimentations en biologie, telles que des études d'épidémiologie génétique ou des expériences en laboratoire.

La validation effectuée précédemment suggère une nouvelle hypothèse en biologie : un sujet possédant l'allèle rare pour le polymorphisme APOB 71Thr/Ile présente plus fréquemment un syndrome métabolique. Suite à cela, une nouvelle étude d'épidémiologie génétique a été menée et a donné des résultats intéressants.

Discussion, situation par rapport à l'état de l'art

Les résultats décrits dans cette sous-section montrent que la fouille des données de la cohorte STANISLAS est une opération complexe qui met en œuvre un ensemble d'opérations variées. De nombreuses règles peuvent être extraites, mais en fait, seul un petit nombre de règles est effectivement intéressant pour l'expert du domaine. Par ailleurs, il est très important que l'expert puisse réitérer le processus de fouille autant de fois qu'il le juge nécessaire. Par conséquent, la préparation des données et l'ajustement des paramètres du système de fouille doivent être des opérations simples et faciles d'accès. Dans ces conditions, l'expert peut espérer obtenir des résultats significatifs.

Nous avons décrit comment obtenir des règles avec une forme particulière, possédant un antécédent minimal et un conséquent maximal, dans le même ordre d'idées que l'expérience décrite dans [CH03]. De plus, les règles extraites qui ont été estimées intéressantes par l'expert ont donné de nouvelles hypothèses en biologie et suggéré des directions de recherche fructueuses. L'expérience vaut donc la peine d'être poursuivie et approfondie.

Même si l'extraction de règles d'association est encore peu utilisée en biologie, il existe quelques études intéressantes [BBJ+02, CH03, QTDDB02, STVN04, SBMP01]. Cependant, peu d'études exploitent des données sur des populations telles que les données de la cohorte

STANISLAS. Creighton et Hanash [CH03] ont étudié des données d'expression des gènes de levures provenant de bases de données publiques en utilisant l'algorithme *Apriori* [AMS$^+$96] et en se concentrant sur des règles ayant un antécédent unique. Cette étude est proche de notre travail du point de vue de la méthodologie utilisée, d'une part parce qu'*Apriori* et *Zart* sont tous les deux algorithmes de recherche par niveau et d'autre part parce que nos deux études se focalisent sur des règles d'association présentant une forme particulière. Dans l'étude de Quentin-Trautvetter *et al.* [QTDDB02], le logiciel CBA[47] (Classification Based on Association) est utilisé pour l'extraction de règles d'association. Les auteurs, confrontés à un nombre de règles générées trop important pour permettre la détection de règles intéressantes, ont sélectionné un nombre limité d'attributs. Leur choix est à rapprocher de nos méthodes de projections verticales et horizontales.

Par rapport à WEKA [WF99] qui est une boîte à outils généraliste sur la fouille de données, CORON a l'avantage d'être un outil très complet pour qui veut effectuer de la recherche de motifs et de règles d'association. Dans WEKA en effet, alors que le nombre d'algorithmes implémentés est très important, il y a peu d'algorithmes d'extraction de motifs et de règles qui sont implémentés (uniquement *Apriori*). Par ailleurs, CORON propose un éventail de d'indices de qualité qui ne sont pas disponibles dans WEKA.

Conclusions et perspectives

En conclusion, l'expérimentation sur le syndrome métabolique qui a été menée ici en suivant la méthodologie pour la fouille de cohortes a donné des résultats intéressants du point de vue de l'expert. Cette méthodologie montre que la préparation des données, l'interprétation des résultats et la possibilité d'itérations successives du processus de fouille sont de première importance pour une expérimentation en biologie. De plus, le rôle de l'expert du domaine est central et déterminant : la totalité du processus de fouille de données est menée sous la supervision de l'expert.

CORON, qui permet de mettre en œuvre la méthodologie, et les outils qui lui sont associés doivent encore être améliorés. Ceci est vrai en particulier pour la préparation des données et pour la visualisation des résultats, où quelques modules de visualisation (dont LECO) doivent encore être testés. Par ailleurs, étant donné que les règles d'association peuvent également être extraites à partir des treillis de Galois, nous prévoyons d'étudier aussi ce type de génération de règles.

[47] `http://www.comp.nus.edu.sg/~dm2/`

8.1.3 L'extraction de motifs rares

Un certain nombre de travaux en fouille de données se sont intéressés à l'extraction de motifs et à la génération de règles d'association à partir de ces motifs. Cependant, ces travaux se sont jusqu'à présent centrés sur la notion de motifs fréquents. Le premier algorithme à avoir permis l'extraction de tous les motifs fréquents est *Apriori* mais d'autres ont été mis au point par la suite, certains n'extrayant que des sous-ensembles de ces motifs (motifs fermés fréquents, motifs fréquents maximaux, générateurs fréquents). Dans cette sous-section, nous nous intéressons aux motifs rares qui peuvent également véhiculer des informations importantes. Les motifs rares correspondent au complémentaire des motifs fréquents. A notre connaissance, ces motifs n'ont pas encore été étudiés en détail, malgré l'intérêt que certains domaines pourraient tirer de ce genre de modèle. C'est en particulier le cas de la médecine, où par exemple, il est important pour un praticien de repérer les symptômes non usuels ou les effets indésirables exceptionnels qui peuvent se déclarer chez un patient pour une pathologie ou un traitement donné.

Travaux en relation. L'extraction de motifs rares et la génération de règles d'association rares n'ont pas encore été étudiées en détail dans la littérature. Ici, nous partons d'une vue d'ensemble de la recherche de motifs fréquents pour introduire notre méthode d'extraction des motifs rares.

Plusieurs approches ont été proposées pour trouver les motifs fréquents dans les bases de données. La première s'appuie sur l'algorithme *Apriori*, qui fut le premier algorithme par niveau à réaliser cette tâche [AMS$^+$96]. Cette méthode identifie les i-motifs à chaque i^{eme} itération puis génère les $(i{+}1)$-motifs fréquents à partir des i-motifs[48]. A chaque itération il requiert un passage sur la base de données pour compter le support des motifs candidats et ensuite élague les candidats non fréquents. Cet algorithme est très simple et efficace pour des données peu corrélées. *Apriori* a été suivi par de nombreuses variations dans le but d'en améliorer l'efficacité [BMUT97, Toi96].

La deuxième approche s'intéresse à la recherche de *motifs fermés fréquents* dans la base de données [PBTL99c]. Les motifs fermés fréquents permettent une représentation condensée et sans perte d'information des motifs fréquents, puisque l'ensemble des motifs fréquents (et leur support) peut être retrouvé à partir des motifs fermés fréquents. Cette idée fut implémentée dans *Close* [PBTL99c], qui est aussi un algorithme par niveau. Depuis *Close* d'autres algorithmes ont été proposés pour la recherche de motifs fermés fréquents [STB$^+$02, ZH02, WHP03].

Un autre sous-ensemble intéressant de motifs fréquents est l'ensemble des *générateurs fréquents*. Bastide *et al.* ont montré comment utiliser les générateurs fréquents pour trouver les règles d'association informatives[49] [BTP$^+$02, BTP$^+$00b]. Parmi les règles partageant les mêmes individus comme support et ayant la même confiance, les règles construites à partir d'un motif fermé et ayant un motif générateur en partie gauche sont celles qui contiennent le plus d'informations [Pas00b].

Le premier algorithme pour trouver les générateurs fréquents fut *Pascal* [BTP$^+$00a]. *Pascal* peut réduire le nombre de passages sur la base de données et compter le support des candidats plus efficacement. *Pascal* trouve tous les motifs fréquents et tous les générateurs fréquents, mais ce n'est pas suffisant pour trouver les règles informatives. Pour la génération des règles d'association informatives, il faut identifier parmi les motifs fréquents, les motifs fermés et les associer aux générateurs. Pour résoudre ce problème, un autre algorithme appelé *Zart* a été proposé récemment [SNK05]. *Zart* est un algorithme multifonctionnel d'extraction de motifs

[48] Un i-motif est un motif de taille i. Par exemple $\{A, C\}$ est un 2-motif.

[49] L'expression "Règles d'association informatives" regroupe la Base Générique et la Base Informative.

qui étend *Pascal* de manière à ce qu'il soit conforme à la génération de règles d'association informatives. *Zart* trouve les motifs fréquents, les motifs fermés fréquents et les générateurs fréquents. De plus, les générateurs fréquents sont associés à leur fermeture. En conséquence, la génération des règles informatives peut être réalisée très rapidement et aisément avec *Zart*.

Une quatrième approche est basée sur l'extraction des *motifs fréquents maximaux*. Un motif fréquent maximal a les propriétés suivantes : tous ses sur-motifs sont non fréquents et tous ses sous-motifs sont fréquents. Des expériences ont montré que cette approche est très efficace pour trouver de grands motifs dans les bases de données [Bay98, AAP00, LK98, GZ01]. Les algorithmes basés sur cette approche identifient, comme *Apriori*, l'ensemble des règles d'association.

Contributions et motivations. Nous présentons une nouvelle méthode pour trouver les motifs rares dans une base de données en deux étapes. La première étape identifie un ensemble générateur minimal appelé ensemble des *motifs rares minimaux*. Dans la seconde étape, ces motifs sont utilisés pour retrouver tous les motifs rares.

La découverte des motifs rares peut se révéler très intéressante, en particulier en médecine et en biologie. Prenons d'abord un exemple simulé d'une base de données médicale où nous nous intéressons à l'identification de la cause des maladies cardio-vasculaires (MCV). Une règle d'association fréquente telle que "{niveau élevé de cholestérol} $\Rightarrow$ {MCV}" peut valider l'hypothèse que les individus qui ont un fort taux de cholestérol ont un risque élevé de MCV. A l'opposé, si notre base de données contient un grand nombre de végétariens, une règle d'association rare "{végétarien} $\Rightarrow$ {MCV}" peut valider l'hypothèse que les végétariens ont un risque faible de contracter une MCV. Dans ce cas, les motifs {végétarien} et {MCV} sont tous deux fréquents, mais le motif {végétarien, MCV} est rare. Un autre exemple est en rapport avec la pharmacovigilance, qui est une partie de la pharmacologie dédiée à la détection et l'étude des effets indésirables des médicaments. L'utilisation de l'extraction des motifs rares dans une base de données des effets indésirables des médicaments pourrait contribuer à un suivi plus efficace des effets indésirables graves et ensuite à prévenir les accidents fatals qui aboutissent au retrait de certains médicaments (par exemple en août 2001, la cérivastatine, médicament hypolipémiant). Finalement, un troisième exemple basé sur les données réelles de la cohorte STANISLAS [SVH+98, MNSVS05a] montre l'intérêt de l'extraction des motifs rares pour la fouille de données dans des cohortes supposées saines. Cette cohorte est composée d'un millier de familles françaises présumées saines. Son principal objectif est de mettre en évidence l'influence des facteurs génétiques et environnementaux sur la variabilité des risques cardio-vasculaires. Une information intéressante à extraire de cette base de données pour l'expert dans ce domaine consiste en des profils qui associent des données génétiques à des valeurs extrêmes ou limites de paramètres biologiques. Cependant, ces types d'associations sont plutôt rares dans les cohortes saines. Dans ce contexte, l'extraction de motifs rares est très utile pour atteindre les objectifs de l'expert.

Concepts de base

Ci-dessous nous utilisons les définitions usuelles de la fouille de données. Nous considérons un ensemble d'*objets* $O = \{o_1, o_2, \ldots, o_m\}$, un ensemble d'*attributs* $A = \{a_1, a_2, \ldots, a_n\}$ et une relation $R \subseteq O \times A$, où $R(o, a)$ signifie que l'objet o possède l'attribut a. En analyse de concepts formels [GW99], le triplet (O, A, R) est appelé contexte formel. Un ensemble d'attributs est appelé *motif*. Un motif de taille i est appelé i-motif. Nous disons qu'un objet $o \in O$ contient le motif $P \subseteq A$, si $(o, p) \in R$ pour tout $p \in P$. Le *support* d'un motif P indique combien d'objets contiennent le motif. Un motif est dit *fréquent* si son support est supérieur ou égal à un *support minimum* donné (noté *min_supp* par la suite). Un motif est dit *rare* ou *non fréquent* si son support est inférieur ou égal à un *support maximum* (noté max_supp par la suite). P_2 est un sur-motif de P_1 ssi $P_1 \subseteq P_2$. Ici, nous nous sommes placés dans le cas particulier où $max\_supp = min\_supp - 1$, c'est-à-dire qu'un motif est rare s'il n'est pas fréquent.[50] Cela implique l'existence d'une seule frontière entre motifs rares et fréquents. [BBR03] fait par ailleurs lui aussi mention de cette frontière. Un motif G est dit *générateur* s'il n'existe pas de sous-motif H ($H \subset G$) de même support. Un motif X est dit *fermé* s'il n'existe pas de sur-motif Y ($X \subset Y$) de même support. L'extraction de motifs fréquents consiste à générer tous les motifs (fermés) fréquents (avec leur support) dont le support est supérieur ou égal à min_supp. L'extraction de motifs rares consiste à générer tous les motifs (avec leur support) dont le support est inférieur ou égal à max_supp.

Une approche basée sur les treillis pour l'énumération des motifs rares

Avant d'exposer nos algorithmes pour trouver les motifs rares, nous présenterons notre méthode du point de vue des treillis (voir [GW99] pour une description détaillée des treillis).

La Figure 8.3 montre le treillis de l'ensemble des parties $P(D)$ de l'ensemble des attributs dans notre base de données exemple D[51] (voir Tableau 8.3). L'ensemble des motifs rares forme un sup-demi-treillis car il est fermé pour l'opération de jointure, c'est-à-dire que pour tous motifs rares X et Y, $X \cup Y$ est aussi rare. D'un autre côté, il ne forme pas un inf-demi-treillis "meet", car la rareté de X et Y n'implique pas celle de $X \cap Y$. Notons que les motifs fréquents forment un inf-demi-treillis, c'est-à-dire que pour tous motifs fréquents X et Y, $X \cap Y$ est aussi fréquent.

	A	B	C	D	E
1	x	x		x	x
2	x		x		
3	x	x	x		x
4		x	x		x
5	x	x	x		x

Table 8.3: Une base de données simple (D) utilisée dans les exemples.

Prenons l'exemple de la base de données D (Tableau 8.3) et fixons $min\_supp = 3$, ce qui signifie que $max\_supp = 2$. Les motifs peuvent être séparés en deux ensembles formant une partition : les motifs rares et les motifs fréquents. Une frontière peut être dessinée entre ces deux ensembles. En bas du treillis nous trouvons le plus petit motif, l'ensemble vide. A chaque niveau se situent les motifs de même taille. Au sommet du treillis on trouve le motif le plus long qui contient tous les attributs. Le support de chaque motif est indiqué dans le coin en haut à droite (voir Figure 8.3).

Avant d'énoncer les définitions essentielles, nous empruntons à *Apriori* [AMS+96] ses deux principes fondamentaux que nous rappelons ici :

Propriété 8.1 *Tous les sous-ensembles d'un motif fréquent sont fréquents.*

Propriété 8.2 *Tous les sur-motifs d'un motif non fréquent sont non fréquents.*

L'ensemble des motifs rares et l'ensemble des motifs fréquents ont tous deux un sous-ensemble minimal générateur. Dans le cas des motifs fréquents, ce sous-ensemble est appelé ensemble des *motifs fréquents maximaux* (MFM).

[50]Les seuils max_supp et min_supp sont donnés en valeur absolue.

[51]Cet exemple est équivalent avec l'exemple présenté dans le Tableau 3.1.

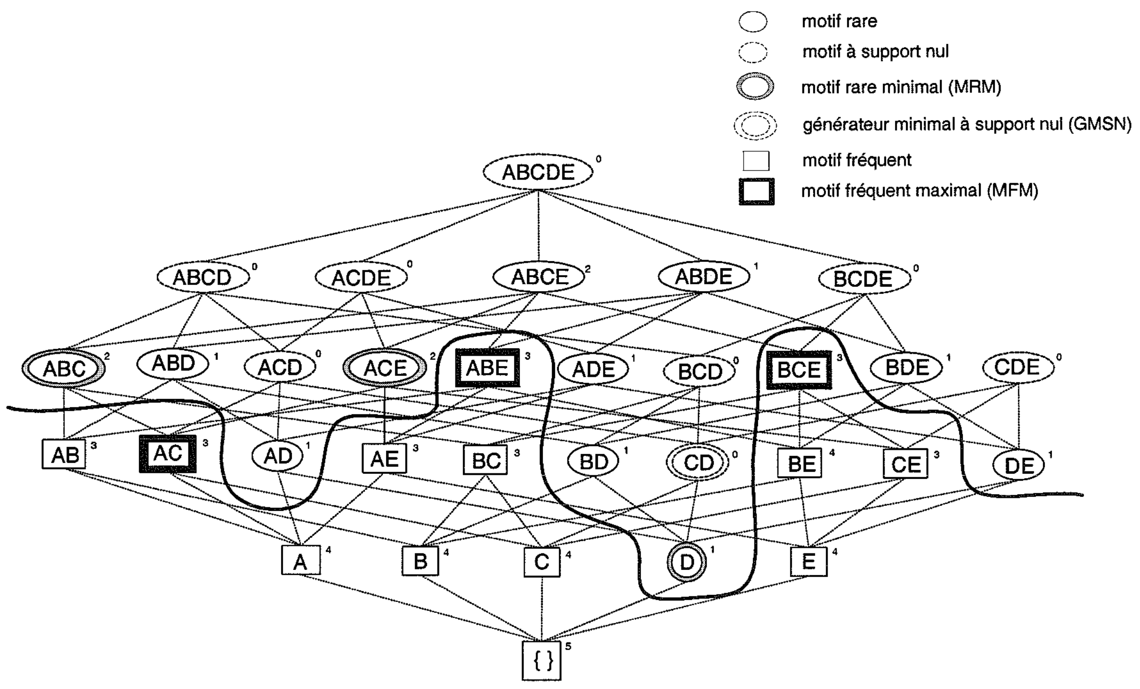


Figure 8.3: Treillis des parties de la base de données D (voir Tableau 8.3).

Définition 8.1 *Un motif est un* MFM *s'il est fréquent (et ainsi tous ses sous-motifs sont fréquents) et si tous ses sur-motifs ne sont pas fréquents.*

Ces motifs sont dits *maximaux*, parce qu'ils n'ont pas de sur-motifs fréquents. Du point de vue du nombre de ces motifs ils sont *minimaux*, c'est-à-dire qu'ils forment un ensemble générateur minimal à partir duquel tous les motifs fréquents peuvent être retrouvés.[52]

Nous pouvons définir les *motifs rares minimaux* (MRM) en tant que complémentaires des MFMs, de la manière suivante :

Définition 8.2 *Un motif est un* MRM *s'il est rare (et ainsi tous ses sur-motifs sont rares) et si tous ses sous-motifs ne sont pas rares.*

Ces motifs forment un ensemble générateur minimal à partir duquel tous les motifs rares peuvent être retrouvés. Tous les motifs fréquents peuvent être retrouvés à partir des MFM. Dans un premier temps, nous devons prendre tous les sous-ensembles possibles des MFM. Dans un deuxième temps, le support des motifs fréquents peut être calculé grâce à un passage sur la base de données. Un processus similaire est mis en œuvre pour retrouver les motifs rares. Nous devons d'abord générer tous les sur-motifs possibles des motifs rares minimaux, puis calculer le support des motifs rares grâce à un passage sur la base de données.

Parmi les motifs rares, nous distinguons deux sous-ensembles : **(a)** les motifs rares de support 0, et **(b)** les motifs rares de support supérieur à 0. Cette distinction est importante, car le nombre total de motifs rares peut être élevé et ainsi nous pouvons privilégier les motifs dont le support est non nul.

[52]Peut-être devrait-on les appeler plutôt motifs fréquents *les plus longs*, car ils n'ont pas de plus long sur-motif fréquent.

Définition 8.3 *Un motif est appelé* motif à support nul *si son support est égal à 0. Sinon, il est appelé* motif à support non nul.

Pour tous les motifs rares nous avons déjà décrit l'ensemble des motifs rares minimaux. Pour les motifs à support nul, un sous-ensemble générateur minimal semblable peut être défini :

Définition 8.4 *Un motif est un* générateur minimal à support nul *(GMSN) si c'est un motif à support nul (ainsi tous ses sur-ensembles sont des motifs à support nul) et si tous ses sous-motifs sont des motifs à support non nul.*

Sur la Figure 8.3 se trouvent un GMSN : {CD}. De plus, les GMSN forment une représentation condensée et sans perte d'information des motifs à support nul, c'est-à-dire qu'à partir des GMSN tous les motifs à support nul peuvent être retrouvés avec leur support (qui est toujours 0). Pour cela, nous avons seulement besoin de générer tous les sur-motifs possibles des GMSN en utilisant les attributs de la base de données.

Trouver les motifs rares

Nous présentons les deux étapes de notre méthode pour trouver les motifs rares. La première étape trouve seulement les motifs rares minimaux, tandis que la seconde retrouve les motifs rares non nuls à partir de l'ensemble des motifs rares minimaux.

Nous ne générons pas les motifs à support nul à cause de leur grand nombre. Pour éviter les motifs à support nul, nous utiliserons les GMSN. La seconde étape de notre méthode permet de restaurer tous les motifs rares non nuls à partir des MRM à l'aide d'une approche par niveau. Si un candidat a un sous-motif GMSN, alors ce candidat est nécessairement un motif à support nul et peut être ainsi élagué. Autrement dit, à l'aide des GMSN nous pouvons réduire l'espace de recherche lors de la recherche des motifs rares.

Trouver les motifs rares minimaux. Bien que cela puisse paraitre surprenant, les motifs rares minimaux peuvent être trouvés simplement à l'aide de l'algorithme bien connu *Apriori*. *Apriori* est basé sur deux principes (voir Propriétés 8.1 et 8.2). Il est conçu pour trouver les motifs fréquents, mais, puisque nous sommes dans le cas où non fréquent signifie rare, cela a pour "effet collatéral" d'explorer également les motifs rares minimaux. Quand *Apriori* trouve un motif rare, il ne générera plus tard aucun de ses sur-motifs car ils sont de manière sûre rares. Puisque *Apriori* explore le treillis des motifs niveau par niveau du bas vers le haut, il comptera le support des motifs rares minimaux. Ces motifs seront élagués et plus tard l'algorithme peut remarquer qu'un candidat a un sous-motif rare. En fait *Apriori* vérifie si tous les $(k-1)$-sous-motifs d'un k-candidat sont fréquents. Si l'un d'entre eux n'est pas fréquent, alors le candidat est rare. Autrement dit, cela signifie que le candidat a un sous-motif rare minimal. Grâce à cette technique d'élagage, *Apriori* peut réduire significativement l'espace de recherche dans le treillis des motifs.

Une légère modification d'*Apriori* suffit pour conserver les MRM. Si le support d'un candidat est inférieur au support minimum, alors au lieu de l'effacer nous l'enregistrons dans l'ensemble des motifs rares minimaux (voir Algorithme 18).

Fonction `Apriori-Gen` : à l'aide des k-motifs fréquents, génère les candidats potentiellement fréquents de taille $(k+1)$. Potentiellement fréquent signifie ne pas avoir de sous-motif rare, c'est-à-dire pas de sous-motif rare minimal. Inclure un motif rare implique être rare (voir Propriété 8.2). Pour une description détaillée de cette fonction voir Algorithme 23.

Algorithme 18 (Apriori-Rare) :

Description :	modification Apriori pour trouver les motifs rares minimaux
Entrée :	base de données + min_supp
Sortie :	tous les motifs fréquents + motifs rares minimaux

```
 1)  $C_1 \leftarrow$ {1-motifs};
 2)  $i \leftarrow 1$;
 3)  while ($C_i \neq \emptyset$)
 4)  {
 5)      SupportCount($C_i$); // compte le support des motifs candidats
 6)      $R_i \leftarrow \{r \in C_i \mid \text{support}(r) < min\_supp\}$; // R – pour les motifs rares
 7)      $F_i \leftarrow \{f \in C_i \mid \text{support}(f) \geq min\_supp\}$; // F – pour les motifs fréquents
 8)      $C_{i+1} \leftarrow$ Apriori-Gen($F_i$); // C – pour les candidats
 9)      $i \leftarrow i + 1$;
10)  }
11)  $I_{MR} \leftarrow \bigcup R_i$; // motifs rares minimaux
12)  $I_F \leftarrow \bigcup F_i$; // motifs fréquents
```

L'exécution de l'algorithme sur la base de données D (Tableau 8.3) avec un support minimum de 3 (équivalent à un support maximum de 2) est illustrée dans le Tableau 8.4. En prenant l'union des tableaux R_i, l'algorithme trouve les motifs rares minimaux ({D} avec support 1, {ABC} et {ACE} avec support 2).

Dans le prochain paragraphe, nous montrons comment restaurer les sur-motifs des MRM (c'est-à-dire comment reconstruire tous les motifs rares) en évitant les motifs à support nul.

Retrouver les motifs rares. Tous les motifs rares sont retrouvés à partir des motifs rares minimaux. Pour cela nous avons besoin de générer tous les sur-motifs possibles des MRM. Les générateurs minimaux à support nul sont utilisés pour filtrer les motifs à support nul pendant la génération des sur-motifs. De cette manière l'espace de recherche peut être réduit de manière considérable. Dans cette section nous présentons un algorithme prototype pour cette tâche appelé *Arima*[53] (A Rare Itemset Miner Algorithm, voir Algorithme 19).

L'exécution de l'algorithme sur la base de données D (Tableau 8.3) avec un support minimum de 3 (équivalent à un support maximum de 2) est illustrée dans le Tableau 8.5.

L'algorithme détermine d'abord le plus court MRM, {D}, qui est rare et ainsi copié dans R_1. Ses sur-motifs de taille 2 sont générés et stockés dans C_2 ({AD}, {BD}, {CD}, et {DE}). Avec un passage sur la base de données leur support peut être compté. Puisque {CD} est un motif à support nul, il est copié dans la liste des GMSN. Les motifs non nuls sont stockés dans R_2. Pour chaque motif rare dans R_2 tous ses sur-motifs sont générés. Par exemple, à partir de {AD} nous pouvons générer les candidats suivants : {ABD}, {ACD} et {ADE}. Si un candidat possède un sous-motif GMSN, alors le candidat est de manière sûre un motif à support nul et peut être élagué ({ACD}). Les candidats potentiels non nuls sont stockés dans C_3. Dans les C_i les doublons ne sont pas permis. A partir des MRM, les 3-motifs sont ajoutés à C_3 ({ABC} et {ACE}). L'algorithme s'arrête quand R_i est vide. L'union des tableaux R_i donne tous les motifs rares à support non nul. A la fin nous avons aussi collecté tous les GMSN. Ainsi si on a

[53] A ne pas confondre avec la méthodologie des modèles ARIMA (Auto Regressive Integrated Moving Average).

C_1	supp
{A}	4
{B}	4
{C}	4
{D}	1
{E}	4

R_1	supp
{D}	1

F_1	supp
{A}	4
{B}	4
{C}	4
{E}	4

C_2	supp
{AB}	3
{AC}	3
{AE}	3
{BC}	3
{BE}	4
{CE}	3

R_2	supp
$\emptyset$	

F_2	supp
{AB}	3
{AC}	3
{AE}	3
{BC}	3
{BE}	4
{CE}	3

C_3	supp
{ABC}	2
{ABE}	3
{ACE}	2
{BCE}	3

R_3	supp
{ABC}	2
{ACE}	2

F_3	supp
{ABE}	3
{BCE}	3

C_4	supp
$\emptyset$	

Table 8.4: Exécution de l'algorithme *Apriori-Rare* avec $min\_supp = 3$ ($max\_supp = 2$).

besoin des motifs à support nul, cette liste peut être utilisée pour les retrouver. Le procédé est similaire : nous aurions besoin de générer tous les sur-motifs possibles des GMSN. Dans notre cas nous ne nous sommes intéressés qu'aux motifs non nuls, mais il est possible de travailler avec les motifs à support nul.

Conclusion et travaux futurs

Dans cette sous-section, nous avons présenté une méthode pour extraire les motifs rares d'une base de données. Notre méthode est composée de deux étapes : **(1)** nous trouvons un sous-ensemble générateur minimal des motifs rares appelés MRM (algorithme *Apriori-Rare*) ; **(2)** à l'aide des MRM nous retrouvons les motifs rares dont le support est strictement supérieur à 0 (algorithme *Arima*).

Notre méthode fait partie des premières à s'intéresser spécifiquement aux motifs rares. *Apriori* fut le premier algorithme pour trouver les motifs fréquents et a été suivi par de nombreux algorithmes plus efficaces. De manière similaire, il ne fait aucun doute que nos algorithmes prototypes pourraient être améliorés de nombreuses manières. Dans le futur nous aimerions travailler sur ce sujet.

Un domaine important dans le cadre de l'utilisation des motifs rares est la génération des règles d'association rares (voir Section 5.3 pour plus de détails).

En outre, dans un futur proche, nous prévoyons de décrire des expériences réalisées sur des données réelles de la cohorte STANISLAS, dans le but de fournir un exemple concret de ce nouvel aspect prometteur de la découverte de connaissances dans les bases de données.

Algorithme 19 (Arima):

Description :	retrouve les motifs rares à partir des MRM
Entrée :	base de données + MRM
Sortie :	tous les motifs rares à support non nul + GMSN

```
 1) $GMSN \leftarrow \emptyset$;
 2) $S \leftarrow$ {tous les attributs de $D$};
 3) $i \leftarrow$ {longueur du plus petit $MRM$};
 4) $C_i \leftarrow \{i\text{-}MRM\}$; // c'est à dire les plus courts motifs dans les $MRM$
 5) $GMSN \leftarrow GMSN \cup \{z \in C_i \mid \text{support}(z) = 0\}$;
 6) $R_i \leftarrow \{r \in C_i \mid \text{support}(r) > 0\}$;
 7) while ($R_i \neq \emptyset$)
 8) {
 9)     boucle sur les éléments de $R_i$ ($r$) {
10)         $Cand \leftarrow$ {tous sur-motifs de $r$ utilisant $S$}; // aucun doublon permis
11)         boucle sur les éléments de $Cand$ ($c$) {
12)             si $c$ a un sous-motif dans $GMSN$ (c'est à dire si $c$ est un sur-motif
                d'un $GMSN$), alors supprime $c$ de $Cand$;
13)         }
14)         $C_{i+1} \leftarrow C_{i+1} \cup Cand$; // aucun doublon permis
15)         $Cand \leftarrow \emptyset$; // ré-initialise $Cand$
16)     }
17)     SupportCount($C_{i+1}$); // compte le support des motifs candidats
18)     $C_{i+1} \leftarrow C_{i+1} \cup \{(i+1)\text{-motifs de } MRM\}$;
19)     $GMSN \leftarrow GMSN \cup \{z \in C_{i+1} \mid \text{support}(z) = 0\}$;
20)     $R_{i+1} \leftarrow \{r \in C_{i+1} \mid \text{support}(r) > 0\}$;
21)     $i \leftarrow i + 1$;
22) }
23) $I_R \leftarrow \bigcup R_i$; // motifs rares à support non nul
```

$GMSN = \emptyset$
$S = \{A, B, C, D, E\}$
$MRM = \{D(1), ABC(2), ACE(2)\}$
$i = 1$

C_1	supp
{D}	1

$GMSN_{avant} = \emptyset$
$GMSN_{apres} = \emptyset$

R_1	supp
{D}	1

C_2	supp
{AD}	1
{BD}	1
{CD}	0
{DE}	1

$GMSN_{avant} = \emptyset$
$GMSN_{apres} = \{CD\}$

R_2	supp
{AD}	1
{BD}	1
{DE}	1

C_3	supp
{ABD}	1
{ADE}	1
{BDE}	1
{ABC}	2
{ACE}	2

$GMSN_{avant} = \{CD\}$
$GMSN_{apres} = \{CD\}$

R_3	supp
{ABD}	1
{ADE}	1
{BDE}	1
{ABC}	2
{ACE}	2

C_4	supp
{ABDE}	1
{ABCE}	2

$GMSN_{avant} = \{CD\}$
$GMSN_{apres} = \{CD\}$

R_4	supp
{ABDE}	1
{ABCE}	2

C_5	supp
$\emptyset$	

$GMSN_{avant} = \{CD\}$
$GMSN_{apres} = \{CD\}$

R_5	supp
$\emptyset$	

Table 8.5: Exécution de l'algorithme *Arima* avec $min\_supp = 3$ ($max\_supp = 2$).

8.1.4 Conclusion et perspectives

Dans ce chapitre nous résumons les contributions à la recherche apportées par cette thèse. Nous présentons une synthèse de notre travail et développons différentes pistes pour nos travaux futurs.

Conclusion

Le sujet principal de cette thèse est *l'extraction de connaissances dans les bases de données* (ECBD). Plus précisément, nous avons étudié deux des plus importantes tâches d'ECBD actuelles, qui sont l'extraction de motifs et la génération de règles d'association. Tout au long de notre travail, notre objectif a été de trouver des règles d'associations *intéressantes* selon plusieurs points de vue : dans un but de fouille efficace, pour réduire au minimum l'ensemble des règles extraites et pour trouver des unités de connaissances intelligibles (et facilement interprétables). Pour atteindre ce but, nous avons développé et adapté des algorithmes spécifiques.

Les contributions principales de cette thèse sont : **(1)** nous avons développé et adapté des algorithmes pour trouver les règles d'association minimales non-redondantes ; **(2)** nous avons défini une nouvelle base pour les règles d'associations appelée "règles fermées" ; **(3)** nous avons étudié un champ de l'ECBD important mais relativement peu étudié, à savoir l'extraction des motifs rares et des règles d'association rares ; **(4)** nous avons regroupé nos algorithmes et une collection d'autres algorithmes ainsi que d'autres opérations auxiliaires d'ECBD dans une boîte à outils logicielle appelée CORON.

Algorithmes pour l'extraction des règles MNR

Dans le Chapitre 3 nous présentons deux algorithmes spécifiquement adaptés pour extraire des règles d'association minimales non-redondantes ($\mathcal{MNR}$). Cet ensemble de règles s'obtient sans perte d'information, et représente de manière informative toutes les règles d'association valides. Notre premier algorithme, *Zart*, est une extension de *Pascal* qui est probablement l'algorithme le plus efficace de recherche des motifs fréquents par niveau. Dans *Zart*, *Pascal* est intégré avec *Apriori-Close* et nos extensions. En plus des capacités de *Pascal*, *Zart* est capable d'identifier l'ensemble des motifs fermés fréquents et de leurs associer leurs générateurs. Nous montrons que ces données supplémentaires fournies par *Zart* sont essentielles pour l'extraction de règles $\mathcal{MNR}$. Dans notre second algorithme, *Eclat-Z*, nous allons plus loin et montrons comment généraliser l'idée de *Zart* pour *n'importe quel* algorithme d'extraction des motifs fréquents. Il est ainsi possible d'étendre n'importe quel algorithme d'extraction des motifs fréquents afin de rajouter le support de l'extraction des règles $\mathcal{MNR}$. Nous présentons une idée générale dont *Eclat-Z* est une implantation concrète. Comme son nom l'indique *Eclat-Z* est une extension d'*Eclat*. *Eclat* est un algorithme vertical identifiant les motifs fréquents très efficacement. Néanmoins, à cause de sa recherche en profondeur, il produit les motifs fréquents de manière non ordonnée (par longueur). Or *Zart* dépend de l'hypothèse que les motifs fréquents sont disponibles en étant ordonnés par longueur. Dans *Eclat-Z* nous résolvons ce problème à l'aide d'une technique spéciale d'indexation de fichiers. Grâce à cette idée, tout algorithme arbitraire de recherche des motifs fréquents peut être transformé en un algorithme trouvant un sous-ensemble utile et intéressant de toutes les règles d'associations valides : les règles d'association minimales non-redondantes.

Règles d'association fermées

Dans le Chapitre 4 nous introduisons une nouvelle base appelée les "règles fermées" que nous positionnons entre l'ensemble de toutes les règles valides et l'ensemble des règles minimales non-redondantes. Dans le cas de bases de données denses et hautement corrélés, la génération des règles fermées est plus efficace que celle des règles d'association. Néanmoins, pour les bases de données creuses, lorsque presque tous les motifs fréquents sont fermés, toutes les règles d'association peuvent être extraites plus efficacement. Quelle que soit la méthode, l'ensemble des règles fermées est toujours plus petit que l'ensemble des règles d'association valides. Les règles fermées étant une représentation concise de toutes les règles valides et ne requérant que les motifs fréquents fermés, elles semblent être une bonne alternative à l'ensemble de toutes les règles d'association.

Motifs rares et règles d'association rares

Le Chapitre 5 est un des chapitres les plus originaux de cette thèse. Dans ce chapitre, nous nous intéressons aux problèmes de l'extraction des motifs rares et de la génération de règles d'association rares. Dans la littérature, ces problèmes n'ont jusqu'alors pas été étudiés en détail, bien que les motifs rares puissent contenir des informations importantes dans la même mesure que les motifs fréquents. En particulier, l'application aux diagnostics médicaux nous parait un champ d'application particulièrement adapté.

Dans le Chapitre 5.1 nous présentons une méthode permettant de trouver tous les *motifs rares.* Pour ce faire, nous utilisons l'algorithme classique *Apriori. Apriori* est connu pour trouver l'ensemble des motifs fréquents, mais il trouve également un ensemble spécial des motifs rares : les motifs rares minimaux (MRM). Une légère modification d'*Apriori*, que nous appelons *Apriori-Rare*, permet ainsi de conserver l'ensemble des MRM. Nous montrons ensuite comment reconstruire l'ensemble de tous les motifs rares à partir des MRM tout en évitant les motifs de support 0.

Dans le Chapitre 5.3 nous allons plus loin en montrant comment générer des *règles d'association rares* valides. Notre travail est motivé par la question ouverte de longue date visant à construire un algorithme efficace pour la découverte de règles à support faible et confiance élevée. Afin de trouver de telles règles en utilisant des algorithmes conventionnels de recherche des motifs fréquents comme *Apriori*, le support minimal doit être fixé à un seuil très faible, ce qui augmente de manière drastique le temps d'exécution de l'algorithme. De plus, lorsque le support minimal est fixé très bas, *Apriori* produit un grand nombre des motifs fréquents. Ce problème est également connue sous le nom de *problème des motifs rares.* Pour résoudre ce problème bien connu, nous proposons la solution suivante : de manière itérative, nous diminuons le support minimal pour *Apriori* jusqu'à atteindre la limite pour laquelle l'algorithme reste utilisable. A ce niveau nous changeons d'algorithme pour *Apriori-Rare* afin de trouver les MRM. Nous prouvons que les MRM sont des générateurs rares minimaux (MRG). En trouvant leurs fermetures, des classes d'équivalence rares sont obtenues, desquelles il est possible d'extraire des règles d'association rares exactes (nous appelons de telles règles "règles MRG exactes"). Nous montrons également comment extraire des règles MRG approximatives ; néanmoins leur intérêt parait limité. De ce fait, nous nous concentrons plutôt sur les règles rares exactes. De plus, de telles règles sont non-redondantes car l'antécédent est minimal et le conséquent maximal, impliquant ainsi que parmi les règles de support et confiance identiques, celles-ci contiennent le plus d'information. Des règles MRG peuvent être trouvées dans tous les cas, y compris lorsque le support minimal est élevé.

Nous pensons que l'extraction de règles d'association rares est un champ intéressant et prometteur de l'ECBD. L'utilisation de règles d'association rares peut s'avérer avantageuse pour toute une variété d'applications pratiques : marketing, business, télécommunications, ou bien applications scientifiques sur des domaines tels que la biologie, l'astronomie, la médecine, etc.

Étant un champ relativement nouveau, cette extraction n'a pas encore été étudiée en détails. Notre approche est ainsi un premier pas dans cette direction. Les motifs rares et les règles d'association rares soulèvent de nombreuses questions. Il existe différentes bases pour l'extraction de règles d'associations fréquentes. Pouvons-nous définir des bases similaires pour l'extraction de règles d'association rares ? Les règles exactes MRG forment un sous-ensemble de toutes les règles d'association rares. Serait-il possible de dériver toutes ces règles rares exactes à partir de l'ensemble des règles MRG exactes ? Dans notre approche nous considérons seulement une frontière, i.e. pour nous un motif est rare s'il n'est pas fréquent. Une piste de recherche intéressante serait de travailler avec deux frontières. Enfin, tandis que nous utilisons un "effet de bord" d'algorithmes par niveau de recherche des motifs fréquents, à savoir les motifs rares minimaux, on peut se demander s'il ne serait pas possible de trouver ces motifs rares directement, i.e. *sans* avoir à extraire au préalable les motifs fréquents. Les réponses à ces questions nécessitent encore d'avantage de recherches.

La boîte à outils Coron

Les algorithmes présentés dans cette thèse ont été implémentés et regroupés dans une plate-forme logicielle unifiée appelée CORON. CORON est une boîte à outils de fouille de données indépendante du domaine. Non seulement CORON incorpore une riche collection d'algorithmes de fouille de données mais CORON permet également un grand nombre d'opérations auxiliaires. À notre connaissance, aucun autre logiciel n'a été conçu spécifiquement pour l'extraction de motifs et la génération de règles d'association. CORON fournit également un support pour la préparation, le filtrage des données ainsi que pour l'interprétation des unités de connaissances extraites.

La plupart des expériences avec CORON ont été réalisées sur une véritable base de données biomédicales appelée la cohorte STANISLAS. Durant ces expériences, nous avons réalisé qu'il nous était nécessaire d'avoir **(1)** une méthodologie pour la fouille et **(2)** un outil permettant de l'implanter. Le Chapitre 6 présente notre méthodologie globale de fouille de données, pouvant être généralisée à n'importe quel ensemble des données. Cette méthodologie peut être utilisée pour les recherches des motifs aussi bien fréquents que rares.

Améliorations futures de Coron. Alors que CORON est déjà une plate-forme complète qui a montré son utilité dans des projets variés, elle pourrait être amélioré de différentes manières. Nous listons ici différentes idées d'améliorations.

Pour le moment, en utilisant CORON, il est possible d'extraire deux représentations condensées de règles d'association fréquentes, à savoir les règles fermées et les règles d'association minimales non-redondantes. Nous sommes également intéressés à travailler avec d'autres bases comme la base de Duquennes-Guigues [GD86] pour les règles exactes, la base de Luxenburger [Lux91] pour les règles approchées ou encore les règles représentatives de Kryszkiewicz [Kry98]. Toutes ces bases sont plus condensées que les règles $\mathcal{MNR}$. Néanmoins, elles ne sont pas simultanément sans perte d'information, correctes et informatives.

A l'heure actuelle, CORON peut seulement travailler avec des contextes binaires. Les contextes pluri-valués devraient également être pris en charge dans une version future.

CORON contient, entre autres algorithmes, *Eclat* et *Charm*. Récemment, Zaki a proposé une optimisation appelée "diffsets" réduisant de manière significative l'utilisation mémoire de ces algorithmes (voir Section 3.2.2). Nous planifions ainsi d'ajouter ces versions optimisées, *dEclat* et *dCharm*, à CORON. Utilisant moins de mémoire, il est probable qu'elles permettent le traitement de bases de données plus grandes ou de valeurs de support minimum plus petits.

CORON ne comprend aucun algorithme de la catégorie "autre" (voir Section 3.2.4). L'algorithme de ce type le plus connu est sans doute *FP-growth* [HPY00]. Il utilise une nouvelle structure de donnée appelée FP-arbre. Cette nouvelle structure est une représentation compressée de toutes les transactions de la base de données. Alors que *FP-growth* trouve des motifs fréquents, certaines de ses modifications comme *Closet* [PHM00] ou *Closet*$^+$ [WHP03] permettent l'extraction des motifs fermés fréquents.

Charm-MFI est une extension simple de *Charm* filtrant les motifs fréquents maximaux parmi les motifs fréquents fermés. Nous avons en projet de la comparer avec d'autres algorithmes créés spécifiquement pour les motifs fréquents maximaux, e.g. *Max-Miner* [Bay98], *DepthProject* [AAP00], *GenMax* [GZ01], etc.

Un des buts du projet CORON est d'intégrer un grand nombre d'algorithmes de recherche des motifs. Comme il n'y a, pour des bases de données arbitraires, aucun algorithme *meilleur*, CORON offre la possibilité à l'utilisateur de tester différents algorithmes et de choisir ainsi celui correspondant le mieux à ses besoins. Néanmoins, une grande liste d'algorithmes pourrait être difficile à manipuler par les utilisateurs. En conséquence, nous devons faire une étude expérimentale portant sur un grand nombre de bases de données afin d'être en mesure de fournir à l'utilisateur des indications sur le meilleur algorithme pressenti pour un ensemble des données spécifique. Alternativement, CORON pourrait également analyser l'ensemble des données et fournir un algorithme qui serait en mesure de donner des performances optimales.

L'ECBD se réfère au processus de découverte de motifs nouveaux et utiles dans des données. Le développement d'algorithmes efficaces n'est qu'une étape de ce processus. Pour l'étape de visualisation, nous utilisons la notion de treillis. Pour construire des treillis, nous commençons par chercher les motifs fermés fréquents, puis calculons l'ordre des concepts. Néanmoins, il y a des méthodes plus efficaces de calcul de l'ordre ; par exemple, l'utilisation d'algorithmes explorant l'ordre parallèlement aux motifs fermés fréquents. Actuellement, CORON ne dispose que d'algorithmes extrayant motifs et règles d'association, mais dans le futur nous espérons y intégrer des algorithmes du domaine de l'analyse des concepts formels [KO01, KO02] afin de rendre son module de visualisation plus efficace. Comme nous avons une collaboration avec l'équipe des développeurs du projet GALICIA, il est fort probable que les deux plate-formes s'échangent mutuellement des algorithmes ou fusionnent dans le futur.

Enfin, un dernier objectif est de connecter CORON avec des systèmes de gestion de bases de données. Par exemple, Coron-base et ASSRULEX devraient être capable de sauvegarder leurs résultats directement dans un SGBD afin de faciliter l'échange de données avec d'autres projets qui souhaiteraient utiliser des données fournies par CORON.

Perspectives à moyen et long terme

Le composant final de tout algorithme de fouille de données est la stratégie de gestion des données : la manière dont les données sont stockées, indexées et accédées. La plupart des algorithmes d'analyse de données connus ont été développés avec l'hypothèse que les bases de données peuvent être consultées rapidement et efficacement en mémoire principale (RAM). L'évolution rapide des technologies de mémoire a également été suivie par l'amélioration des technologies de stockage secondaire. De nombreuses bases de données massives dépassent actuellement les capacités de la RAM, rendant nécessaire le développement de nouveaux algorithmes et/ou de nouvelles gestions de données. Actuellement, des bases de données de l'ordre du giga- ou teraoctet ne sont pas rares. Le but final serait d'être capable de manipuler efficacement de telles bases de données.

Un autre défi est le traitement de bases de données en évolution constante, comme les facturations d'appels téléphonique ou de consommation d'électricité. Les bases de données peuvent s'enrichir de nouvelles entrées, nouveaux attributs ou bien des attributs d'entrées existantes peuvent changer de valeur. Le connaissances extraites avant les changements peuvent ainsi nécessiter une mise à jour ou se voir invalidées. Ceci requiert l'utilisation d'algorithmes incrémentaux. Alors que les bases de données continuent d'augmenter en taille, il est généralement admis que des bases très grandes ne peuvent être traitées efficacement qu'à l'aide d'algorithmes parallèles dans un environnement distribué ; le calcul parallèle étant idéal pour traiter les problèmes de passage à l'échelle. Les algorithmes de fouille de données devraient être mieux intégrés avec les SGBD afin de fournir représentation, stockage et accès communs.

La combinaison des caractéristiques précédemment mentionnées — intégration avec les SGBD pour le calcul parallèle incrémental — pourrait permettre des solutions de fouille de données très puissantes. Il serait intéressant de développer de telles solutions pour la recherche de règles d'association rares dans de grandes bases de données.

8.2 Rezümé

8.2.1 Konklúzió

A dolgozat központi témája *ismeretfeltárás adatbázisokban* (KDD).[54] Egészen pontosan minták keresésével, ill. asszociációs szabályok generálásával foglalkoztunk, melyek jelenleg a KDD legfontosabb kutatási területei közé tartoznak. Munkánk során végig azt a célt tartottuk szem előtt, hogy *érdekes* asszociációs szabályokat nyerjünk ki. Ezeknek a szabályoknak többféle kritériumnak is meg kell felelniük, úgymint **(a)** hatékonyan kell tudni őket megkeresni; **(b)** a kinyert szabályhalmaz mérete legyen minimális; valamint **(c)** ezen szabályok legyenek érthetőek, könnyen értelmezhetőek.

A tézis tudományos eredményei a következők: **(1)** Kifejlesztettünk ill. adaptáltunk olyan algoritmusokat, melyekkel minimális nem-redundáns asszociációs szabályok ($\mathcal{MNR}$) generálhatók; **(2)** Definiáltunk egy új bázist az asszociációs szabályok számára (zárt szabályok); **(3)** Foglalkoztunk a KDD egy igen fontos, de eddig még részleteiben fel nem tárt területével, nevezetesen a ritka minták ill. a ritka asszociációs szabályok kérdésével; **(4)** Egy egységes szoftvereszközben gyűjtöttük össze a dolgozatban bemutatott valamennyi algoritmust, további ismert algoritmusokkal, ill. egyéb kiegészítő műveletekkel együtt (CORON rendszer).

8.2.2 MNR szabályokat kereső algoritmusok

A 3. fejezetben két olyan algoritmust mutatunk be, melyeket kifejezetten a minimális nem-redundáns asszociációs szabályok keresésére terveztünk. Ezen szabályhalmaz az összes asszociációs szabálynak egy veszteségmentes, pontos és informatív reprezentációja. Az első algoritmus (*Zart*) a *Pascal* algoritmus gyakorlati kiterjesztése. A szintenkénti, gyakori mintákat kereső algoritmusok között talán a *Pascal* tekinthető a leghatékonyabbnak. A *Zart* algoritmusban a *Pascal* algoritmust ötvözzük az *Apriori-Close*-zal, ill. még egy saját kiterjesztést is hozzátettünk. A *Zart* algoritmus – a *Pascal*-tól eltérően – a gyakori zárt mintákat is megkeresi, s ezekhez hozzátársítja a megfelelő generátor mintákat. Megmutatjuk, hogy a *Zart* algoritmus eredménye szükséges az $\mathcal{MNR}$ szabályok előállításához. A második algoritmus (*Eclat-Z*) esetében megmutatjuk hogyan általanosítható a *Zart*-ban bemutatott ötlet bármely gyakori mintákat kereső algoritmusra. Ezáltal tetszőleges gyakori mintákat kereső algoritmus kiterjeszthető oly módon, hogy az adott algoritmus támogassa az $\mathcal{MNR}$ szabályok előállítását is. Az *Eclat-Z* algoritmus ennek az általános ötletnek egy konkrét megvalósítása. Mint ahogy azt a neve is mutatja, az *Eclat-Z* algoritmus alapja az *Eclat*. Az *Eclat* egy vertikális algoritmus, mely nagyon hatékonyan keresi meg az összes gyakori mintát. Mivel mélységi keresést végez, ezért a gyakori mintákat hossz szerint rendezetlen módon állítja elő. A *Zart* algoritmus azonban megköveteli, hogy a minták hossz szerint növekvő sorrendben legyenek. Az *Eclat-Z* esetében ezt a problémát egy speciális file indexelési technikával oldottuk meg. Ezen ötlet alapján tehát a tetszőleges, gyakori mintákat kereső algoritmus átalakítható oly módon, hogy az algoritmus az összes asszociációs szabálynak csupán egy érdekes részhalmazát, a minimális nem-redundáns asszociációs szabályokat keresse meg.

8.2.3 Zárt asszociációs szabályok

A 4. fejezetben egy új bázist, a zárt szabályokat mutatjuk be. Ez a bázis az összes szabály, ill. a minimális nem-redundáns szabályok közé helyezhető el. Sűrű, erősen korrelált adatbázisok

[54]Knowledge Discovery in Databases

esetén a zárt szabályok halmazát hatékonyabban elő lehet állítani mint az összes asszociációs szabályt. Ritka adatbázisok esetén azonban, mikor a gyakori minták nagyrésze zárt, az összes szabályok halmaza hatékonyabban kereshető meg. Elmondható azonban, hogy a zárt szabályok száma minden esetben kevesebb mint az összes asszociációs szabály. Mivel a zárt szabályok az összes asszociációs szabálynak egy veszteségmentes, tömörített reprezentációja, ill. ezen szabályok előállítása nem igényel semmi egyebet csupán a gyakori zárt mintákat, ezért a zárt szabályok egy jó alternatívát kínálnak az összes asszociációs szabállyal szemben.

8.2.4 Ritka minták és ritka asszociációs szabályok

Az 5. fejezet tartalmazza a dolgozat leglényegesebb eredményeit. Ebben a fejezetben a ritka minták és ritka asszociációs szabályok előállításával foglalkozunk. A szakirodalomban ezt a két területet még nem kutatták kellő alapossággal annak ellenére, hogy a ritka minták is fontos információkat rejthetnek, csakúgy mint a gyakori minták. Orvosi diagnózisok esetén például különösen fontos szerephez jutnak a ritka minták.

Az 5.1. fejezetben bemutatunk egy olyan módszert, mellyel az összes ritka minta megkereshető. Erre a célra a jól ismert *Apriori* algoritmust használjuk. Az *Apriori*-ról köztudott, hogy egy gyakori mintákat kereső algoritmus, de valójában a ritka mintáknak egy speciális részhalmazát, a minimális ritka mintákat is megtalálja. Az *Apriori* egy enyhén módosított változata, az *Apriori-Rare* algoritmus ahelyett, hogy törölné a minimális ritka mintákat, megtartja őket. Megmutatjuk, hogy hogyan lehet a minimális ritka minták halmazából visszaállítani az összes ritka mintát.

Az 5.3. fejezetben továbblépünk, s megmutatjuk hogyan lehet ritka asszociációs szabályokat előállítani. Munkánknak ezt a részét egy hosszú ideje megválaszolatlan kérdés motiválta, mégpedig az, hogy hogyan lehet hatékonyan olyan szabályokat előállítani, melyek support értéke alacsony, megbízhatóságuk viszont nagyon magas. Ha ilyen szabályokat szeretnénk, akkor a hagyományos, gyakori mintákat kereső algoritmusok esetén – mint amilyen az *Apriori* is – a minimum support értéket nagyon alacsonyra kell állítani. Ennek viszont az az eredménye, hogy az algoritmus futásideje drasztikusan megnő, ill. az algoritmus nagyon nagy számú gyakori mintát talál. Ez az ún. „ritka minták problémája". Mi erre a következő megoldást javasoljuk. Az *Apriori* algoritmust használva addig csökkentjük iteratív módon a minimum support értékét, amíg el nem érjük azt a határt, amelyen túl már működésképtelen az algoritmus. Ezen a ponton átváltunk az *Apriori-Rare* algoritmusra, mely kiszűri a minimális ritka mintákat. Bebizonyítjuk, hogy a minimális ritka minták egyben minimális ritka generátorok (MRG) is. Ezután megkeressük ezen minták lezártját, s így rendelkezésünkre áll néhány ritka ekvivalencia osztály. Ezen ritka ekvivalencia osztályokból pedig már elő lehet állítani pontos ritka asszociációs szabályokat (ezeket a szabályokat mi „pontos MRG szabályoknak" neveztük el). Azt is megmutatjuk, hogy hogyan lehet megközelítő, azaz nem pontos MRG szabályokat előállítani. Mivel ez utóbbi szabályok jelentősége kétséges, ezért inkább a pontos ritka szabályokra koncentrálunk. Az MRG szabályok ezen túlmenően nem-redundánsok, mivel a szabályok bal oldala minimális, míg jobb oldaluk maximális. Ez azt jelenti, hogy az azonos support-tal és megbízhatósággal rendelkező szabályok között ezek a szabályok tartalmazzák a legtöbb információt. MRG szabályokat minden esetben lehet találni, még nagyon magas minimum support értékek mellett is.

Úgy gondoljuk, hogy a ritka asszociációs szabályok a KDD egy nagyon érdekes és nagyon ígéretes kutatási területét képezik. A ritka asszociációs szabályokat sikerrel lehetne alkalmazni számos területen, úgymint marketing, üzleti alkalmazások, telekommunikáció, ill. különböző tudományterületeken, mint pl. biológia, asztronómia, orvostudomány, stb. Mivel ez még aránylag új területnek számít, a szakirodalom még nem foglalkozott vele részletesen. A mi munkánk ezáltal az elsők között szerepel.

8.2.5 A Coron rendszer

A dolgozatban bemutatott összes algoritmust implementáltuk, s egy egységes szoftver platformban, a CORON nevű rendszerben gyűjtöttük össze őket. A CORON egy többféle kutatási területen alkalmazható, platformfüggetlen, többcélú adatbányászati eszköztár, amely nem csupán számos adatbányászati algoritmust fog össze, de többféle kiegészítő szolgáltatást is nyújt, mint pl. az adatok előkészítése, tisztítása, ill. az eredmények megjelenítése, értelmezése. Legjobb tudomásunk szerint nem létezik még egy olyan, a CORON-hoz hasonló adatbányászati szoftver, mely direkt módon mintakeresésre, ill. asszociációs szabályok előállítására lett volna kifejlesztve.

A CORON rendszert legtöbbet egy valós orvosi adatbázison (STANISLAS cohort) alkalmaztuk. A tesztek során arra jöttünk rá, hogy szükségünk van **(1)** egy adatbányászati metodológiára, ill. **(2)** egy olyan eszközre, amely teljes mértékben megvalósítja a metodológia lépéseit. A 6. fejezetben található ez az adatbányászati metodológia, amely tetszőleges adatbázis esetén használható. A metodológia mind gyakori, mind ritka asszociációs szabályokra alkalmazható.

A 6. fejezet végén – a STANISLAS adatbázis mellett – még további három projektet is bemutatunk, melyek sikeresen alkalmazzák a CORON rendszert.

Appendix A

Test Environment

Test Platform

All the experiments in the thesis were carried out on the same Intel Pentium IV 2.4 GHz machine running under Debian GNU/Linux operating system with 512 MB of RAM. All the experiments were carried out with the CORON system. All times reported are real, wall clock times, as obtained from the Unix *time* command between input and output. Time values are given in seconds.

Test Databases

For testing and comparing the algorithms, we chose several publicly available real and synthetic databases to work with. Table A.1 shows the characteristics of these datasets, i.e. the name and size of the database, the number of transactions, the number of different attributes, the average transaction length, and the largest attribute in each database.

database name	database size (bytes)	# of transactions	# of attributes	# of attributes in average	largest attribute
T20I6D100K	7,833,397	100,000	893	20	1,000
T25I10D10K	970,890	10,000	929	25	1,000
C20D10K	800,020	10,000	192	20	385
C73D10K	3,205,889	10,000	1,592	73	2,177
MUSHROOMS	603,513	8,416	119	23	128

Table A.1: Database characteristics.

The T20I6D100K and T25I10D10K[55] are sparse datasets, constructed according to the properties of market basket data that are typically sparse, weakly correlated data. The number of frequent itemsets is small, and nearly all the FIs are closed. The C20D10K and C73D10K are census datasets from the PUMS sample file, while the MUSHROOMS[56] describes the characteristics of various species of mushrooms. The latter three are dense, highly correlated datasets. Weakly correlated data, such as synthetic data, constitute easy cases for the algorithms that extract frequent itemsets, since few itemsets are frequent. On the contrary, correlated data constitute far more difficult cases for the extraction due to the large number of frequent itemsets. Such data are typical of real-life datasets.

[55]`http://www.almaden.ibm.com/software/quest/Resources/`
[56]`http://kdd.ics.uci.edu/`

Appendix B

Classical Itemset Mining Algorithms

B.1 Apriori

Short Overview of the Apriori Algorithm

Apriori[57] is the first efficient algorithm for finding all frequent itemsets in a dataset [AS94, MTV94, AMS+96]. This is a levelwise, breadth-first, bottom-up algorithm. *Apriori* has been presented in a general way in Section 3.2.1, thus here we only present the algorithm and the experimental results. *Apriori* has been followed by lots of variations in order to improve different efficiency aspects [BMUT97, PCY95, SON95, Toi96].

The Algorithm

Pseudo code. The main block of the algorithm is given in Algorithm 20. *Apriori* uses two different kinds of tables, their description is provided in Tables B.1 and B.2. We assume that an itemset is an ordered list of attributes, since we will rely on this in the `Apriori-Gen` function (Algorithm 23). Note that we have this assumption for all levelwise algorithms presented in the thesis.

C_i	potentially frequent candidate i-itemsets fields: **(1)** itemset, **(2)** support
F_i	frequent i-itemsets fields: **(1)** itemset, **(2)** support

Table B.1: Tables used in *Apriori*.

itemset	–	an arbitrary itemset
support	–	support of the itemset

Table B.2: Fields of the tables of *Apriori*.

Running example. The execution of *Apriori* on dataset D (Table 3.1) with $min\_supp = 3$ (60%) is illustrated in Table B.3. The algorithm first performs a database scan to count the supports of 1-itemsets. The candidate itemset {D} is pruned because it is not frequent. At

[57]Agrawal and Srikant [AS94] and Mannila *et al.* [MTV94] independently proposed the same technique. The two works were cumulated afterwards in [AMS+96].

Algorithm 20 (Apriori):

```
Description:  finds all frequent itemsets
Input:        dataset + min_supp
Output:       all frequent itemsets

 1)  $C_1 \leftarrow$ {1-itemsets};
 2)  $i \leftarrow 1$;
 3)  while ($C_i \neq \emptyset$)
 4)  {
 5)      SupportCount($C_i$);
 6)      $F_i \leftarrow \{f \in C_i \mid f.\text{support} \geq min\_supp\}$; // $F$ – for frequent itemsets
 7)      $C_{i+1} \leftarrow$ Apriori-Gen($F_i$); // $C$ – for candidates
 8)      $++i$;
 9)  }
10)  $I_F \leftarrow \bigcup F_i$; // frequent itemsets
```

Algorithm 21 (SupportCount procedure):

```
Description:  counts the support of potentially frequent candidate itemsets
Input:        a $C_i$ table with potentially frequent candidate itemsets
Method:       the procedure fills the *support* field of the $C_i$ table

1)  loop over the objects of the input dataset ($o$)
2)  {
3)    $S \leftarrow$Subsets($C_i$, $o$); // such elements of $C_i$ that are subsets of $o$
4)    loop over the elements of $S$ ($s$):
5)      $++s$.support;
6)  }
```

the next iteration all candidate 2-itemsets are generated and stored in C_2. Then a database scan is performed to determine the supports of the six candidate itemsets. In C_3 there are four potentially frequent candidates, but after counting their supports it turns out that actually only two of them are frequent. Note that the support of the other two candidates had to be counted too because it was not known whether they are frequent or not. As all their subsets are frequent, they could have been frequent too. The itemsets of F_3 have no 4-long supersets, thus the algorithm stops. The union of F_i tables ($F_1 \cup F_2 \cup F_3$) gives the list of all frequent itemsets in D.

Experimental Results

Apriori was originally made to extract frequent itemsets from market basket data that are typically sparse, weakly correlated data. In these datasets, the number of frequent itemsets is small, and *Apriori* performs very well. Later on, the need for finding frequent itemsets in dense, strongly correlated data has risen, and it turned out that *Apriori* is not too efficient on such datasets. As a consequence, *Apriori* has been followed by lots of other mining algorithms. Some of them are some kind of optimizations / extensions of *Apriori* (*Apriori-Close*, *Pascal*, *Zart*), some of them are based on a different idea (*Eclat*, *Charm*), and some of them are a hybrid solution of these approaches (*Eclat-Z*, *Charm-MFI*).

Algorithm 22 (Subsets function):

Description:	in a set of itemsets finds the subsets of a given itemset
Input:	S – a set of itemsets
	l – an itemset whose subsets we are looking for in S
Output:	such elements of S that are subsets of l

This task can be solved very efficiently with the trie data structure (see Appendix C.2).

Algorithm 23 (Apriori-Gen function):

Description:	from a set of i-long frequent itemsets it generates their $(i+1)$-long supersets and only keeps the potentially frequent candidates
Input:	F_i – set of frequent i-long itemsets
Output:	table C_i with potentially frequent candidate itemsets

```
1) insert into $C_{i+1}$ // join step
   select $p[1], p[2], \ldots, p[i], q[i]$
   from $F_i\ p, F_i\ q$
   where $p[1] = q[1], \ldots, p[i-1] = q[i-1], p[i] < q[i]$;
2) loop over the rows of $C_{i+1}$ $(c)$ // prune step
3) {
4)   $S \leftarrow (i-1)$-long subsets of $c$;
5)   loop over the elements of $S$ $(s)$:
6)     if $(s \notin F_i)$ then $C_{i+1} \leftarrow C_{i+1} \setminus \{c\}$;
7) }
8) return $C_{i+1}$;
```

C_1	supp
{A}	4
{B}	4
{C}	4
{D}	1
{E}	4

F_1	supp
{A}	4
{B}	4
{C}	4
{E}	4

C_2	supp
{AB}	3
{AC}	3
{AE}	3
{BC}	3
{BE}	4
{CE}	3

F_2	supp
{AB}	3
{AC}	3
{AE}	3
{BC}	3
{BE}	4
{CE}	3

C_3	supp
{ABC}	2
{ABE}	3
{ACE}	2
{BCE}	3

F_3	supp
{ABE}	3
{BCE}	3

C_4	supp
$\emptyset$	

Table B.3: Execution of *Apriori* on dataset D with $min\_supp = 3$ (60%).

B.2 Apriori-Close

Overview of the Apriori-Close Algorithm

Apriori-Close was proposed in [PBTL99a]. This algorithm is an extension of *Apriori* and it can identify not only frequent, but *frequent closed itemsets* too simultaneously. The idea is the following. By definition, a closed itemset has no proper superset with the same support. At each i^{th} step all i-long frequent itemsets are marked as "closed". At the next $(i+1)^{th}$ iteration for each $(i+1)$-long itemset we test if it has an i-long subset with the same support. If so, then the i-long frequent itemset is not a closed itemset and we mark it as "not closed". When the algorithm terminates with the enumeration of all frequent itemsets, the itemsets still marked as "closed" are the frequent closed itemsets of the dataset. As a consequence, the largest frequent itemsets are always closed. As we will see it in the experimental results, this kind of filtering of closed itemsets does not induce any serious additional computation time.

Note that this algorithm is also called *Close*$^+$ in [Pas00a], but this name is quite misleading, because this algorithm is not an extension of *Close* [PBTL99c], but an extension of *Apriori*. Thus, we will refer to this algorithm as *Apriori-Close*.

Detailed Description of Apriori-Close

The idea of *Apriori-Close* has been adapted in our algorithm *Zart*, thus we present the pseudo code of *Apriori-Close* in Section 3.3.1.

Running example. The execution of *Apriori-Close* on dataset D (Table 3.1) with $min\_supp = 3$ (60%) is illustrated in Table B.4. For finding frequent itemsets it works exactly like *Apriori*. Filtering frequent closed itemsets is done the following way. First, all itemsets are marked "closed" in F_1. Then, using F_2, itemsets {B} and {E} are marked "not closed" because they have a proper superset with the same support ({BE})). All itemsets are marked "closed" in F_2, but {AB}, {AE}, {BC} and {CE} turn out to be "not closed" because they have a proper superset with the same support in F_3. At the end, there are 6 itemsets marked "closed": {A}, {C}, {AC}, {BE}, {ABE} and {BCE}. In dataset D exactly these itemsets are the frequent closed itemsets.

Experimental Results

We compared the efficiency of *Apriori-Close* with *Apriori*. The execution times of the algorithms on different datasets is illustrated in Table B.5. This table also shows the number of FIs, the number of FCIs, and the proportion of the number of FCIs to the number of FIs. Response times are presented graphically in Figures B.1, B.2 and B.3.

As a conclusion, we can say that *Apriori-Close* gives almost equivalent response times to *Apriori* on both weakly and strongly correlated data, i.e. the filtering of closed itemsets among frequent itemsets is not an expensive process.

C_1	supp
{A}	4
{B}	4
{C}	4
{D}	1
{E}	4

F_1	supp	closed
{A}	4	yes
{B}	4	~~yes~~
{C}	4	yes
{E}	4	~~yes~~

C_2	supp
{AB}	3
{AC}	3
{AE}	3
{BC}	3
{BE}	4
{CE}	3

F_2	supp	closed
{AB}	3	~~yes~~
{AC}	3	yes
{AE}	3	~~yes~~
{BC}	3	~~yes~~
{BE}	4	yes
{CE}	3	~~yes~~

C_3	supp
{ABC}	2
{ABE}	3
{ACE}	2
{BCE}	3

F_3	supp	closed
{ABE}	3	yes
{BCE}	3	yes

C_4	supp
$\emptyset$	

Table B.4: Execution of *Apriori-Close* on dataset D with $min\_supp = 3$ (60%).

min_supp (%)	Apriori	Apriori-Close	# FIs	# FCIs	$\frac{\#FCIs}{\#FIs}$
T20I6D100K					
2	72.67	71.14	378	378	100.00%
1	107.63	106.53	1,534	1,534	100.00%
0.75	134.49	132.88	4,710	4,710	100.00%
0.5	236.10	232.45	26,836	26,208	97.66%
0.25	581.11	585.04	155,163	149,217	96.17%
C20D10K					
50	61.18	62.67	1,823	456	25.01%
40	71.60	71.84	2,175	544	25.01%
30	123.57	125.27	5,319	951	17.88%
20	334.87	338.69	20,239	2,519	12.45%
10	844.44	860.21	89,883	8,777	9.76%
MUSHROOMS					
60	3.10	3.03	51	19	37.25%
50	6.03	5.99	163	45	27.61%
40	13.93	13.56	505	124	24.55%
30	46.18	54.70	2,587	425	16.43%
20	554.95	545.79	53,337	1,169	2.19%

Table B.5: Response times of *Apriori* and *Apriori-Close*.

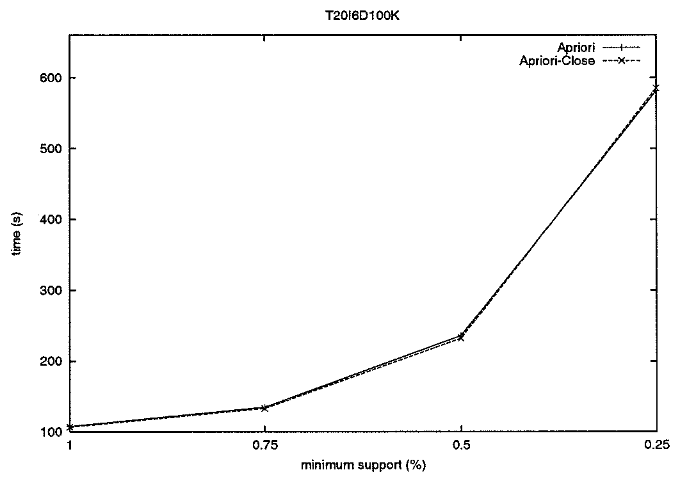


Figure B.1: Response times of *Apriori-Close* for T20I6D100K.

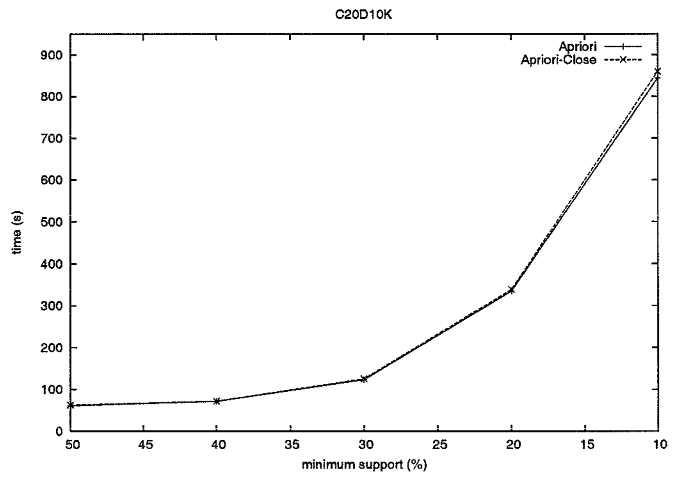


Figure B.2: Response times of *Apriori-Close* for C20D10K.

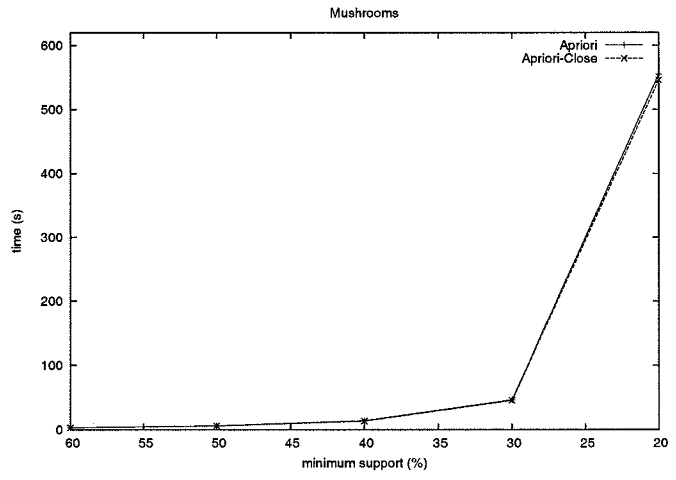


Figure B.3: Response times of *Apriori-Close* for MUSHROOMS.

B.3 Eclat

This appendix is based on Section 3.2.2, where we presented *Eclat* [ZPOL97, Zak00] in a general way. As seen, *Eclat* is a vertical algorithm that traverses a so-called itemset-tidset search tree (IT-tree) in a depth-first manner. The IT search tree of dataset D (Table 3.1) is depicted in Figure 3.2. The goal of *Eclat* is to find all frequent itemsets in this search tree. *Eclat* processes the input dataset in a vertical way, i.e. it associates to each attribute its tidset pair. Here we present the algorithm in detail through an example, and we provide experimental results.

The Algorithm

Eclat uses a special data structure for storing frequent itemsets called IT-search tree. This structure is composed of IT-nodes. An IT-node is an itemset-tidset pair, where an itemset is a set of items, and a tidset is a set of transaction identifiers. That is, an IT-node shows us which transactions (or objects) include the given itemset.

Pseudo code. The main block of the algorithm[58] is given in Algorithm 24. First the IT-tree is initialized, which includes the following steps: the root node, representing the empty set, is created. The support of the empty set is equal to the number of transactions in the dataset (100%). *Eclat* transforms the layout of the dataset in vertical format, and inserts under the root node all frequent attributes. After this the dataset itself can be deleted from the main memory since it is not needed anymore. Then we call the `extend` procedure recursively for each child of the root. At the end, all frequent itemsets are discovered in the IT-tree.

`addChild` procedure: this method inserts an IT-node under the current node.

`save` procedure: this procedure has an IT-node as its parameter. This is the method that is responsible for processing the itemset. It can be implemented in different ways, e.g. by simply printing the itemset and its support value to the standard output, or by saving the itemset in a file, in a database, etc.

`delete` procedure: this method deletes a node from the IT-tree, i.e.: it removes the reference on the node from its parent, and frees the memory that is occupied by the node.

`getCandidate` function: this function has two nodes as its parameters (*curr* and *other*). This function creates a new candidate node, i.e. it takes the union of the itemsets of the two nodes, and it calculates the intersection of the tidsets of the two nodes. If the support of the candidate is less than the minimum support, it returns null, otherwise it returns the candidate as an IT-node. In Section 3.2.2 we presented an optimization method for the support count of 2-itemsets. This technique can be used here: if the itemset part of *curr* and *other* consists of one attribute only, then the union of their itemsets is a 2-itemset. In this case, instead of taking the intersection of their tidsets, we consult the upper-triangular matrix to get its support. Naturally, this matrix had been built before in the initialization phase. In the "skipped intersections" column of Table B.6, it is indicated how many intersection operations can be saved thanks to this technique. Consult Appendix E for a detailed description and an example.

`sortChildren` procedure: this procedure gets an IT-node as parameter. The method sorts the children of the given node in ascending order by their support values. This step is highly recommended since it results in a much less number of rare candidates (see the last two columns of Table B.6 for the difference).

[58]Note that the main block of Charm is exactly the same.

Algorithm 24 (Eclat & Charm):

```
 1)  root.itemset ← $\emptyset$; // root is an IT-node whose itemset is empty
 2)  root.tidset ← {all transaction IDs}; // the empty set is present in every transaction
 3)  root.support ← $|O|$; // from the previous: support($\emptyset$)=100%
 4)  root.parent ← null; // the root has no parent node
 5)  loop over the vertical representation of the dataset ($attr$) {
 6)    if ($attr$.supp $\geq$ $min\_supp$) root.addChild(attr);
 7)  }
 8)  delete the vertical representation of the dataset; // free memory, not needed anymore
 9)  sortChildren(root); // optimization, results in a less number of rare candidates
10)
11)  while root has children
12)  {
13)    $child$ ← first child of root;
14)    extend($child$);
15)    save($child$); // processing the itemset
16)    delete($child$); // free memory, not needed anymore
17)  }
```

Algorithm 25 ("extend" procedure):

Method: extends an IT-node recursively (discovers FIs in its subtree)
Input: *curr* – an IT-node whose subtree is to be discovered

```
 1)  loop over the "brothers" (other children of its parent) of $curr$ ($other$)
 2)  {
 3)    $candidate$ ← getCandidate($curr$, $other$);
 4)    if ($candidate \neq$ null)
 5)      $curr$.addChild($candidate$);
 6)  }
 7)  sortChildren($curr$); // optimization, results in a less number of rare candidates
 8)
 9)  while $curr$ has children
10)  {
11)    $child$ ← first child of $curr$;
12)    extend($child$);
13)    save($child$); // processing the itemset
14)    delete($child$); // free memory, not needed anymore
15)  }
```

Running example. The execution of *Eclat* on dataset D (Table 3.1) with $min\_supp = 3$ (60%) is illustrated in Figure B.4. The execution order is indicated on the left side of the nodes in circles. For the sake of easier understanding, the element reordering optimization is not applied.

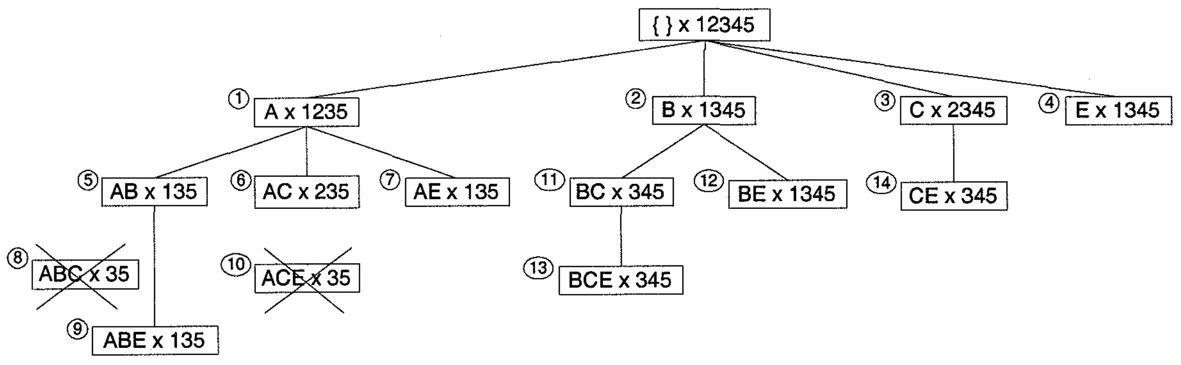


Figure B.4: Execution of *Eclat* on dataset D with $min\_supp = 3$ (60%).

The algorithm first initializes the IT-tree with the root node, which is the smallest itemset, the empty set, which is present in each transaction, thus its support is 100%. Using the vertical representation of the dataset, frequent attributes with their tidsets are added directly under the root. The children of the root node are extended recursively one by one. Let us see the prefix-based equivalence class of attribute A. This class includes all frequent itemsets that have A as their prefix. 2-long supersets of A are formed by using the "brother" nodes of A (nodes that are children of the parent of A, i.e. B, C and E). As AB, AC and AE are all frequent itemsets, they are added under A. The *extend* procedure is called now recursively on AB. Its first 3-long superset, ABC, is not frequent, thus it is not added in the IT-tree. ABE is frequent, thus it is added. During the extension of AC it turns out that ACE is not frequent either. With this, the subtree of A is completely discovered. After processing the nodes, this subtree can be deleted from main memory. Extension of nodes continues with B. When the algorithm stops, all frequent itemsets are discovered.

Experimental Results and Conclusion

We compared the efficiency of *Eclat* with *Apriori* and *Pascal.* These three algorithms are similar in the sense that they are specifically built to find all frequent itemsets. The execution times of the algorithms on different datasets is illustrated in Table B.6. Response times are presented graphically in Figures B.5, B.6 and B.7. As we can see, *Eclat* performs much better than levelwise algorithms, on both weakly and strongly correlated data.

In this appendix we presented the frequent itemset mining algorithm *Eclat* that is based on a different approach. *Eclat* is not a levelwise, but a depth-first, vertical algorithm. As such, it makes only one database scan. *Eclat* requires no complicated data structures, like trie, and it uses simple intersection operations to generate candidate itemsets (candidate generation *and* support counting happen in a single step). *Apriori* has been followed by lots of optimizations, extensions. The same is true for *Eclat.* Experimental results show that *Eclat* outperforms levelwise, frequent itemset mining algorithms. It also means that *Eclat* can also be used on such datasets that other levelwise algorithms cannot simply handle.

min_supp (%)		Apriori	Pascal		Eclat	skipped intersections	+ reordering	− reordering
T20I6D100K								
2		72.67	71.15		7.05	67,887	0	3
1		107.63	106.24		7.77	169,012	753	2,251
0.75		134.49	132.00		9.92	208,448	4,500	13,036
0.5		236.10	228.37		22.56	250,136	27,698	75,087
0.25		581.11	562.47		119.22	301,114	274,685	641,676
C20D10K								
50		61.18	16.68		1.09	0	3	80
40		71.60	19.10		1.14	11	0	117
30		123.57	26.74		1.35	22	33	2,078
20		334.87	53.28		2.33	220	18	4,484
10		844.44	110.78		6.53	512	241	20,912
MUSHROOMS								
60		3.10	2.04		0.81	10	2	7
50		6.03	3.13		0.79	35	6	29
40		13.93	6.00		0.84	118	39	146
30		46.18	12.79		1.00	189	132	722
20		554.95	30.30		3.08	491	482	4,280

Table B.6: Response times of *Eclat*. The column "skipped intersections" indicates the number of intersection operations that can be saved in the case of 2-itemsets using the upper-triangular matrix optimization (Appendix E). The last two columns show the number of rare candidates generated by *Eclat* with (+) and without (−) element reordering, respectively.

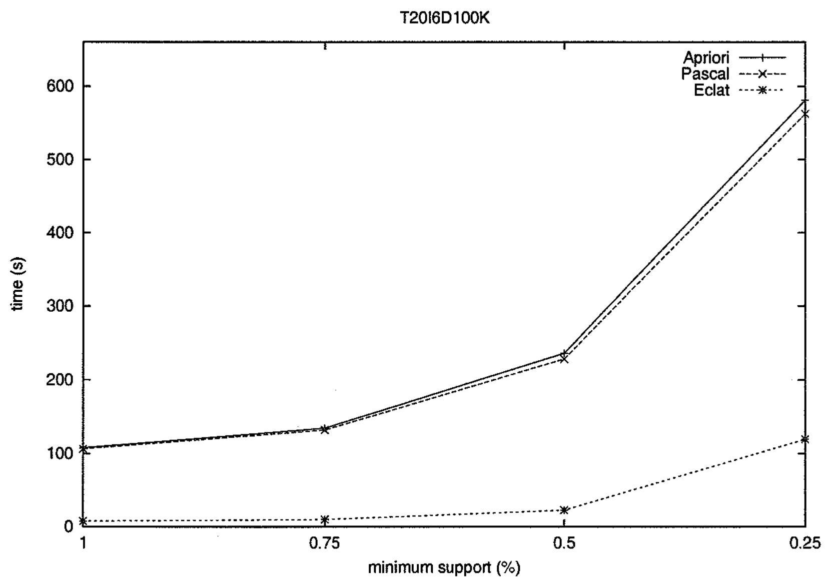


Figure B.5: Response times of *Eclat* for T20I6D100K.

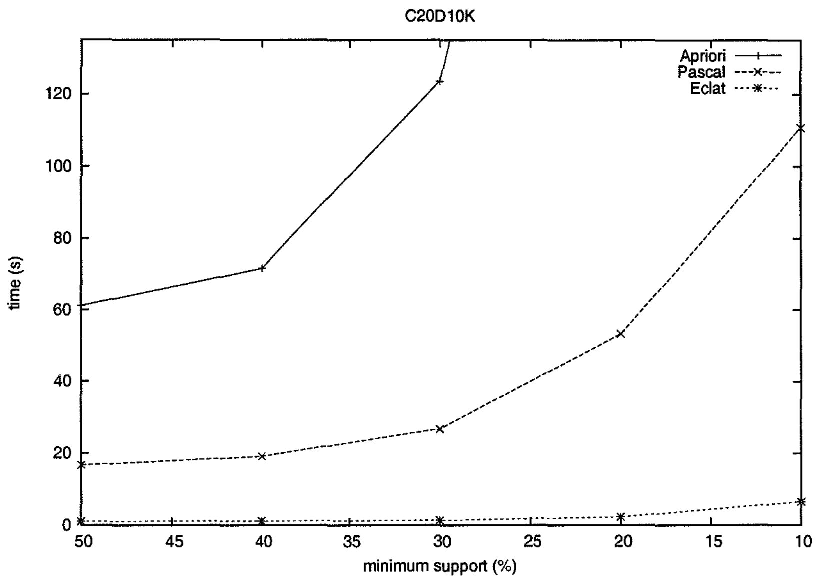


Figure B.6: Response times of *Eclat* for C20D10K.

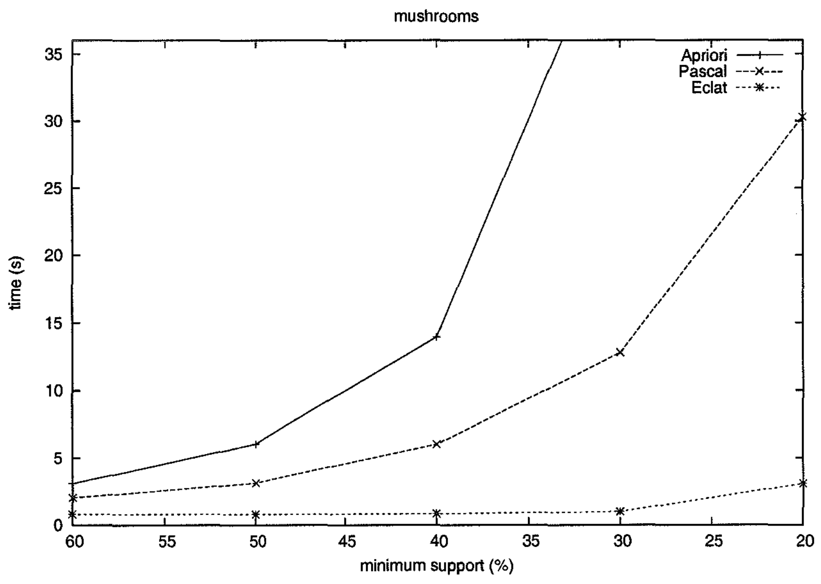


Figure B.7: Response times of *Eclat* for MUSHROOMS.

B.4 Charm

This appendix is based on Section 3.2.2, where we presented the common parts of *Eclat* and *Charm* [ZH02]. *Eclat* was designed to find all frequent itemsets in a dataset. *Charm* is a modification of *Eclat*, allowing one to find frequent *closed* itemsets only.[59] Since *Charm* is based on *Eclat*, reading Appendix B.3 is highly recommended for an easier understanding. *Charm* is a vertical algorithm that traverses the itemset-tidset search tree (IT-tree) in a depth-first manner. The IT search tree of dataset D (Table 3.1) is depicted in Figure 3.2. The goal of *Charm* is to find frequent closed itemsets only in this search tree. *Charm*, just like *Eclat*, processes the input dataset in a vertical way, i.e. it associates to each attribute its tidset pair. Here we present the algorithm in detail. For the example we will use another dataset D', depicted in Table B.7, because D (Table 3.1) is not appropriate to show a speciality of *Charm*, namely the subsumption check (see later). This subsection mainly relies on [ZH02], where the proof of Theorem B.1 can also be found.

	A	B	C	D	E
1					x
2	x				
3	x		x	x	
4	x	x	x	x	x
5	x	x		x	x
6		x	x	x	x

Table B.7: Another dataset (D') for *Charm*.

Basic Properties of Itemset-Tidset Pairs

There are four basic properties of IT-pairs that *Charm* exploits for efficient exploration of closed itemsets. Assume that we are currently processing a node $P \times t(P)$, where $[P] = \{l_1, l_2, \ldots, l_n\}$ is the prefix class. Let X_i denote the itemset Pl_i, then each member of $[P]$ is an IT-pair $X_i \times t(X_i)$.

Theorem B.1 *Let $X_i \times t(X_i)$ and $X_j \times t(X_j)$ be any two members of a class $[P]$, with $X_i \leq_f X_j$, where f is a total order (e.g. lexicographic or support-based). The following four properties hold:*

1. *If $t(X_i) = t(X_j)$, then $\gamma(X_i) = \gamma(X_j) = \gamma(X_i \cup X_j)$*
2. *If $t(X_i) \subset t(X_j)$, then $\gamma(X_i) \neq \gamma(X_j)$, but $\gamma(X_i) = \gamma(X_i \cup X_j)$*
3. *If $t(X_i) \supset t(X_j)$, then $\gamma(X_i) \neq \gamma(X_j)$, but $\gamma(X_j) = \gamma(X_i \cup X_j)$*
4. *If $t(X_i) \neq t(X_j)$, then $\gamma(X_i) \neq \gamma(X_j) \neq \gamma(X_i \cup X_j)$*

The Algorithm

Pseudo code. The main block of the algorithm is exactly the same as the main block of *Eclat* (see Algorithm 24), thus we do not repeat it here. The difference is in the `extend` procedure (Algorithm 26). While *Eclat* finds all frequent itemsets in the subtree of a node, *Charm* concentrates on frequent closed itemsets only.

[59]Actually, this algorithm could also be called Eclat-Close, or simply Eclat-C.

The initialization phase is equivalent to *Eclat*'s: first the root node is created that represents the empty set. By definition, the empty set is included in every transaction, thus its support is equal to the number of transactions in the dataset (100%). *Charm* transforms the layout of the dataset in vertical format, and inserts under the root node all frequent attributes. After this, the dataset itself can be deleted from main memory since it is not needed anymore. Then the `extend` procedure is called recursively for each child of the root. At the end, all frequent *closed* itemsets are discovered in the IT-tree.

Algorithm 26 ("extend" procedure):

Method: extends an IT-node recursively (discovers FCIs in its subtree)
Input: *curr* – an IT-node whose subtree is to be discovered

```
loop over the "brothers" (other children of its parent) of curr (other)
{
  if (curr.tidset = other.tidset) { // Property 1
    delete(other);
    replaceInSubtree(curr, other.itemset);
  }
  else if (curr.tidset $\subset$ other.tidset) { // Property 2
    replaceInSubtree(curr, other.itemset);
  }
  else if (curr.tidset $\supset$ other.tidset) { // Property 3
    candidate $\leftarrow$ getCandidate(curr, other);
    delete(other);
    if (candidate $\neq$ null)  curr.addChild(candidate);
  }
  else { // if (curr.tidset $\neq$ other.tidset) // Property 4
    candidate $\leftarrow$ getCandidate(curr, other);
    if (candidate $\neq$ null)  curr.addChild(candidate);
  }

  sortChildren(curr); // optimization, results in a less number of rare candidates

  while curr has children
  {
    child $\leftarrow$ (first child of curr);
    extend(child);
    save(child); // processing the itemset
    delete(child); // free memory, not needed anymore
  }
}
```

The following methods are equivalent to the methods of *Eclat* with the same name: `addChild`, `delete`, `getCandidate`, `sortChildren`. Their description can be found in Appendix B.3.

`replaceInSubtree` procedure: it has two parameters, an IT-node (*curr*), and an itemset X (the itemset part of another node). The method is the following: let Y be the union of X and the itemset part of *curr*. Then, traverse recursively the subtree of *curr*, and replace everywhere the itemset of *curr* (as a sub-itemset) with Y.

save procedure: this procedure is a bit different from the procedure described in *Eclat*. First, it must be checked whether the current itemset is closed or not. It can be done by testing if a proper superset of the current node with the same support was found before. If yes, then the current node is not closed. If the test is negative, i.e. the current itemset is closed, we can process the itemset as we want (print it to the standard output, save it in a database, etc.).

Running example. The execution of *Charm* on dataset D' (Table B.7) with $min\_supp = 3$ (50%) is illustrated in Figure B.8. The execution order is indicated on the left side of the nodes in circles. For the sake of easier understanding, the element reordering optimization is not applied.

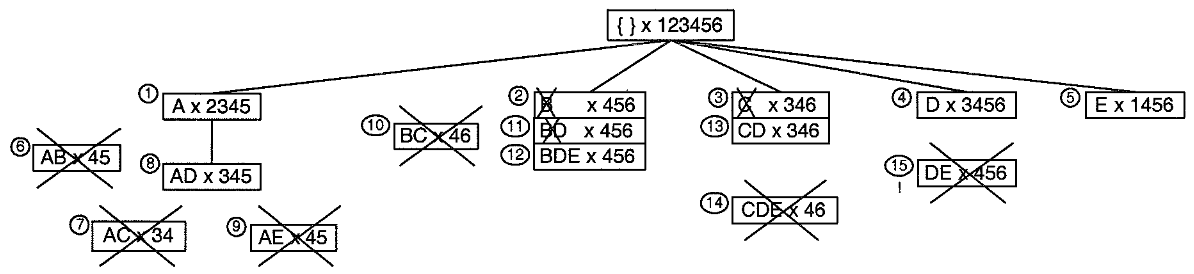


Figure B.8: Execution of *Charm* on dataset D' with $min\_supp = 3$ (50%).

The algorithm first initializes the IT-tree with the root node, and adds all frequent attributes under it. The children of the root node are extended recursively one by one. *Extending A.* The itemsets AB and AC are rare. The tidsets of A and D have no relation (Property 4), thus AD is generated and inserted under A. The itemset AE is also rare. *Extending B.* The itemset BC is rare. The tidset of B is a proper subset of the tidset of D, thus in the subtree of node B the sub-itemset "B" is replaced by "BD" everywhere. Since the subtree of node B consists of one node only, this replacement only concerns one node ($B \times 456$ becomes $BD \times 456$). As the tidset of BD is a proper subset of the tidset of E again, the same procedure is repeated, and $BD \times 456$ becomes $BDE \times 456$. *Extending C.* Like in the previous two cases, applying Property 2 results in the node $CD \times 346$. The itemset CDE is rare. *Extending D.* After applying Property 4, we get the itemset DE. With this itemset there is a "problem": although it is frequent, this itemset is not closed because we already found a superset of it with the same support (BDE), thus DE is not added to the IT-tree. When the algorithm stops, all frequent closed itemsets are discovered.

Fast subsumption checking. Let X_i and X_j be two itemsets. We say that X_i *subsumes* X_j (or X_j is *subsumed by* X_i), iff $X_j \subset X_j$ and $supp(X_i) = supp(X_j)$. Recall that in the **save** procedure, before adding an itemset X to the set of closed itemsets, *Charm* checks if X is subsumed by a previously found closed itemset. In other words, *Charm* can find itemsets that are actually *not* closed itemsets. It might seem to be a problem, but Zaki managed to find a very efficient way to filter these non-closed itemsets.

Zaki proposes a hash structure for storing FCIs in order to perform fast subsumption checking. It also means that *Charm* stores the found frequent closed itemsets in the main memory. The idea is the following. *Charm* computes a hash function on the tidset and stores in the hash table a closed set with its support value. Let $h(X_i)$ denote the hash function on the tidset of X_i. This hash function has one important criteria: it must return the same value for itemsets that are included by the same set of objects. Several hash functions could be possible, but *Charm* uses the sum of the tids in the tidset (note that this is not the same as support, which is the cardinality of the tidset). Itemsets having the same hash value are stored in a list at the same position of the hash. Before adding X to the set of closed itemsets, we retrieve from the hash table all entries with the hash key $h(X)$. For each element C in this list check if $supp(X) = supp(C)$. If yes, check if $X \subset C$. If yes, then X is subsumed by C, and we do not register X in the hash table.

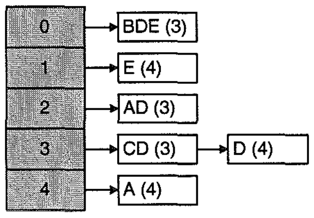


Figure B.9: Hash table for the IT-tree in Figure B.8.

Let us see Figure B.9 that depicts the hash structure of the IT-tree in Figure B.8. For this example, the size of the hash table is set to five.[60] At each position of the hash table there are pointers to lists. In each list we can find itemsets that have the same hash key. In the running example we saw that DE is not closed. Using the hash table it can be determined the following way. First, compute the sum of the tids in its extent $(4 + 5 + 6 = 15)$; then modulo this sum by the size of the hash table to get its hash value: $15 \bmod 5 = 0$. Traverse the list of the hash table at position 0. We find that BDE has the same support value as DE, thus check if DE is a subset of BDE. As the answer is positive, it means that DE is not closed.

Experimental Results

We compared the efficiency of *Charm* with *Close* and *Pascal*$^+$. *Charm* and *Close* are specifically built to find FCIs, while *Pascal*$^+$ can filter FCIs among FIs.

Close was the first algorithm that concentrated on frequent closed itemsets instead of the whole set of frequent itemsets [PBTL99c]. *Close* is a levelwise algorithm that identifies the closure of candidate generators. If it finds a closure that already has been explored, *Close* deletes it together with its generator. Thanks to this technique, *Close* can greatly reduce the number of candidate generators in highly correlated data. However, computing closures this way is an expensive operation. This algorithm is also implemented in our platform CORON.

Pascal$^+$ was introduced in Section 3.3.1. *Pascal*$^+$ is based on *Pascal*. First it finds all frequent itemsets, then it marks closed itemsets. Experimental evaluations show that *Pascal*$^+$ gives almost equivalent response times to *Pascal*.

The execution times of the algorithms on different datasets is illustrated in Table B.8. Response times are presented graphically in Figures B.10, B.11 and B.12. On sparse datasets, like T20I6D100K, *Charm* is about twice as fast. However, on dense, highly correlated datasets there is a huge difference between vertical and levelwise approaches. It is also interesting that *Pascal*$^+$, even though it finds both frequent and frequent closed itemsets, does not perform much worse than *Close*.

[60]In our implementation, we set the size of the hash table to 100,000.

min_supp (%)		Charm	Close	Pascal$^+$
T20I6D100K				
2		32.80	79.37	71.23
1		73.25	121.66	106.84
0.75		91.77	148.89	133.54
0.5		135.87	251.69	231.99
0.25		361.96	615.51	575.71
C20D10K				
50		1.35	17.11	16.69
40		1.55	19.55	19.03
30		1.60	27.13	26.69
20		2.28	51.18	54.56
10		3.99	103.29	118.60
MUSHROOMS				
60		0.86	1.94	2.07
50		0.86	2.77	3.18
40		0.91	5.01	6.05
30		1.12	10.15	12.99
20		1.43	20.22	35.88

Table B.8: Response times of *Charm*.

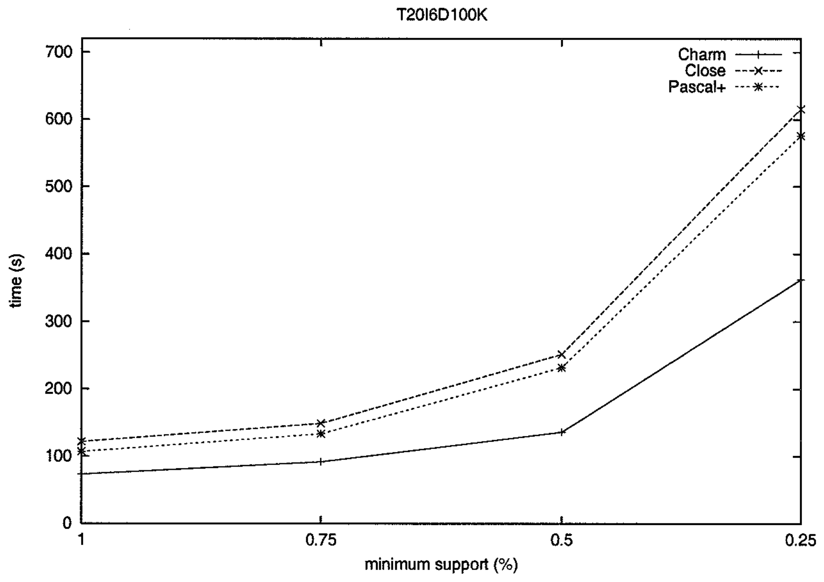


Figure B.10: Response times of *Charm* for T20I6D100K.

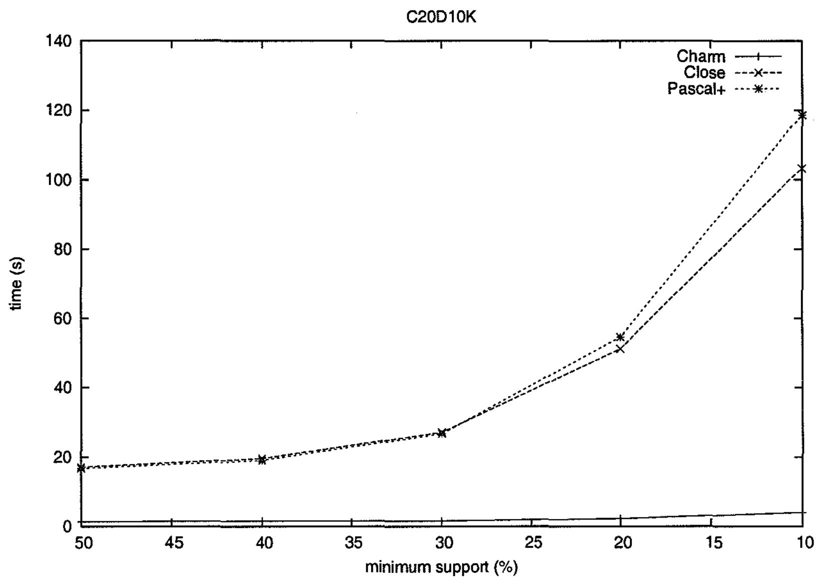


Figure B.11: Response times of *Charm* for C20D10K.

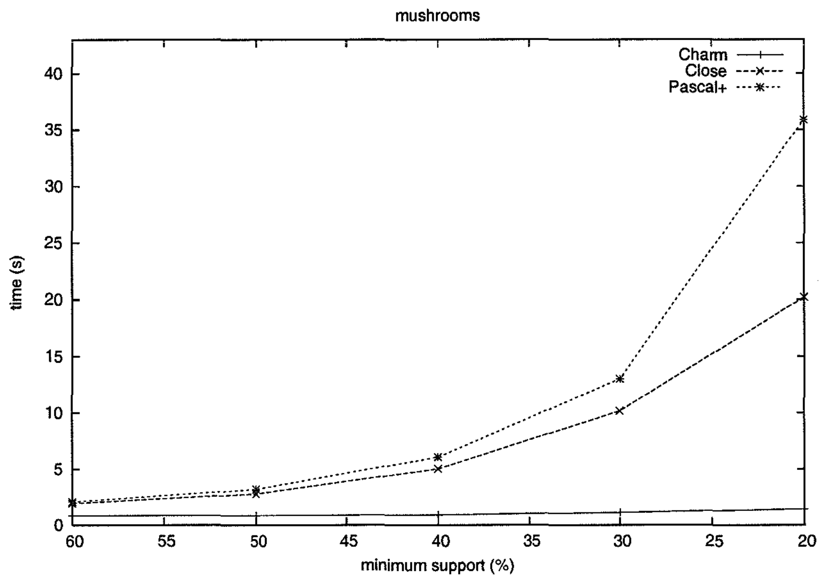


Figure B.12: Response times of *Charm* for MUSHROOMS.

Appendix C

The Trie Data Structure

C.1 The Trie Data Structure

The *trie* (or *prefix tree*) data structure is a tree for storing strings in which each node corresponds to a prefix [AHU85]. The root is associated with the empty string. The descendants of each node represent strings that begin with the prefix stored at that node. The name of the data structure comes from the word "re**trie**val" and is pronounced as "try" in order to distinguish it from the more general "tree". A common application of a trie is that of storing dictionary words, where there are usually lots of words with the same prefix. Tries represent words in a very compact way, and they allow for very fast word lookup, insertion and deletion.

Example. Store the following words in a trie (see Figure C.1): *tar*, *tardily*, *tardy*, *target*, *tack*, *temp*.

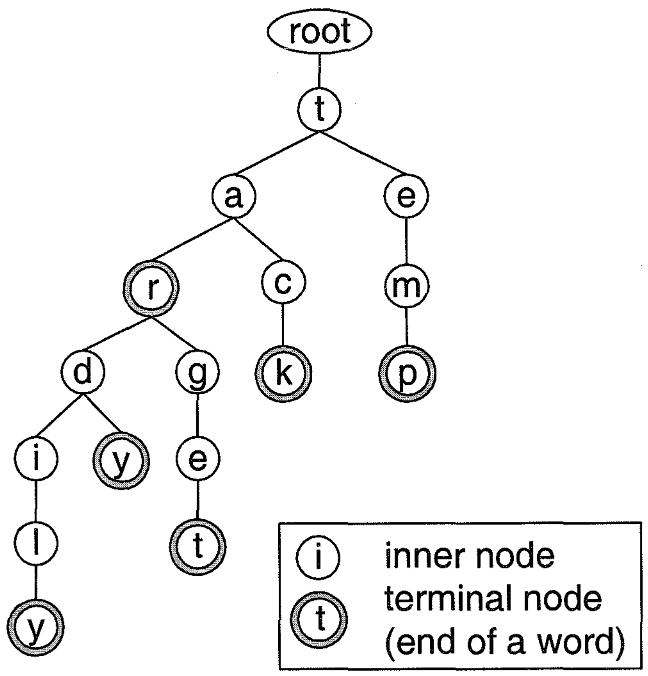
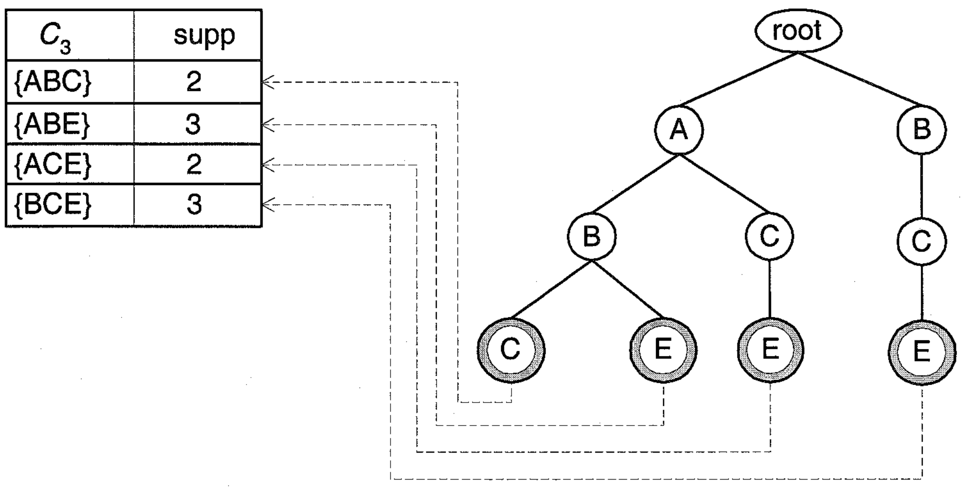


Figure C.1: Trie of dictionary words.



C_3	supp
{ABC}	2
{ABE}	3
{ACE}	2
{BCE}	3

Figure C.2: Trie of itemsets.

Our structure is derived from the one proposed in [PBTL99c]. In our implementations, itemsets with their associated descriptions (e.g. support value) form a `Row` object. `Row` objects are stored in tables. If an operation requires a trie (e.g. subsets or supersets functions), then a trie is built over the itemsets. It is important to note that all itemsets are sorted in *lexicographic* order. Each node in the trie has a value, which is a 1-long item (an attribute). Because of lexicographic order, the value of each child is *greater* than the value of its parent. An itemset is represented as a path in the trie, starting from the root node. Each node has a pointer back to its parent. Children of a node are stored in a hash table. Each terminal node (marked with double circles) has a pointer to its corresponding `Row` object. In our pseudo codes we use the following terminologies: *node*.value is the item stored in the node. In the case of terminal nodes, *node*.itemset refers to the corresponding itemset on which the node points to.

Example. Figure C.2 shows the trie that is built upon the table C_3 of Table B.3.

C.2 Subset Function with a Trie

The Algorithm

Here we present an efficient algorithm for finding all subsets of a given set using the trie data structure.[61] Our algorithm is derived from the one proposed in [PBTL99c].

Pseudo code. The pseudo code of the algorithm is given in Algorithm 27. The `findSubsetsOf` method is a recursive method with three parameters. The first parameter is the given set whose subsets are to be found. The second parameter is the current node, which is the root of the subtree in which the subsets are searched for. Initially, the algorithm is called with the root of the trie. The subsets found by the algorithm are collected in a list that is a global variable. This list is empty initially.

[61]Equality ($\subseteq$) is allowed.

Algorithm 27 (subsets with a trie):

Description:	finds all subsets of a given set in a trie
Input:	trie – trie of itemsets
	set – the set whose subsets we are searching for
Output:	all subsets of the given set

```
$subsets \leftarrow \emptyset$;
findSubsetsOf($set$, $trie$.root, $subsets$);
return $subsets$;

void findSubsetsOf($set$, $currNode$, $subsets$)
{
    if ($currNode$ == null) return; // stopping recursion
    // else
    $currNode$.label $\leftarrow$ $set$;
    if ($currNode$ is a terminal node) then $subsets \leftarrow subsets \cup currNode$.itemset;
    // where $currNode$.itemset is the corresponding itemset of the current node

    loop over the elements of $currNode$.label ($attr$)
    {
        $childNode \leftarrow currNode$.getChild($attr$); // get child whose value is $attr$
        if ($childNode$ != null)
        {
            $setToPass \leftarrow$ ($currNode$.label \ $attr$);
            findSubsetsOf($setToPass$, $childNode$, $subsets$);
        }
    }
}
```

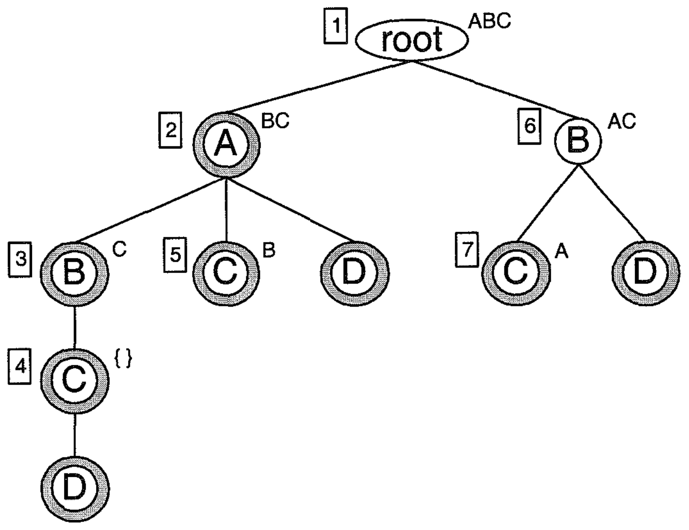


Figure C.3: Subset function with a trie.

Example. Suppose we have the following itemsets: A, AB, AC, AD, BC, BD, ABC and $ABCD$. Among these itemsets we want to find all subsets of ABC, i.e. A, AB, AC, BC and ABC.

Figure C.3 shows the trie that contains the eight itemsets. Each node in the trie has a value (i.e. an item) and it is shown as a letter inside the node. Terminal nodes are marked with double circles. The numbers in the boxes represent the order in which the nodes are visited. We will use these numbers also to reference the nodes of the trie. For instance, node 1 is the root node, node 2 is the left-most child node of the root with value A, etc. The algorithm labels each node it visits. A node's label appears in the top right-hand corner of the node. Henceforth, we will write "subset" for any subset of ABC. Any subset that is found is stored in a list.

First, the root is labeled with the set whose subsets we are searching for (ABC). The first item of ABC is A, and the root has a child node (node 2) with the value A, so we pass $ABC \setminus A = BC$ to node 2. Since node 2 is a terminal node, it is a subset. Node 2 has a child node whose value is B (node 3), thus $BC \setminus B = C$ is passed to node 3. Again, node 3 is a subset. Node 3 has a child node with value C (node 4), and so we pass $C \setminus C = \emptyset$ to node 4, which is also a subset. We come back through recursion to node 2, and now $BC \setminus C = B$ is passed to node 5 (since it has the value B). Node 5, being a terminal node, is therefore also a subset. The remaining child node of node 2 has value D and the item D is not in the label of node 2 (BC). Hence it is not visited. We return to root via recursion. Now we pass $ABC \setminus B = AC$ to the child node of root with value B, which is node 6. This is not a terminal node, and so it is not a subset. Node 6 has a child node with the value C (node 7). Thus $AC \setminus C = A$ is passed to node 7. Node 7 is a terminal node, and so it is a subset. Recursing back to root, we find that the root does not have a child node with the value C. Therefore, the algorithm terminates.

In this method equality is allowed. If we need to find *proper* subsets of a given set, we simply need to remove the given set from the answer set.

C.3 Superset Function with a Trie

The Algorithm

Here we propose an efficient algorithm for finding all supersets of a given set using the trie data structure.[62] The proposed algorithm also gives a very efficient solution if we only need the smallest proper superset of a given set (see Property 4.1). Below we show three different uses of the algorithm.

(1) Finding *all supersets* of a given set. Our algorithm traverses the trie in a breadth-first manner (BFT). As a consequence, supersets are produced in ascending order by itemset length. In order to reduce the search space, we rely on the property of our trie that itemsets are ordered lexicographically. As we will see in the given example, at a certain point it is guaranteed that no supersets are present in the subtree of a node, allowing the trie to be cut at that node.

(2) Finding *all proper supersets* of a given set. All proper supersets of a given set can be found by first finding the set of its supersets and then simply removing the given set from this set.

[62]Equality ($\supseteq$) is allowed.

(3) Finding the *smallest proper superset* of a given set. Due to breadth-first traversal, supersets are produced in ascending order of length. To find the smallest proper superset, all supersets need not be found. Instead, when the first superset is found, it is tested to see if it is larger than the given set. If it is, then the search process can be stopped, since this implies that the smallest proper superset has been found. However, if the given set is present in the trie, it will be found the very first time (since $\forall S,\ S \supseteq S$). In this case, the search is continued till the next superset is found.

Pseudo code. The pseudo code of this algorithm is given in Algorithm 28. In its present form, the algorithm finds all supersets of a given set, as explained above under **(1)**. The algorithm can be easily modified for the other two cases.

In breadth-first traversal, two kinds of nodes are distinguished. A node is said to be *closed* if it has already been visited and processed. A node is called *open* if it is "visible" from a closed node but it has not yet been visited and processed. All the other nodes in the search tree are not (yet) discovered. Note that in Algorithm 28 we use the term "closed" in this sense.[63]

[63]Not to be confused with "closed itemsets".

Algorithm 28 (supersets with a trie):

Description: finds all supersets of a given set in a trie
Input: trie – trie of itemsets
set – the set whose supersets we are looking for
Output: all supersets of the given set

```
 1)  supersets ← ∅;
 2)  mark all nodes in trie as "not yet discovered"; // initialization for the BFT
 3)  mark trie.root open; // initialization for the BFT
 4)  trie.root.label ← set;
 5)
 6)  loop:
 7)  while there are open nodes in trie
 8)  {
 9)      currNode ← first open node in trie; // choose it by breadth-first
10)
11)      if (currNode is not the root of trie)
12)      {
13)          if (currNode.label is not empty)
14)          {
15)              if (currNode.value == first element of currNode.label) {
16)                  delete first element of currNode.label;
17)              }
18)              else if (currNode.value > first element of currNode.label)
19)              {
20)                  make currNode closed;
21)                  continue loop; // continue execution at line 6
22)                  // Stop the exploration of this subtree. Because of lexicographic order,
23)                  // it is sure that we will not find any supersets in this subtree.
24)              }
25)          }
26)
27)          if ( (currNode.label is empty) and (currNode is a terminal node) )
28)              supersets ← supersets ∪ currNode.itemset;
29)          }
30)      }
31)
32)      loop over the children of currNode (child) {
33)          child.label ← currNode.label;
34)          make child open;
35)      }
36)      make currNode closed;
37)  } // end while
38)
39)  return supersets;
```

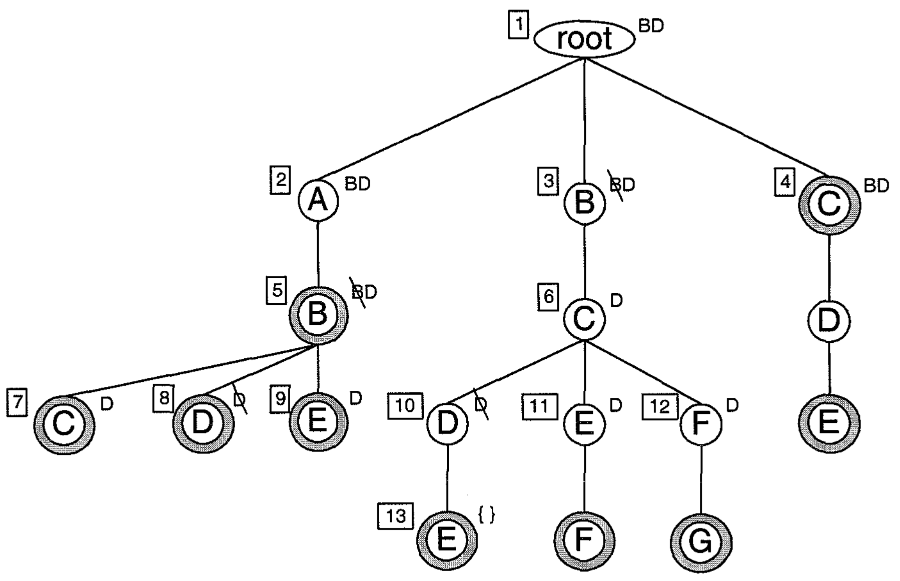


Figure C.4: Superset function with a trie.

Example

Suppose we have the following itemsets: *AB*, *ABC*, *ABD*, *ABE*, *BCDE*, *BCEF*, *BCFG*, *C* and *CDE*. Among these itemsets we want to find all supersets of *BD*, i.e. *ABD* and *BCDE*. Figure C.4 shows the trie that contains the nine itemsets.

Each node in the trie has a value (i.e. an item) and it is shown as a letter inside the node. Terminal nodes are marked with double circles. The numbers in the boxes represent the order in which the nodes are visited. We will use these numbers also to reference the nodes of the trie. For instance, node 1 is the root node, node 2 is the left-most child node of the root with value *A*, etc. The algorithm labels nodes. A node's label appears in the top right-hand corner of the node. Henceforth, we will write "superset" for any superset of *BD*. Any superset that is found is stored in a list.

Labels are an essential part of the algorithm. Let $\mathcal{S}$ be the *search set*, i.e. the set whose supersets are to be found, and let $\mathcal{L}$ be the label of a node. Then:

1. The set $\mathcal{S} \setminus \mathcal{L}$ contains the elements that have already been found in the current path.[64]

2. The set $\mathcal{L}$ contains the elements that are still to be found by extending the search path.

For instance, merely by looking at the label of node 6, it is evident that the current path (root $\rightarrow$ node 3 $\rightarrow$ node 6) contains *B* and does not contain *D*. If a node *n* has an empty label, it implies that all the elements of the search set have been found. It also implies that all the terminal nodes of the sub-tree starting at node *n* are supersets of the search set.

Problem: find all supersets of *BD* in the trie of Figure C.4. First, the trie is initialized for breadth-first traversal: all nodes are marked as "not yet discovered"; the root is marked "open";

[64]The "current path" is the path from the root to the current node.

the root is labeled with the search set (here, BD). Children of the root node are marked "open" and the label BD is passed to them. Now, root is marked "closed". Since we use breadth-first traversal, the first open node is node 2. Node 5 is labeled with BD and it is marked "open". Now node 2 is marked "closed". The value of node 3 is equal to the first element of its label, thus "B" is removed from its label. Its new label, D, is passed to node 6 and node 6 is marked "open". Now node 3 is marked "closed". Node 4's value, C, is larger than the first element of its label (lexicographically $C > B$). Because of the lexicographic ordering, it is not possible to find a superset in the subtree of node 4, thus the trie is cut at node 4. The subtree of node 4 will not be traversed. Node 4 is marked "closed". The value of node 5 is equal to the first element of its label. The new label of node 5, D, is passed to its children, and the children are marked "open" while node 5 is marked "closed". From node 6, D is passed to its children, which are marked "open" and node 6 is marked "closed". Node 7 is marked "closed". In node 8, the value of the node is equal to the first element of its label, thus its label is changed to the empty set. Since its label is empty and node 8 is a terminal node, the corresponding itemset (ABD) is a superset. Node 8 is marked "closed". At node 9 the trie is cut and it is marked "closed". In node 10 the label is changed to the empty set. As node 10 is not a terminal node, it is not a superset. Node 10 is marked "closed" and node 13 is marked "open". Node 13 is labeled by the empty set. The trie is cut at nodes 11 and 12. Node 13 has an empty label and it is a terminal node, thus $BCDE$ is a superset.

Appendix D

Horizontal and Vertical Data Layouts

Most itemset mining algorithms use a *horizontal* database layout, such as the one shown in Figure D.1, consisting of a list of transactions (or objects), where each transaction has an identifier followed by a list of items that are included in that transaction. Some algorithms, like *Eclat* or *Charm*, use a *vertical* database layout, such as the one shown in Figure D.1, consisting of a list of items (or attributes), where each item is associated with a list of transactions that include the given item. One layout can easily be converted into the other on-the-fly, with very little cost. This process requires only a trivial amount of overhead.

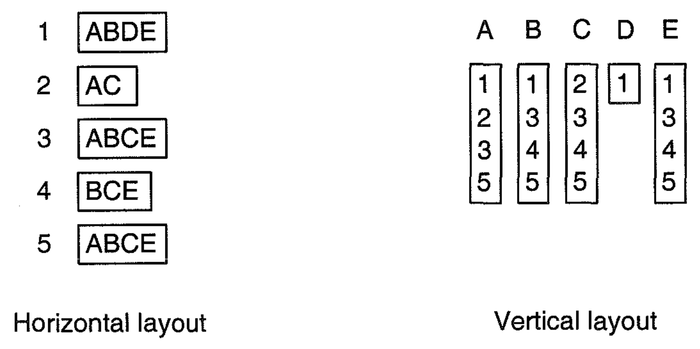


Figure D.1: Horizontal and vertical layouts of dataset D (Table 3.1).

Horizontal to Vertical Database Transformation

For each transaction t, we read its item list. During the transformation process, we build an array that is indexed by items of the database. We insert the ID of t in those positions of the array that are indexed by the items present in the associated list of t.

Example. Consider the item list of transaction 1, shown in Figure D.1. We read its first item, A, and insert 1 in the array indexed by item A. We repeat this process for all other items in the list and for all other transactions. Figure D.2 shows the transformation process step by step.[65]

[65]In the figure, "tid" stands for "transaction ID".

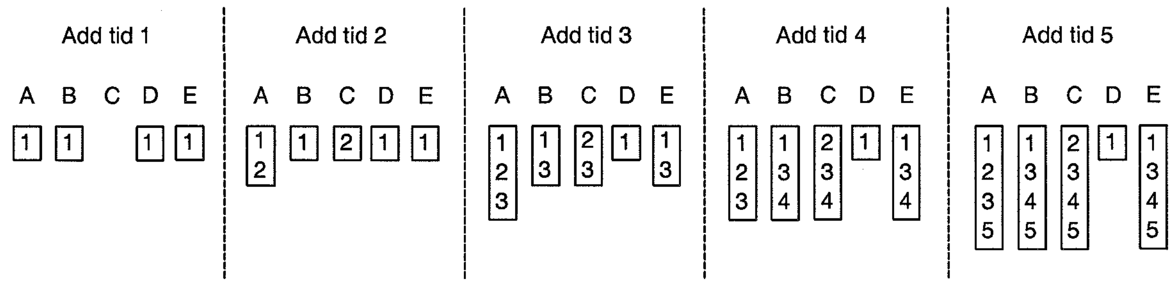


Figure D.2: Horizontal to vertical database transformation.

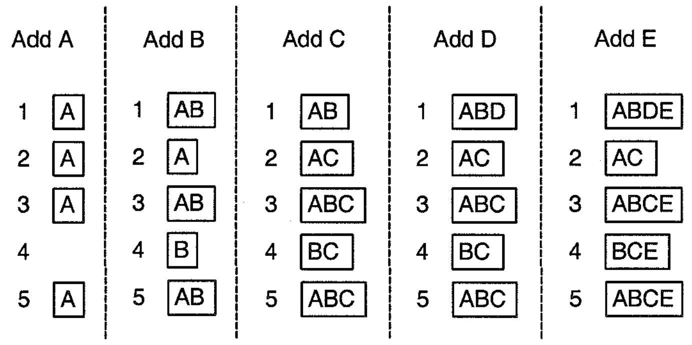


Figure D.3: Vertical to horizontal database transformation.

Vertical to Horizontal Database Transformation

For each item i, we read its transaction list. During the transformation process, we build an array that is indexed by transaction IDs. We insert item i in those positions of the array that are indexed by the transactions present in the associated list of i.

Example. Consider the transaction list of item A, shown in Figure D.1. We read its first transaction ID, 1, and insert A in the array indexed by transaction 1. We repeat this process for all other transaction IDs in the list and for all other items. Figure D.3 shows the transformation process step by step.

Appendix E

Efficient Support Count of 2-itemsets

Here we present an optimization of the support count of 2-itemsets. This method was proposed by Zaki in [ZH02] for the *Charm* algorithm. However, this optimization can be used with *Eclat* and with breadth-first algorithms too, such as *Apriori.*

In the case of vertical algorithms (e.g. *Eclat*, *Charm*), this method significantly reduces the number of intersection operations. The idea is that the support of 2-itemsets is calculated and stored in a matrix. Then, an intersection operation is performed *only* if it surely results in a frequent itemset.

In the case of levelwise algorithms, with this method we can read the support from the matrix directly, and we do not need to use a trie for finding the subsets of each transacation in C_2 (where C_2 contains the potentially frequent 2-itemsets). Note that this optimization only concerns the support count of 2-itemsets in C_2. The support values of larger candidates are determined by a trie.

The Algorithm

The algorithm requires that the database be in horizontal format. In the case of vertical algorithms it means that first the database must be transformed (see Appendix D). If the database has n attributes, then an $(n-1) \times (n-1)$ upper triangular matrix is built, such as the one shown in Figure E.1. This matrix will contain the support values of 2-itemsets, thus its entries are initialized by 0. A row (transaction) of the database is decomposed into a list of 2-itemsets. For each element in this list, the value of its corresponding entry in the matrix is incremented by 1. This process is repeated for each row of the database.

$$
\begin{array}{c|c|c|c|c|}
 & \text{E} & \text{D} & \text{C} & \text{B} \\
\hline
\text{A} & 0 & 0 & 0 & 0 \\
\cline{2-5}
\text{B} & 0 & 0 & 0 & \multicolumn{1}{c}{} \\
\cline{2-4}
\text{C} & 0 & 0 & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} \\
\cline{2-3}
\text{D} & 0 & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} \\
\cline{2-2}
\end{array}
$$

Figure E.1: Initialized upper triangular matrix for counting the support of 2-itemsets.

Read row 1

$$\begin{array}{c|cccc} & E & D & C & B \\ \hline A & 1 & 1 & 0 & 1 \\ B & 1 & 1 & 0 & \\ C & 0 & 0 & & \\ D & 1 & & & \end{array}$$

Read row 2

$$\begin{array}{c|cccc} & E & D & C & B \\ \hline A & 1 & 1 & 1 & 1 \\ B & 1 & 1 & 0 & \\ C & 0 & 0 & & \\ D & 1 & & & \end{array}$$

Read row 3

$$\begin{array}{c|cccc} & E & D & C & B \\ \hline A & 2 & 1 & 2 & 2 \\ B & 2 & 1 & 1 & \\ C & 1 & 0 & & \\ D & 1 & & & \end{array}$$

Read row 4

$$\begin{array}{c|cccc} & E & D & C & B \\ \hline A & 2 & 1 & 2 & 2 \\ B & 3 & 1 & 2 & \\ C & 2 & 0 & & \\ D & 1 & & & \end{array}$$

Read row 5

$$\begin{array}{c|cccc} & E & D & C & B \\ \hline A & 3 & 1 & 3 & 3 \\ B & 4 & 1 & 3 & \\ C & 3 & 0 & & \\ D & 1 & & & \end{array}$$

Figure E.2: Support count of 2-itemsets of dataset D (Table 3.1) with an upper triangular matrix.

Example. The first row of dataset D (Table 3.1) includes the itemset $ABDE$. This itemset is decomposed into the following list of 2-itemsets: $\{AB, AD, AE, BD, BE, DE\}$. We read the first element of this list, AB, and increment its entry in the triangular matrix. We repeat this process for all other itemsets in the list and for all other rows of the database. Figure E.2 shows the process step by step.

Appendix F

Comparison of Generators Representation and Frequent Closed Itemset Representation

Here we compare the sizes of the generators representation and the frequent closed itemset representation of frequent itemsets. Tables F.1 and F.2 show the database name, the used minimum support value, the number of frequent generators and minimal rare generators, the size of the generators representation (which is the sum of the cardinality of the previous two sets), the size of the FCI representation (i.e. the number of FCIs), and the ratio of the size of the FCI representation to the size of the generators representation expressed in percentage. As we can see, the FCI representation is *always* much more condensed than the generators representation, especially in the case of sparse datasets.

dataset	min_supp	|FG|	|MRG|	**size of generators repr. (|FG|+|MRG|)**	**size of FCI repr.**	$\frac{\lvert\text{FCI repr.}\rvert}{\lvert\text{gen. repr.}\rvert}$
T20I6D100K	10%	7	907	914	7	0.77%
	5%	99	5,645	5,744	99	1.72%
	2%	378	68,411	68,789	378	0.55%
	1%	1,534	169,819	171,353	1,534	0.90%
	0.75%	4,710	211,578	216,288	4,710	2.18%
	0.5%	26,305	268,915	295,220	26,208	8.88%
	0.25%	149,447	537,765	687,212	149,217	21.71%
T25I10D10K	10%	20	1,099	1,119	20	1.79%
	5%	142	10,798	10,940	142	1.30%
	2%	533	106,931	107,464	533	0.50%
	1%	2,728	223,262	225,990	2,676	1.18%
	0.75%	8,164	279,363	287,527	7,841	2.73%
	0.5%	57,857	413,827	471,684	52,033	11.03%
	0.25%	193,104	1,190,931	1,384,035	150,601	10.88%

Table F.1: Comparison of the generators representation and the FCI representation of frequent itemsets.

dataset	min_supp	|FG|	|MRG|	**size of generators repr. (|FG|+|MRG|)**	**size of FCI repr.**	$\frac{\|\text{FCI repr.}\|}{\|\text{gen. repr.}\|}$
C20D10K	10%	9,331	901	10,232	8,777	85.78%
	5%	23,051	2,002	25,053	21,213	84.67%
	2%	57,659	7,735	65,394	50,729	77.57%
	1%	102,315	18,666	120,981	85,608	70.76%
	0.75%	127,745	24,641	152,386	103,892	68.18%
	0.5%	170,260	37,816	208,076	132,952	63.90%
	0.25%	267,248	69,104	336,352	193,448	57.51%
C73D10K	95%	121	1,622	1,743	93	5.34%
	90%	1,368	1,701	3,069	942	30.69%
	85%	3,513	1,652	5,165	2,359	45.67%
	80%	6,280	1,802	8,082	4,262	52.73%
	75%	13,917	1,939	15,856	9,367	59.08%
	70%	29,007	2,727	31,734	19,501	61.45%
	65%	71,874	3,675	75,549	47,491	62.86%
MUSHROOMS	50%	53	147	200	45	22.50%
	40%	153	254	407	124	30.47%
	30%	544	409	953	425	44.60%
	20%	1,704	1,004	2,708	1,169	43.17%
	10%	7,585	3,077	10,662	4,850	45.49%
	5%	21,391	8,806	30,197	12,789	42.35%
	1%	105,520	41,557	147,077	52,708	35.84%

Table F.2: Comparison of the generators representation and the FCI representation of frequent itemsets.

Appendix G

The Coron Toolkit

G.1 Coron-base

Coron-base is the most important module of the CORON platform. This module is responsible for the extraction of different itemsets, providing input to the other modules of the platform. With Coron-base one can extract the following itemsets:

- frequent itemsets (FIs)
- frequent closed itemsets (FCIs)
- frequent generators (FGs)
- maximal frequent itemsets (MFIs)
- minimal rare itemsets (MRIs)
- rare itemsets (RIs)
- minimal zero generators (MZGs)
- minimal rare generators (MRGs; equivalent to MRIs by Proposition 5.2)

Coron-base has a command line and a graphical user interface as well.

G.1.1 Command-line Interface

Usage: `coron [switches] <database> <min_supp> [-alg:<alg>]`

There are two compulsory parameters:

1. the database file (in .basenum, .bool or .rcf format), and
2. the minimum support (in absolute or relative value).

Throughout this guide we will work with the dataset shown in Table G.1.[66] In the examples we assume that this dataset is stored in a file called `laszlo.rcf` in .rcf format. The supported file formats are shown in Table G.2. Line i of a .basenum file contains items that are included in object i. The .bool file is a binary matrix representation of the binary database. The .rcf file is very similar to the .bool file format but it has the advantage that *names* can be assigned to objects and to attributes.

[66]This database is the same as dataset D in Table 3.1.

	a (1)	b (2)	c (3)	d (4)	e (5)
o_1	x	x		x	x
o_2	x		x		
o_3	x	x	x		x
o_4		x	x		x
o_5	x	x	x		x

Table G.1: A toy dataset for the examples.

(1) .basenum	**(2)** .bool	**(3)** .rcf
`1 2 4 5`	`1 1 0 1 1`	`[Relational Context]`
`1 3`	`1 0 1 0 0`	`Default Name`
`1 2 3 5`	`1 1 1 0 1`	`[Binary Relation]`
`2 3 5`	`0 1 1 0 1`	`Name_of_dataset`
`1 2 3 5`	`1 1 1 0 1`	`o1 | o2 | o3 | o4 | o5 |`
		`a | b | c | d | e |`
		`1 1 0 1 1`
		`1 0 1 0 0`
		`1 1 1 0 1`
		`0 1 1 0 1`
		`1 1 1 0 1`
		`[END Relational Context]`

Table G.2: Our example dataset (Table G.1) in different file formats.

The minimum support can be given in either absolute or relative value, e.g. 2 or 40%.

There are two kinds of switches:

1. `-option` (example: `-names`)
2. `-key:value` (example: `-alg:apriori`)

The algorithm to be used can be specified with the `-alg:<alg>` switch. The available algorithms are described below.

Example:

```
./start test/laszlo.rcf 2 -names -alg:apriori
```

Result:

```
# Database file name:                  test/laszlo.rcf
# Database file size:                  208 bytes
# Number of lines:                     5
# Largest attribute:                   5
# Number of attributes:                5
# Number of attributes in average: 3.4
# min_supp:                            2, i.e. 40%
# Chosen algorithm:                    Apriori

{a} (4)
{b} (4)
...
# FIs: 15
```

At the beginning and at the end there are some statistics about the dataset and the number of found itemsets.

If we only want to analyze the input dataset *without* calculating the itemsets, use the `-stat` option:

```
./start test/laszlo.rcf -stat
```

In this case the program terminates after showing the database statistics.

The `-names` option is highly recommended. It works only for .rcf files. With this option, attribute numbers can be replaced by their names. The example above without `-names` would look like this:

```
./start test/laszlo.rcf 40% -alg:apriori
```

Result:

```
{1} (4)
{2} (4)
...
```

This means: the first attribute has support 4, the second attribute has also support 4, etc.

Other options:

`--help`	help information
`--version`, `-V`	version information
`--update`	check for a new version

Verbosity options:

`-v:m`	memory usage
`-v:f`	function information (which function is called)
`-v:t`	time information (runtime)

These options can be combined with `-vc`:

`-vc:mf`	equivalent to `-v:m␣-v:f`

Verbosity options display some additional information while the program is running. These kind of feedbacks are always redirected to the standard error, and these lines start with a '>' sign. Because of the redirection to stderr, this information does not mix with normal result.

Statistical information is sent to the standard output, and these lines always start with a '#' sign. This way, these lines can be easily filtered.

G.1.2 Available Algorithms

1. *Apriori* (-alg:apriori)

This is the basic algorithm for finding FIs. It is efficient for sparse datasets only. Since it is a levelwise algorithm, it produces itemsets in ascending order by length. Sample output:

```
{a} (4)
```

This means: {a} is a frequent itemset with support 4.

2. *Apriori-Close* (-alg:aprioriclose, -alg:ac)

A simple extension of *Apriori*. It also marks FCIs. It is almost as efficient as *Apriori*, i.e. the derivation of FCIs does not induce serious additional computation time.

```
{a} (4) +
{b} (4)
```

In each algorithm, the '+' sign means that the itemset is closed. That is: {a} is a frequent closed itemset with support 4; {b} is frequent (but not closed) with support 4.

3. *Apriori-Rare* (-alg:apriorirare)

This algorithm is based on *Apriori*, thus its efficiency is like *Apriori*'s. While enumerating FIs, it filters minimal rare itemsets (MRIs). An MRI is a rare itemset, and all its proper subsets are frequent.

There are two kinds of MRIs: MRIs with support 0 (zero itemsets), and MRIs with support higher than 0 (non-zero itemsets). It must be specified which group to extract:

• `-all`	extract zero itemsets too, or
• `-nonzero`	extract non-zero itemsets only

```
./start test/laszlo.rcf 3 -names -alg:apriorirare -nonzero
```

```
{d} (1)
```

This means: {d} is an MRI with support 1.

4. *Arima* (-alg:arima)

Arima (A Rare Itemset Miner Algorithm) finds rare itemsets (RIs) in the following way: it takes MRIs generated by *Apriori-Rare*. Then, using non-zero MRIs it finds their proper supersets. In the result zero-itemsets are avoided because of their large number.
Minimal zero generators (MZGs) are used to reduce the search space (lots of zero itemsets are pruned thanks to this technique).

```
./start test/laszlo.rcf 3 -names -alg:arima
```

```
{a, d, e} (1)
```

This means: {a, d, e} is a rare itemset with support 1.

5. *BtB* (-alg:btb)

BtB (Breaking the Barrier) finds rare (non-zero) equivalence classes. The result is like *Zart*'s with the difference that these equivalence classes are rare. From this result we can generate MRG association rules.

Idea: get non-zero MRIs and find their closures. Result is produced in the following form: closure + list of generators.

```
./start test/laszlo.rcf 3 -names -alg:btb
```

```
{a, b, c, e} (2) +; [{a, b, c}, {a, c, e}]
```

This means: {a,b,c,e} is a rare closed itemset with support 2. It has two generators: {a,b,c} and {a,c,e}. By the property of equivalence classes, their support is also 2.

6. *Charm* (-alg:charm)

A *very* efficient algorithm for finding FCIs. Not a breadth-first, but a depth-first algorithm. It needs to keep all FCIs in the main memory in order to decide if a newly found itemset is closed or not. When the algorithm finishes the enumeration of itemsets, it has all FCIs in the main memory in a hash structure. Normally the itemsets are not ordered by length, but if needed, the hash structure can be easily traversed in such a way that the result be ordered.

```
{a, b, e} (3) +
```

This means: {a,b,e} is an FCI with support 3.

7. *Charm-MFI* (-alg:charmmfi, -alg:mfi)

This algorithm is based on *Charm* (thus its efficiency is almost the same). Filters MFIs among FCIs. The two options (`-alg:charmmfi` and `-alg:mfi`) result in different outputs:

```
./start test/laszlo.rcf 3 -names -alg:charmmfi
```

```
{a, c} (3) [mfi: +] +
{b, e} (4) [mfi: -] +
```

It prints *all* FCIs and marks which FCIs are MFIs. That is: {a,c} is an FCI with support 3, and it is an MFI too. On the contrary, {b,e} is not an MFI.

```
./start test/laszlo.rcf 3 -names -alg:mfi
```

```
{a, b, e} (3) [mfi: +] +
```

It *only* prints MFIs. Those FCIs that are not MFIs are not displayed. That is: {a,b,e} is an MFI. Note that all MFIs are FCIs too (just like all FCIs are FIs).

8. *Close* (-alg:close)

A levelwise algorithm that finds FCIs. It is more efficient on dense datasets. Its authors do not mention that during the levelwise search the algorithm can find the same FCIs several times. Thus, if the result is written immediately to a file, then it will contain duplicates. To avoid it, FCIs must be kept in main memory in order to prune duplicates. It results in a large memory consumption.

```
{b, c, e} (3) +
```

This means: {b,c,e} is an FCI with support 3.

9. *Eclat* (-alg:eclat)

Actually, *Charm* is a modified version of *Eclat*. *Eclat* is not a breadth-first, but a depth-first algorithm. It finds all FIs in a very efficient way. In the output, frequent itemsets are *not* ordered by their length.

```
{a, b, e} (3)
```

This means: {a,b,e} is an FI with support 3.

10. *Eclat-Z* (-alg:eclatz)

Eclat-Z (Eclat-Zart) is a combination of *Eclat* with the idea introduced in *Zart*. *Eclat* finds efficiently all FIs. These itemsets are stored in a file. When done, the itemsets in the file are processed in ascending order by length (remember, *Eclat* produces FIs in an unordered way). This can be done efficiently with a special file indexing technique. During this levelwise post-process, FGs and FCIs are marked, and generators are associated to their closures.

```
{b, e} (4) +; [{b}, {e}]
```

This means: {b,e} is an FCI with support 4. It has two generators, {b} and {e} (with the same support).

11. *Pascal* (-alg:pascal)

An efficient levelwise algorithm for finding all FIs. It also marks which itemsets are key generators. Among levelwise FI-miner algorithms, it may be the most efficient.

```
{a} (4) [key: +]
{b, e} (4) [key: -]
```

This means: {a} is an FI with support 4. It is an FG too. The itemset {b,e} is an FI with support 4, but it is not a key generator.

12. *Pascal+* (-alg:pascalplus)

Pascal$^+$ is a simple extension of *Pascal*. In addition to *Pascal*, it also tells which itemsets are closed.

```
{a} (4) [key: +] +
{b, c} (3) [key: +]
{b, e} (4) [key: -] +
```

This means: {a} is an FI, an FG and an FCI. The itemset {b,c} is an FI, an FG, but not an FCI. The itemset {b,e} is an FI, not an FG, but an FCI.

13. *Titanic* (-alg:titanic)

Titanic, similary to *Pascal*, uses pattern counting inference to minimize the number of database passes. *Titanic* has an interesting "trick": it can derive the closure of frequent generators *without* accessing the database. Unfortunately, in order to do so, it must keep in main memory a large set of itemsets, and it must also perform lots of intersection operations. This results in an algorithm with a huge memory consumption.

```
{b} (4) [key: +; closure: {b, e}]
```

This means: {a} is an FG with support 4. Its closure is {b, e} with the same support value.

14. *Zart* (-alg:zart)

This algorithm is based on *Pascal*. In addition to *Pascal*, it marks FCIs, and associates generators to their closures. It is almost as efficient as *Pascal*. The output of the algorithm is ideal for generating minimal non-redundant association rules.

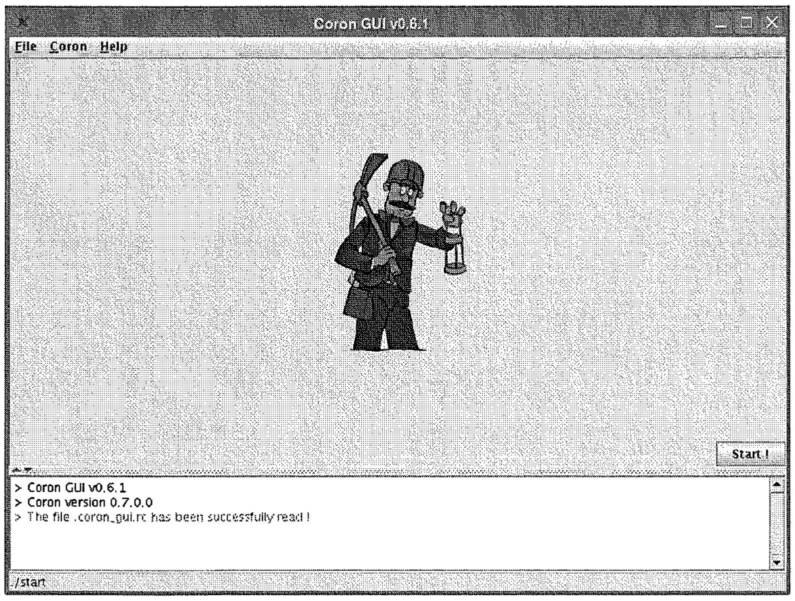
```
{b, e} (4) +; [{b}, {e}]
```

This means: {b,e} is an FCI with support 4. It has two generators, {b} and {e} (with the same support).

G.1.3 Graphical User Interface

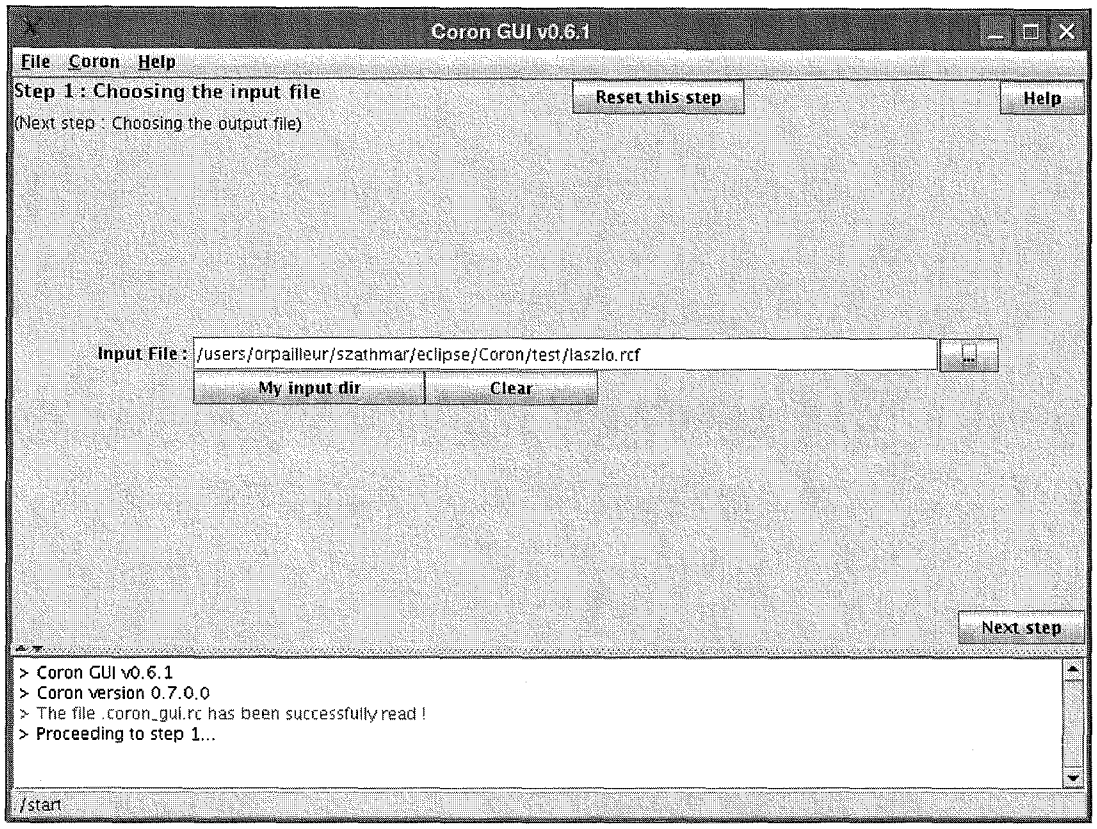
Coron-base is equipped with a graphical frontend too. Figures G.1 – G.7 show the different steps of the interface. At step 1 the user chooses the input file. At step 2 he chooses an output file because the result is saved in a file in all cases. It is possible to use a temporary file. After defining the minimum support (step 3) and choosing the mining algorithm (step 4), the software summarizes the user's choice at step 5. The user can go back at each step to modify his choice. After pressing the "Start calculation!" button, the result is saved in a file, which can be visualized at the end (step 6).

The graphical interface uses a configuration file called `.coron_gui.rc`, which is placed in the `HOME/.coron` directory. When the GUI is launched for the very first time, this file is created automatically with the default values. This file can be edited by the user to customize the software.



```
> Coron GUI v0.6.1
> Coron version 0.7.0.0
> The file .coron_gui.rc has been successfully read !
```

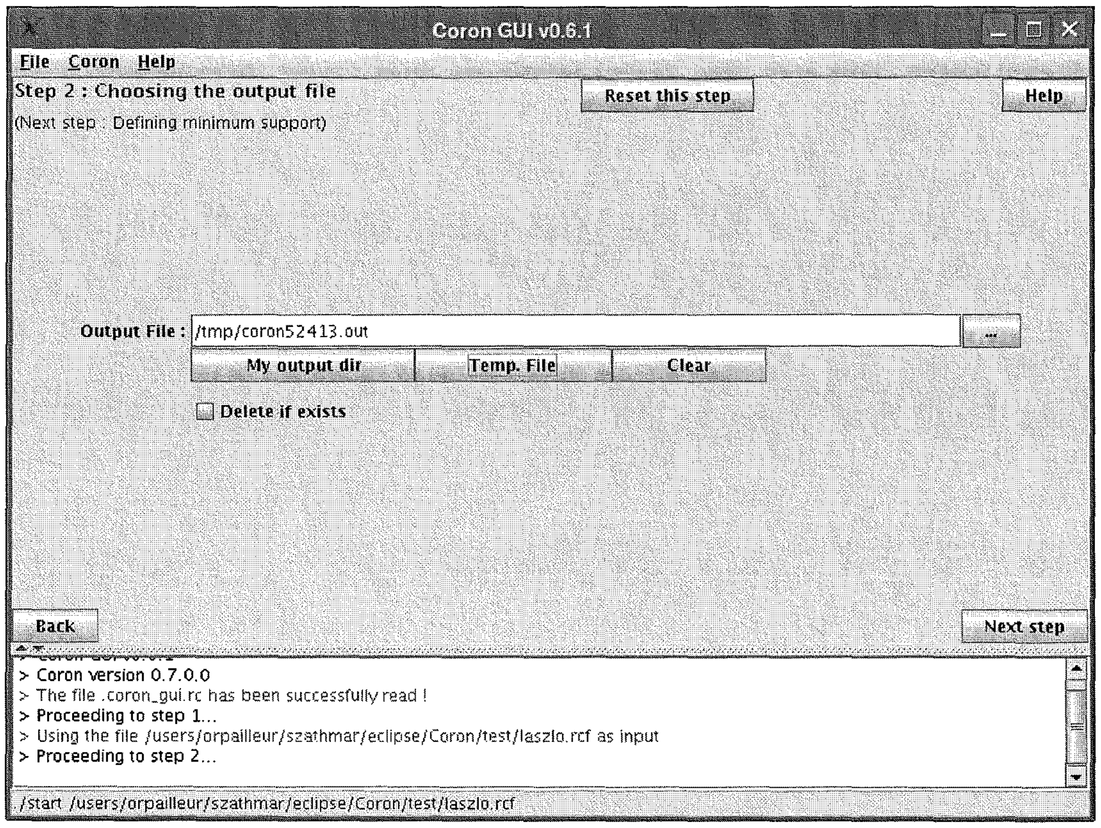
Figure G.1: Step 0 – Welcome screen.


Coron GUI v0.6.1
File Coron Help
Step 1 : Choosing the input file
(Next step : Choosing the output file)
Reset this step
Help
Input File : /users/orpailleur/szathmar/eclipse/Coron/test/laszlo.rcf
My input dir
Clear
Next step

```
> Coron GUI v0.6.1
> Coron version 0.7.0.0
> The file .coron_gui.rc has been successfully read !
> Proceeding to step 1...
```

/start

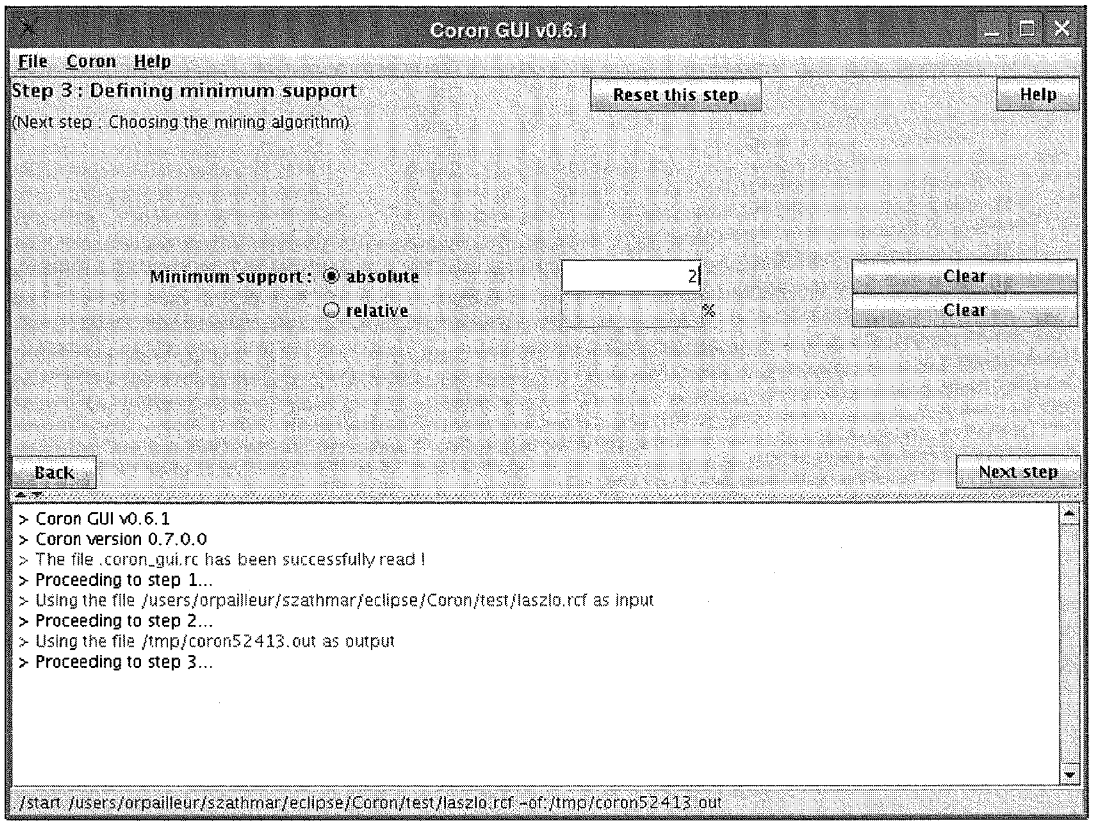
Figure G.2: Step 1 – Choosing the input file.



```
> Coron version 0.7.0.0
> The file .coron_gui.rc has been successfully read !
> Proceeding to step 1...
> Using the file /users/orpailleur/szathmar/eclipse/Coron/test/laszlo.rcf as input
> Proceeding to step 2...
```

/start /users/orpailleur/szathmar/eclipse/Coron/test/laszlo.rcf

Figure G.3: Step 2 – Choosing the output file.



Step 3 : Defining minimum support

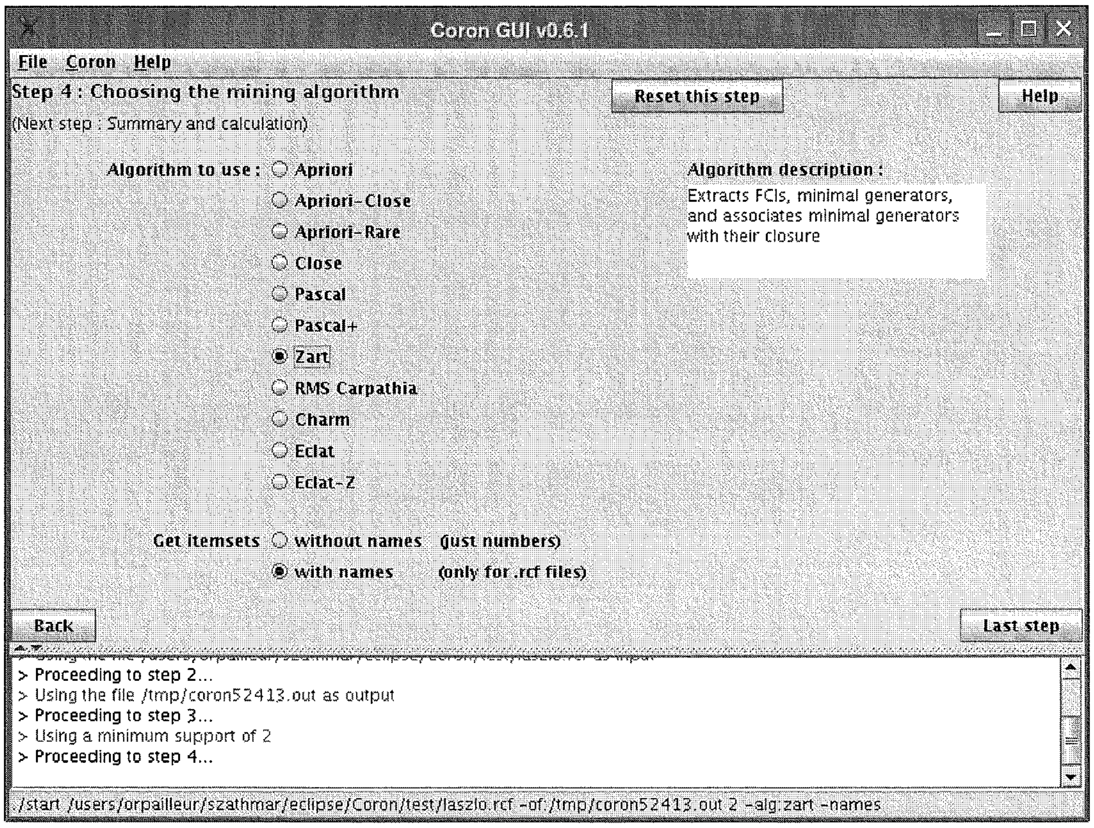
(Next step : Choosing the mining algorithm)

Minimum support : [✓ absolute] 2 relative %

```
> Coron GUI v0.6.1
> Coron version 0.7.0.0
> The file .coron_gui.rc has been successfully read !
> Proceeding to step 1...
> Using the file /users/orpailleur/szathmar/eclipse/Coron/test/laszlo.rcf as input
> Proceeding to step 2...
> Using the file /tmp/coron52413.out as output
> Proceeding to step 3...
```

```
./start /users/orpailleur/szathmar/eclipse/Coron/test/laszlo.rcf -of:/tmp/coron52413.out
```

Figure G.4: Step 3 – Defining the minimum support.



Step 4 : Choosing the mining algorithm

(Next step : Summary and calculation)

Algorithm to use : Apriori, Apriori-Close, Apriori-Rare, Close, Pascal, Pascal+, [✓ Zart], RMS Carpathia, Charm, Eclat, Eclat-Z

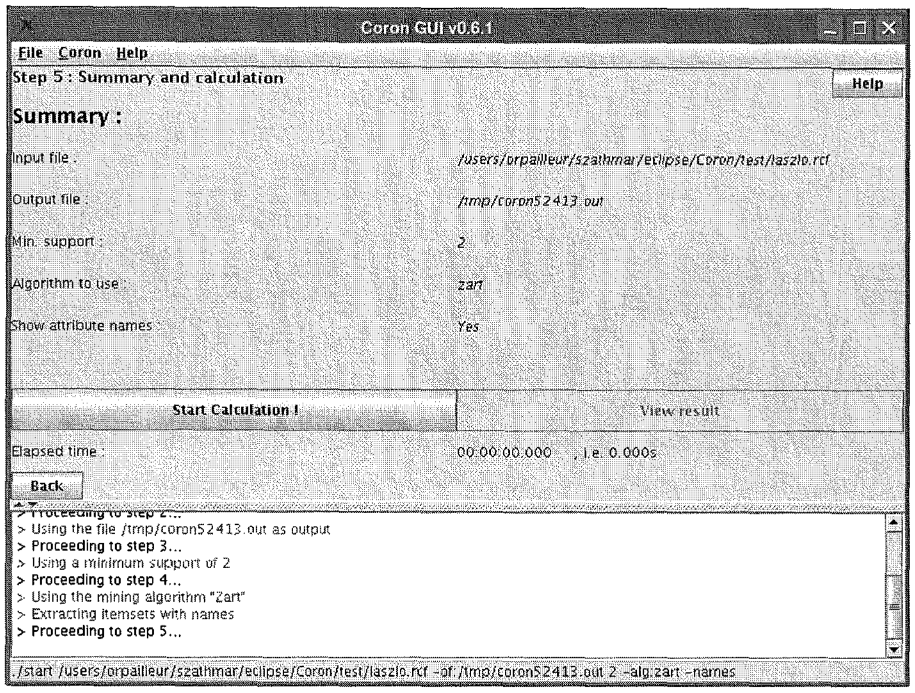
Algorithm description : Extracts FCIs, minimal generators, and associates minimal generators with their closure

Get itemsets without names (just numbers) [✓ with names (only for .rcf files)]

```
> Proceeding to step 2...
> Using the file /tmp/coron52413.out as output
> Proceeding to step 3...
> Using a minimum support of 2
> Proceeding to step 4...
```

```
./start /users/orpailleur/szathmar/eclipse/Coron/test/laszlo.rcf -of:/tmp/coron52413.out 2 -alg:zart -names
```

Figure G.5: Step 4 – Choosing the mining algorithm.



Step 5 : Summary and calculation

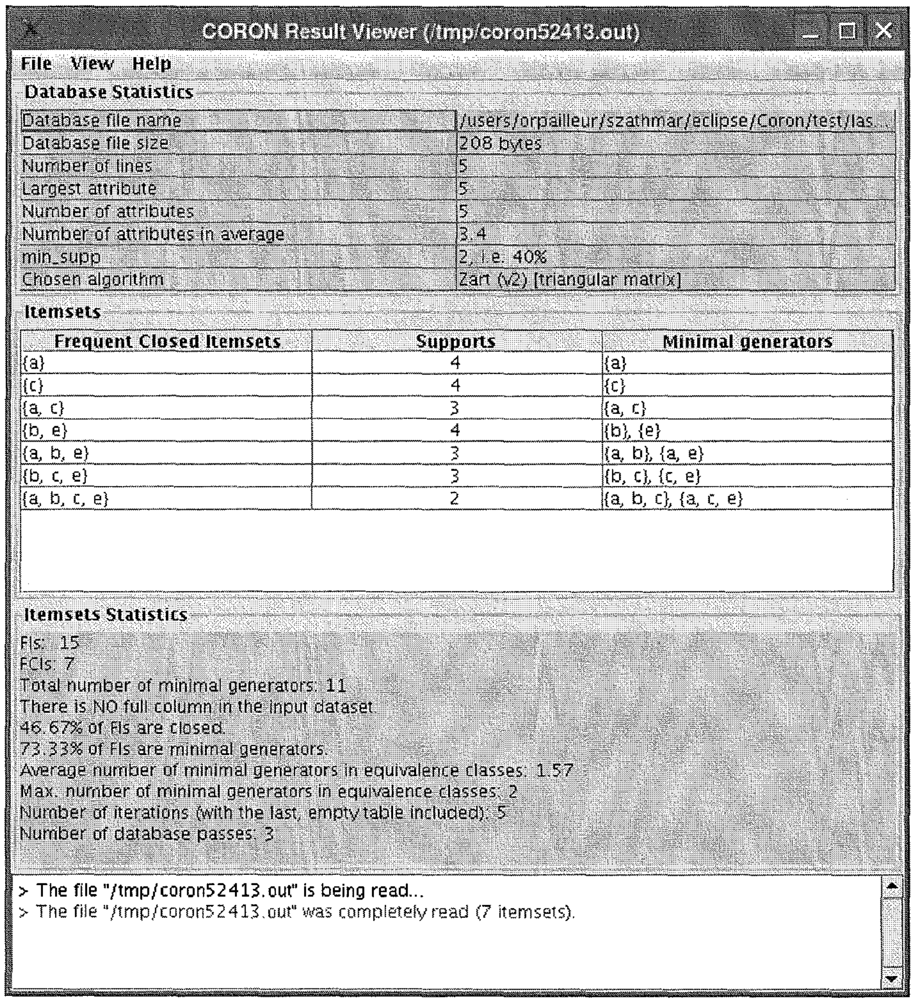
Summary :

Input file :	/users/orpailleur/szathmar/eclipse/Coron/test/laszlo.rcf
Output file :	/tmp/coron52413.out
Min. support :	2
Algorithm to use :	zart
Show attribute names :	Yes
Elapsed time :	00:00:00.000 , i.e. 0.000s

```
> Using the file /tmp/coron52413.out as output
> Proceeding to step 3...
> Using a minimum support of 2
> Proceeding to step 4...
> Using the mining algorithm "Zart"
> Extracting itemsets with names
> Proceeding to step 5...
```

```
./start /users/orpailleur/szathmar/eclipse/Coron/test/laszlo.rcf -of:/tmp/coron52413.out 2 -alg:zart -names
```

Figure G.6: Step 5 – Statistics of the user's choice.



Database Statistics

Database file name	/users/orpailleur/szathmar/eclipse/Coron/test/las...
Database file size	208 bytes
Number of lines	5
Largest attribute	5
Number of attributes	5
Number of attributes in average	3.4
min_supp	2, i.e. 40%
Chosen algorithm	Zart (v2) [triangular matrix]

Itemsets

Frequent Closed Itemsets	Supports	Minimal generators
{a}	4	{a}
{c}	4	{c}
{a, c}	3	{a, c}
{b, e}	4	{b}, {e}
{a, b, e}	3	{a, b}, {a, e}
{b, c, e}	3	{b, c}, {c, e}
{a, b, c, e}	2	{a, b, c}, {a, c, e}

Itemsets Statistics

```
FIs: 15
FCIs: 7
Total number of minimal generators: 11
There is NO full column in the input dataset
46.67% of FIs are closed.
73.33% of FIs are minimal generators.
Average number of minimal generators in equivalence classes: 1.57
Max. number of minimal generators in equivalence classes: 2
Number of iterations (with the last, empty table included): 5
Number of database passes: 3
```

```
> The file "/tmp/coron52413.out" is being read...
> The file "/tmp/coron52413.out" was completely read (7 itemsets).
```

Figure G.7: Step 6 – Displaying the result.

G.2 AssRuleX

AssRuleX (Association Rule eXtractor) is the second most important module of the Coron platform. This module is responsible for the extraction of different sets of association rules. With AssRuleX one can extract the following association rules:

1. all valid association rules	`-rule:all`
2. closed association rules	`-rule:closed`
3. all informative association rules	`-rule:all_inf`
4. reduced informative association rules	`-rule:inf`
5. Generic Basis (GB)	`-rule:GB`
6. (all) Informative Basis (IB)	`-rule:all_IB`
7. reduced Informative Basis (IB)	`-rule:IB`
8. rare informative association rules	`-rule:rare`

Note that under "all informative association rules" we mean the minimal non-redundant association rules ($\mathcal{MNR}$), under "reduced informative association rules" we mean the transitive reduction of $\mathcal{MNR}$ (i.e. $\mathcal{RMNR}$), and under "rare informative association rules" we mean the exact MRG rules.[67]

G.2.1 Command-line Interface

Usage:

```
assrulex [switches] <database> <min_supp> <min_conf> -alg:<alg> -rule:<rule>
```

There are five compulsory parameters:

1. database file (in .basenum, .bool or .rcf format)
2. minimum support
3. minimum confidence
4. name of the algorithm to be used
5. rule set that we want to extract with the previously specified algorithm.

The minimum support can be given in either absolute or relative value, e.g. 2 or 40%.

The minimum confidence can be given as a real value (between 0 and 1.0, e.g. 0.5), or as a percentage (between 0% and 100%, e.g. 50%).

There are two kinds of switches:

1. `-option` (example: `-names`)
2. `-key:value` (example: `-alg:apriori`)

Other options:

[67]The reason of this "confusion" is that Bastide *et al.* called their rules as "informative rules" in [BTP$^+$02]. However, Kryszkiewicz in [Kry02] uses the concept "informative" in a different sense, and she calls the same set of rules as "minimal non-redundant rules". In the thesis we use this latter terminology, but since we started to develop AssRuleX using the old terminology, we decided not to change it to allow backward compatibility.

`--help`	help information
`--version, -V`	version information
`--update`	check for a new version

Verbosity options:

`-v:m`	memory usage
`-v:f`	function information (which function is called)
`-v:t`	time information (runtime)

These options can be combined with `-vc`:

`-vc:mf`	equivalent to `-v:m␣-v:f`

The following algorithm/association rules combinations can be used:

```
Apriori:
   1) all association rules                        -rule:all
Close:
   1) closed association rules                     -rule:closed
Pascal:
   1) all association rules                        -rule:all
Pascal+:
   1) all association rules                        -rule:all
   2) closed association rules                     -rule:closed
Charm:
   1) closed association rules                     -rule:closed
Zart:
   1) all association rules                        -rule:all
   2) closed association rules                     -rule:closed
   3) all informative association rules            -rule:all_inf
   4) reduced informative association rules        -rule:inf
   5) Generic Basis (GB)                           -rule:GB
   6) (all) Informative Basis (IB)                 -rule:all_IB
   7) reduced Informative Basis (IB)               -rule:IB
Eclat-Z:
   1) all association rules                        -rule:all
   2) closed association rules                     -rule:closed
   3) all informative association rules            -rule:all_inf
   4) reduced informative association rules        -rule:inf
   5) Generic Basis (GB)                           -rule:GB
   6) (all) Informative Basis (IB)                 -rule:all_IB
   7) reduced Informative Basis (IB)               -rule:IB
BtB:
   1) rare association rules                       -rule:rare
```

Example:

```
./start test/laszlo.rcf 4 50% -names -alg:zart -rule:inf
```

Result:

```
# Database file name:              test/laszlo.rcf
# Database file size:              208 bytes
# Number of lines:                 5
# Largest attribute:               5
# Number of attributes:            5
# Number of attributes in average: 3.4
# min_supp:                        4, i.e. 80%
# min_conf:                        50%
# Chosen algorithm:                Zart
# Rules to extract:                reduced informative association rules

{b} => {e} (supp=4 [80.00%]; conf=1.000 [100.00%]; suppL=4 [80.00%];
suppR=4 [80.00%]; class=FF) +

{e} => {b} (supp=4 [80.00%]; conf=1.000 [100.00%]; suppL=4 [80.00%];
suppR=4 [80.00%]; class=FF) +

# Number of found rules: 2
# Number of FF rules: 2
```

At the beginning and at the end there are some statistics about the dataset and the number of found rules.

If we only want to analyze the input dataset *without* calculating the itemsets, use the `-stat` option:

```
./start test/laszlo.rcf 4 50% -names -alg:zart -rule:inf -stat
```

In this case the program terminates after showing the database statistics.

The `-names` option is highly recommended. It works only for .rcf files. With this option, attribute numbers can be replaced with their names.

Let us see what a rule looks like:

```
{b} => {e} (supp=4 [80.00%]; conf=1.000 [100.00%]; suppL=4 [80.00%];
suppR=4 [80.00%]; class=FF) +
```

This means: the antecedent is {b}, the consequent is {e}. The support of the rule is 4, which is equivalent to 80% in this dataset (Table G.1). Confidence: 100%. Support of the left part of the rule: 4; support of the right part of the rule: 4. The rule is in the FF class, i.e. both sides of the rule are frequent (frequent itemset implies frequent itemset). The rule is closed.

There are some other quality measures available for the rules. They can be visualized with the `-full` or `-measures` switch.

```
{b} => {e} (supp=4 [80.00%]; conf=1.000 [100.00%]; suppL=4 [80.00%]; suppR=4
[80.00%]; lift=1.250; conv=NOT_DEF; dep=0.200; nov=0.160; sat=1.000; class=FF) +
```

This means:

1. left part of the rule ({b})
2. right part of the rule ({e})
3. support of the rule (4, i.e. 80%)
4. confidence of the rule (1.0, i.e. 100%)
5. support of the left part of the rule (4, i.e. 80%)
6. support of the right part of the rule (4, i.e. 80%)
7. lift (1.250)
8. conviction (not defined in the case of exact association rules)
9. dependency (0.200)
10. novelty (0.160)
11. satisfaction (1.000)
12. classification of the rule (type FF, i.e. frequent itemset implies frequent itemset)
13. is it a closed rule? (in the example the rule is closed)

Notes: in some cases a statistical measure cannot be calculated for a rule. In this case "NOT_DEF" is displayed. The '+' at the end means that the rule is closed, i.e. the union of the antecedent and consequent is a closed itemset.

With the `-examples` switch one can visualize the positive and negative examples of each rule. Positive example: objects that contain left *and* right sides of the rule. Negative example: objects that contain the left, but *not* the right side of the rule.

Example:

```
./start test/laszlo.rcf 2 50% -names -alg:zart -rule:inf -examples
```

Sample output:

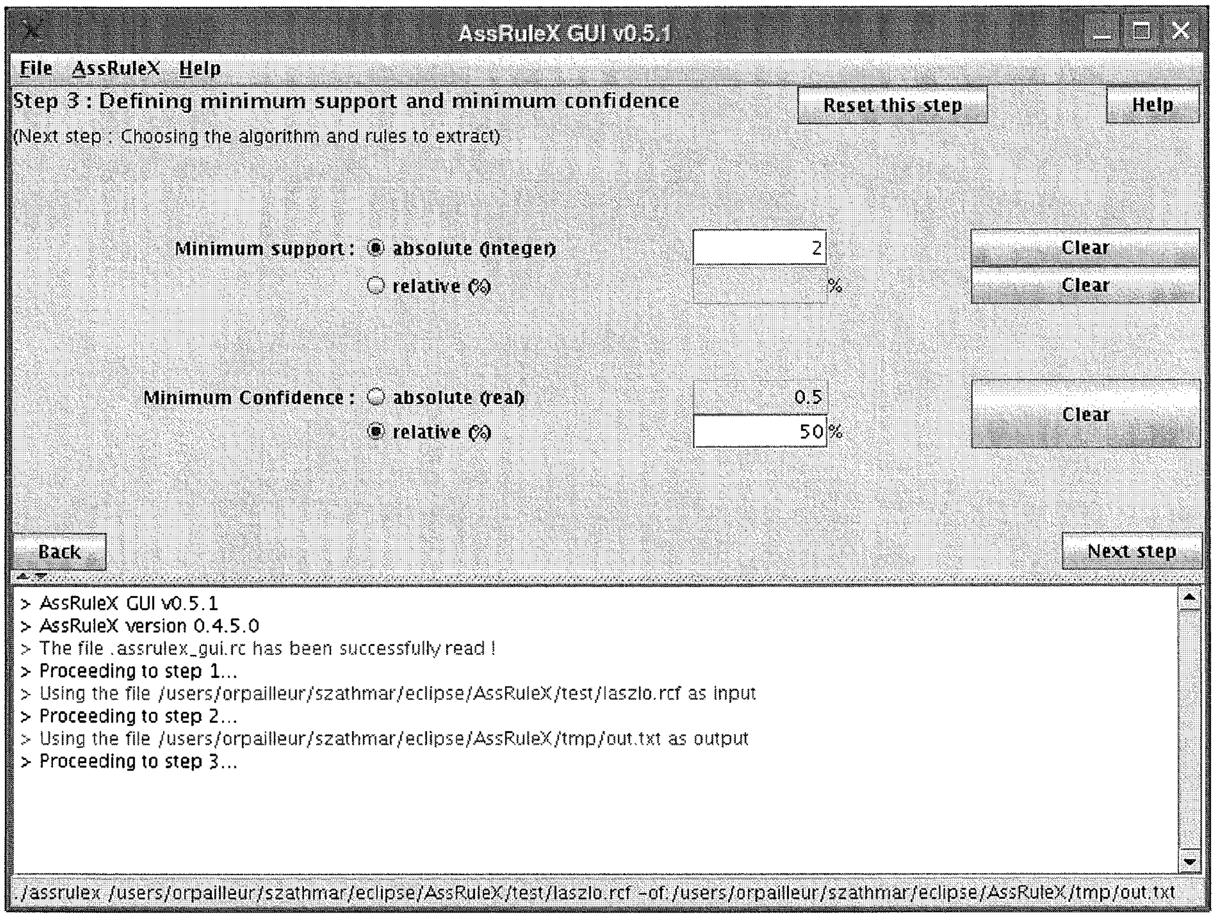
```
{a} => {b, e} (supp=3 [60.00%]; conf=0.750 [75.00%]; suppL=4 [80.00%];
suppR=4 [80.00%]; class=FF) +
Positive examples (objects that contain left AND right sides of the rule):
[o1, o3, o5]
Negative examples (objects that contain left, BUT NOT the right side of the rule):
[o2]
```

G.2.2 Graphical User Interface

AssRuleX also has a graphical frontend. The graphical interface is very similar to Coron-base's, thus the following figures only show those screens that are different.

At step 1 the user chooses the input file. At step 2 we need to choose an output file because the result is saved in a file in all cases. It is possible to use a temporary file. After defining the minimum support and minimum confidence (step 3, Figure G.8), we must choose the mining algorithm and the type of rules to be extracted (step 4, Figure G.9). The software summarizes the user's choice at step 5. We can go back at each step to modify our choice. After pressing the "Start calculation!" button, the result is saved in a file, which can be visualized at the end (step 6, Figure G.10).

The graphical interface uses a configuration file called `.assrulex_gui.rc`, which is placed in the `HOME/.coron` directory. When the GUI is launched for the very first time, this file is created automatically with the default values. This file can be edited by the user to customize the software.



Step 3 : Defining minimum support and minimum confidence
(Next step : Choosing the algorithm and rules to extract)

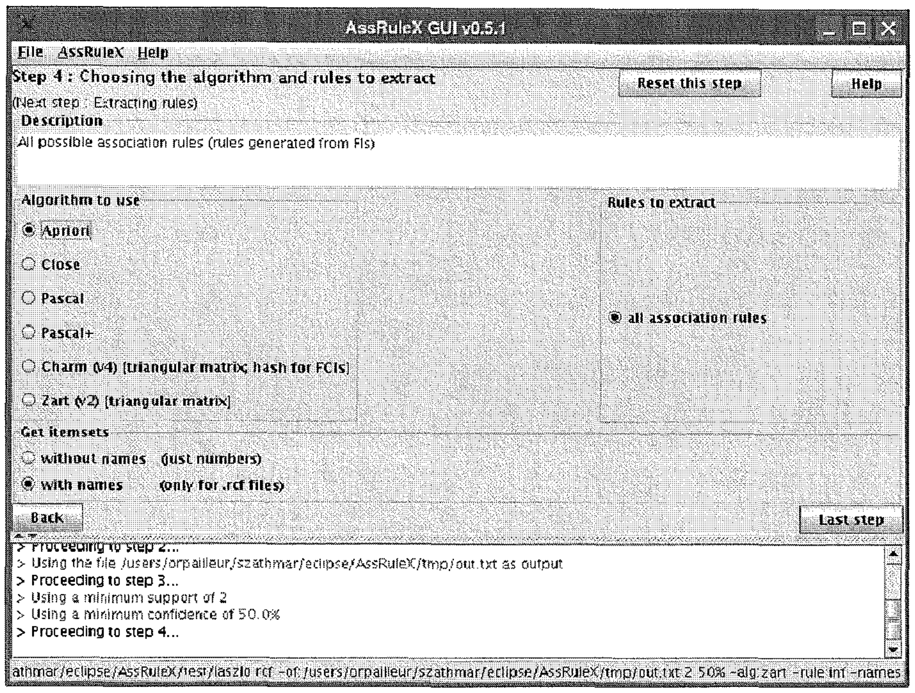
Minimum support : [✓ absolute (integer)] 2 relative (%) %

Minimum Confidence : absolute (real) 0.5 [✓ relative (%)] 50 %

```
> AssRuleX GUI v0.5.1
> AssRuleX version 0.4.5.0
> The file .assrulex_gui.rc has been successfully read !
> Proceeding to step 1...
> Using the file /users/orpailleur/szathmar/eclipse/AssRuleX/test/laszlo.rcf as input
> Proceeding to step 2...
> Using the file /users/orpailleur/szathmar/eclipse/AssRuleX/tmp/out.txt as output
> Proceeding to step 3...
```

```
./assrulex /users/orpailleur/szathmar/eclipse/AssRuleX/test/laszlo.rcf -of:/users/orpailleur/szathmar/eclipse/AssRuleX/tmp/out.txt
```

Figure G.8: Step 3 – Defining the minimum support and minimum confidence thresholds.



Step 4 : Choosing the algorithm and rules to extract
(Next step : Extracting rules)

Description
All possible association rules (rules generated from FIs)

Algorithm to use
- [✓ Apriori]
- Close
- Pascal
- Pascal+
- Charm (v4) [triangular matrix, hash for FCIs]
- Zart (v2) [triangular matrix]

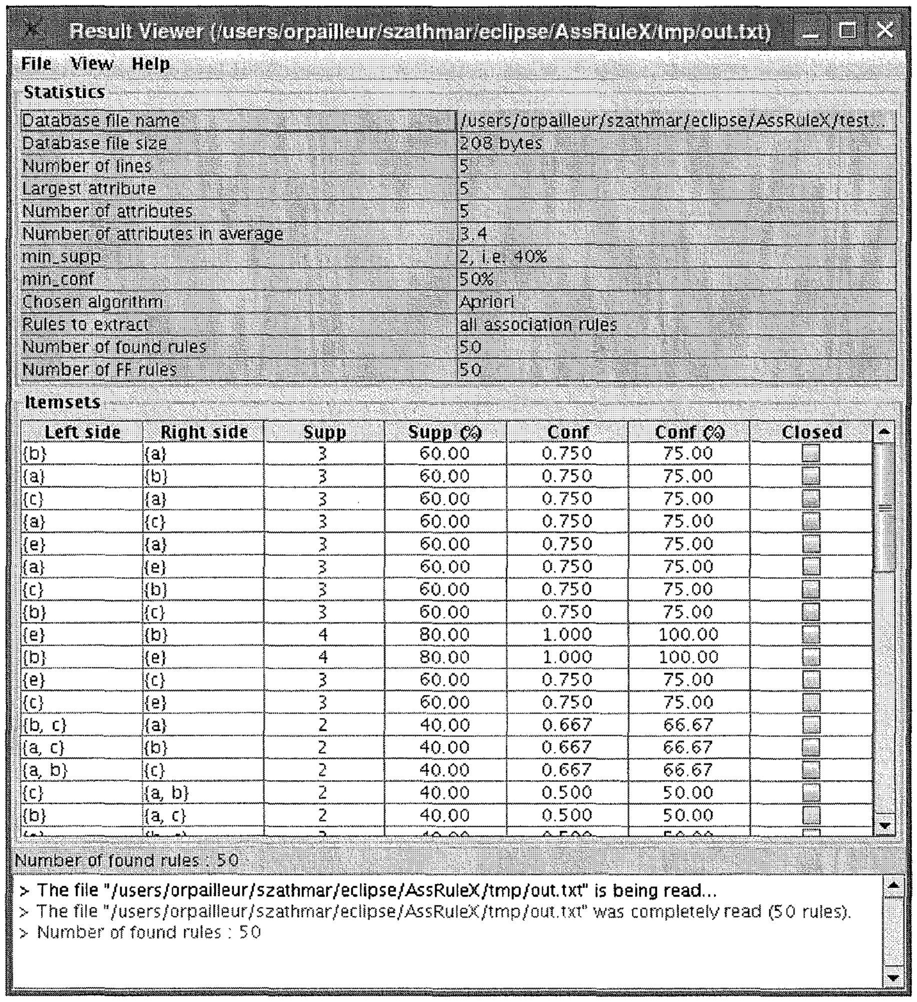
Rules to extract
- [✓ all association rules]

Get itemsets
- without names (just numbers)
- [✓ with names (only for .rcf files)]

```
> Proceeding to step 2...
> Using the file /users/orpailleur/szathmar/eclipse/AssRuleX/tmp/out.txt as output
> Proceeding to step 3...
> Using a minimum support of 2
> Using a minimum confidence of 50.0%
> Proceeding to step 4...
```

```
athmar/eclipse/AssRuleX/test/laszlo.rcf -of /users/orpailleur/szathmar/eclipse/AssRuleX/tmp/out.txt 2 50% -alg zart -rule inf -names
```

Figure G.9: Step 4 – Choosing the mining algorithm and rules to extract.


Result Viewer (/users/orpailleur/szathmar/eclipse/AssRuleX/tmp/out.txt)
File View Help

Statistics

Database file name	/users/orpailleur/szathmar/eclipse/AssRuleX/test...
Database file size	208 bytes
Number of lines	5
Largest attribute	5
Number of attributes	5
Number of attributes in average	3.4
min_supp	2, i.e. 40%
min_conf	50%
Chosen algorithm	Apriori
Rules to extract	all association rules
Number of found rules	50
Number of FF rules	50

Itemsets

Left side	Right side	Supp	Supp (%)	Conf	Conf (%)	Closed
{b}	{a}	3	60.00	0.750	75.00	☐
{a}	{b}	3	60.00	0.750	75.00	☐
{c}	{a}	3	60.00	0.750	75.00	☐
{a}	{c}	3	60.00	0.750	75.00	☐
{e}	{a}	3	60.00	0.750	75.00	☐
{a}	{e}	3	60.00	0.750	75.00	☐
{c}	{b}	3	60.00	0.750	75.00	☐
{b}	{c}	3	60.00	0.750	75.00	☐
{e}	{b}	4	80.00	1.000	100.00	☐
{b}	{e}	4	80.00	1.000	100.00	☐
{e}	{c}	3	60.00	0.750	75.00	☐
{c}	{e}	3	60.00	0.750	75.00	☐
{b, c}	{a}	2	40.00	0.667	66.67	☐
{a, c}	{b}	2	40.00	0.667	66.67	☐
{a, b}	{c}	2	40.00	0.667	66.67	☐
{c}	{a, b}	2	40.00	0.500	50.00	☐
{b}	{a, c}	2	40.00	0.500	50.00	☐

Number of found rules : 50

```
> The file "/users/orpailleur/szathmar/eclipse/AssRuleX/tmp/out.txt" is being read...
> The file "/users/orpailleur/szathmar/eclipse/AssRuleX/tmp/out.txt" was completely read (50 rules).
> Number of found rules : 50
```

Figure G.10: Step 6 – Displaying the result.

G.3 Pre-processing: Data Preparation and Filtering

G.3.1 Filter-DB

Filter-DB (Filter-DataBase) is a utility for pre-processing datasets for further work with Coron-base and AssRuleX. It can filter horizontally (i.e. the rows) and/or vertically (i.e. the attributes) of the input dataset.

Usage: `filter-db <option> <database> [-to=basenum|bool|rcf]`

Three different input and output file formats are supported (.basenum, .bool and .rcf). In the input case, the format is decided automatically by the extension of the file. In the output case, .rcf format is the default one but any of the other two formats can also be specified.

Horizontal Filtering

Option: `-attributes=<file>`, where the configuration file `<file>` can have two forms.

The configuration file has one line. The configuration file has just one line, a list of attributes, e.g.

```
1 2
```
,
which means: keep rows that have attributes 1 *and* 2. Output:

```
[Relational Context]
Default Name
[Binary Relation]
Name_of_dataset
o3 | o5 |
a | b | c | d | e |
1 1 1 0 1
1 1 1 0 1
[END Relational Context]
```

The configuration file has two lines. The configuration file has one more line, containing a number, e.g.

```
1 2 3 5
3
```
,
which means: keep rows that have at least three attributes of {1,2,3,5}. That is: keep rows that have the following attributes: 235 or 135 or 125 or 123 or 1235. Output:

```
[Relational Context]
Default Name
[Binary Relation]
Name_of_dataset
o2 | o3 | o5 |
a | b | c | d | e |
0 1 1 0 1
1 1 1 0 1
1 1 1 0 1
[END Relational Context]
```

Vertical Filtering

Keep some columns. The option to use is `-columnkeep=<file>`, where the configuration file `<file>` can have one line, a list of attributes, e.g.

```
1 2 3
```
,
which means: in rows only keep the following columns: 1 *and* 2 *and* 3. Note that the other columns are still present, but all their values are set to 0.

Output:

```
[Relational Context]
Default Name
[Binary Relation]
Name_of_dataset
o1 | o2 | o3 | o4 | o5 |
a | b | c | d | e |
1 0 1 0 0
0 1 1 0 0
1 1 1 0 0
0 1 0 0 0
1 1 1 0 0
[END Relational Context]
```

Delete some columns. The option to use is `-columndelete=<file>`, where the configuration file `<file>` can have one line, a list of attributes, e.g.

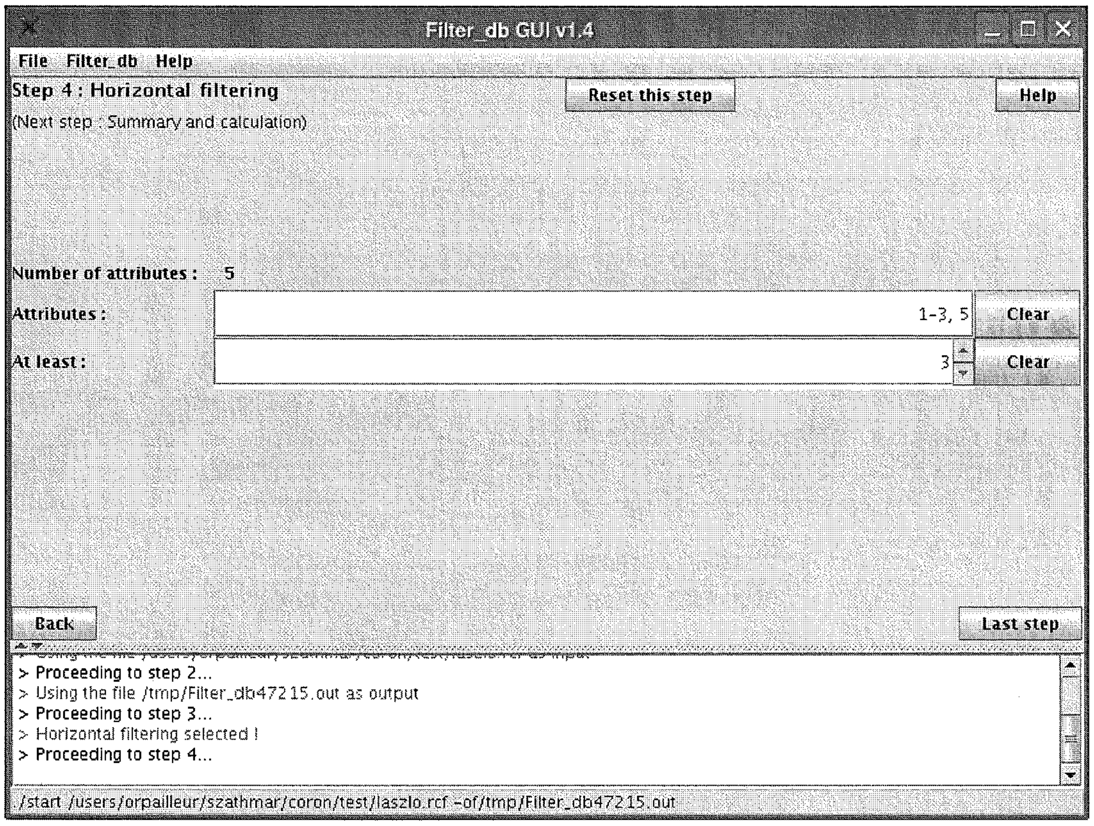
```
1 3
```
,
which means: delete the 1st and 3rd columns in each row of the dataset. Note that these columns are *still* present, but all their values are set to 0.

Output:

```
[Relational Context]
Default Name
[Binary Relation]
Name_of_dataset
o1 | o2 | o3 | o4 | o5 |
a | b | c | d | e |
0 0 0 1 0
0 1 0 0 1
0 1 0 0 1
0 1 0 0 1
0 1 0 0 1
[END Relational Context]
```

Graphical User Interface

Filter-DB also has a graphical frontend. Figure G.11 shows the input screen of horizontal filtering, and Figure G.12 shows the input screen of vertical filtering.

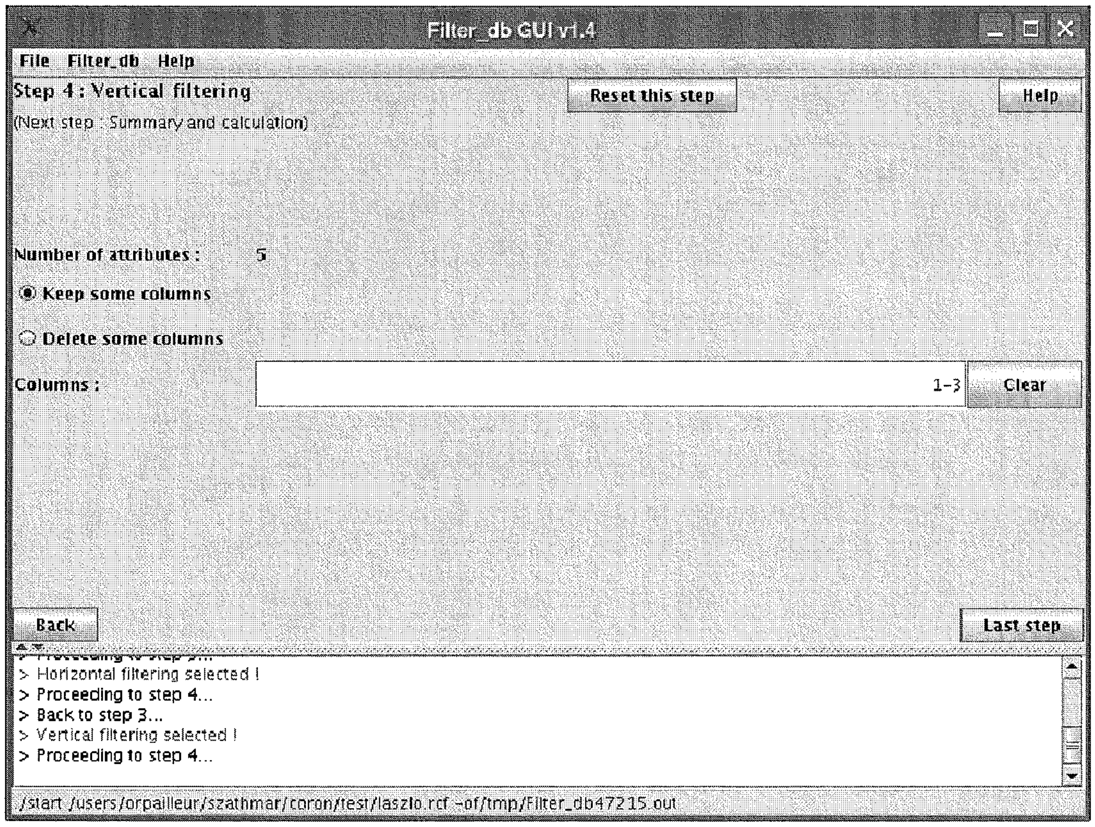


Step 4 : Horizontal filtering
(Next step : Summary and calculation)

Number of attributes : 5
Attributes : 1-3, 5
At least : 3

```
> Proceeding to step 2...
> Using the file /tmp/Filter_db47215.out as output
> Proceeding to step 3...
> Horizontal filtering selected !
> Proceeding to step 4...
./start /users/orpailleur/szathmar/coron/test/laszlo.rcf -of/tmp/Filter_db47215.out
```

Figure G.11: Horizontal filtering.



Step 4 : Vertical filtering
(Next step : Summary and calculation)

Number of attributes : 5
[✓ Keep some columns]
Delete some columns
Columns : 1-3

```
> Horizontal filtering selected !
> Proceeding to step 4...
> Back to step 3...
> Vertical filtering selected !
> Proceeding to step 4...
./start /users/orpailleur/szathmar/coron/test/laszlo.rcf -of/tmp/Filter_db47215.out
```

Figure G.12: Vertical filtering.

G.3.2 Filter-Compl

Filter-Compl (Filter Complement) is a utility for filtering certain objects/individuals from an .rcf file. Using AssRuleX we can generate rules, and with the `-examples` option we can look at the positive and negative examples. Idea: take the complement of the positive examples and work with the rest. The output of this software is another .rcf file that contains the complements.

Usage: `filter-compl <database> <config_file_with_positive_examples>`

There are two compulsory parameters: **(1)** the database file in .rcf format, and **(2)** a text file with the positive examples.

Let us see a concrete example.

Input

The database file:

```
[Relational Context]
Default Name
[Binary Relation]
Name_of_dataset
o1 | o2 | o3 | o4 | o5 |
a | b | c | d | e |
1 0 1 1 0
0 1 1 0 1
1 1 1 0 1
0 1 0 0 1
1 1 1 0 1
[END Relational Context]
```

The configuration file with the positive examples:

```
[o1,o2,o5]
```

Output

```
# This file contains the complements of the following individuals:
# [o1, o2, o5]

[Relational Context]
Default Name
[Binary Relation]
Name_of_dataset
o3 | o4
a | b | c | d | e |
1 1 1 0 1
0 1 0 0 1
[END Relational Context]
```

G.4 Post-processing of the Extracted Rules

G.4.1 Filter-Rules

Filter-Rules is a utility for post-processing rules produced by AssRuleX. With this utility we can filter rules that satisfy certain criteria.

Usage: `filter-rules <rules_file> <reg_exp> <option_1> <option_2>`

The parameters are the following:

`<rules_file>`	is a text file that contains association rules
`<reg_exp>`	is a Java regular expression
`<option_1>`	is either '`-delete`' or '`-keep`'
`<option_2>`	is either '`-left`', '`-right`' or '`-any`'

`<option_1>` has two possible values, which decide what to do with a rule that matches our criterion: delete it or keep it.

`<option_2>` has three possible values, that determine the side on which we search for the given attribute: left side, right side or both sides.

Examples

Let us suppose that we have the following rules in a file called `example.txt`:

```
{ANP_7ValVal} => {Hypertension}
{ADRB3_64TrpTrp} => {Hypertension}
{ANP_8Val, ENaCa_493TrpTrp} => {Hypertension}
{Hypertension, FCER1B_237GluGlu} => {LPL_9AspAsp}
{Hypertension, CCR3_39ProPro, LPL-93TT, LPL_291AsnAsn} => {LPL_9AspAsp}
{Hypertension, LPL-93TT} => {LPL_9AspAsp}
```

(1) `./start test/example.txt "Val" -keep -left`

This means: show all rules that have attributes on the left side with the "Val" substring. Result:

```
{ANP_7ValVal} => {Hypertension}
{ANP_8Val, ENaCa_493TrpTrp} => {Hypertension}
```

Note that `"Val"` is not a simple string, but a Java regular expression that matches any attribute that contains the substring "Val". If we want exact matching, then we should use `"^Val$"` instead. Note that with `"^Val$"` we could not find any matching rule in this example.

(2) `./start test/example.txt "^Hypertension$" -keep -right`

This means: show the rules that have the attribute "Hypertension" on the right side. Result:

```
{ANP_7ValVal} => {Hypertension}
{ADRB3_64TrpTrp} => {Hypertension}
{ANP_8Val, ENaCa_493TrpTrp} => {Hypertension}
```

(3) `./start test/example.txt "^LPL-93TT$" -delete -any`

This means: delete rules that contain the attribute "LPL-93TT". Actually, it means displaying rules that do not contain the attribute "LPL-93TT". Result:

```
{ANP_7ValVal} => {Hypertension}
{ADRB3_64TrpTrp} => {Hypertension}
{ANP_8Val, ENaCa_493TrpTrp} => {Hypertension}
{Hypertension, FCER1B_237GluGlu} => {LPL_9AspAsp}
```

G.4.2 Rule-Coloring

Rule-Coloring is a utility that allows the coloring of some attributes. This way, even if we have a large list of association rules, the important attributes can be easily found. As input, it requires two files: **(1)** a file with the association rules, and **(2)** a color schema. The output is a colored HTML file.

Usage: `rule-coloring <rules_file> <color_schema> [-show:all|colored]`

The parameters are the following:

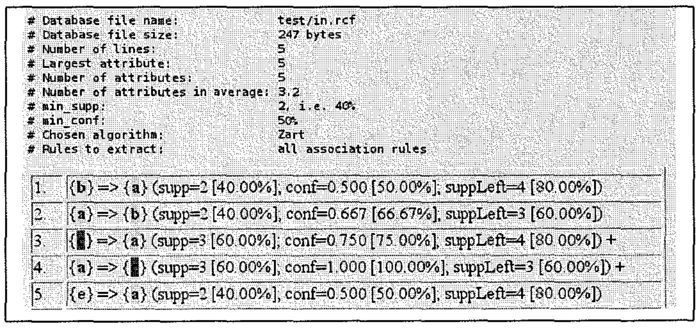
`<rules_file>`	is a text file with the rules
`<color_schema>`	is a text file with the color schema
`-show:all`	show all rules (non-colored rules too) (default)
`-show:colored`	show colored rules only

Example: `./start test/rulz.txt schema/rulz.col > tmp/out.html`

The color schema file uses the syntax of CSS files:

```
body { background-color: #EAF1FD; }
yellow { a }        // let attribute 'a' be yellow
#A0FFFF { b }       // using red-green-blue components
red { c }           // let attribute 'c' be red
```

Sample result:



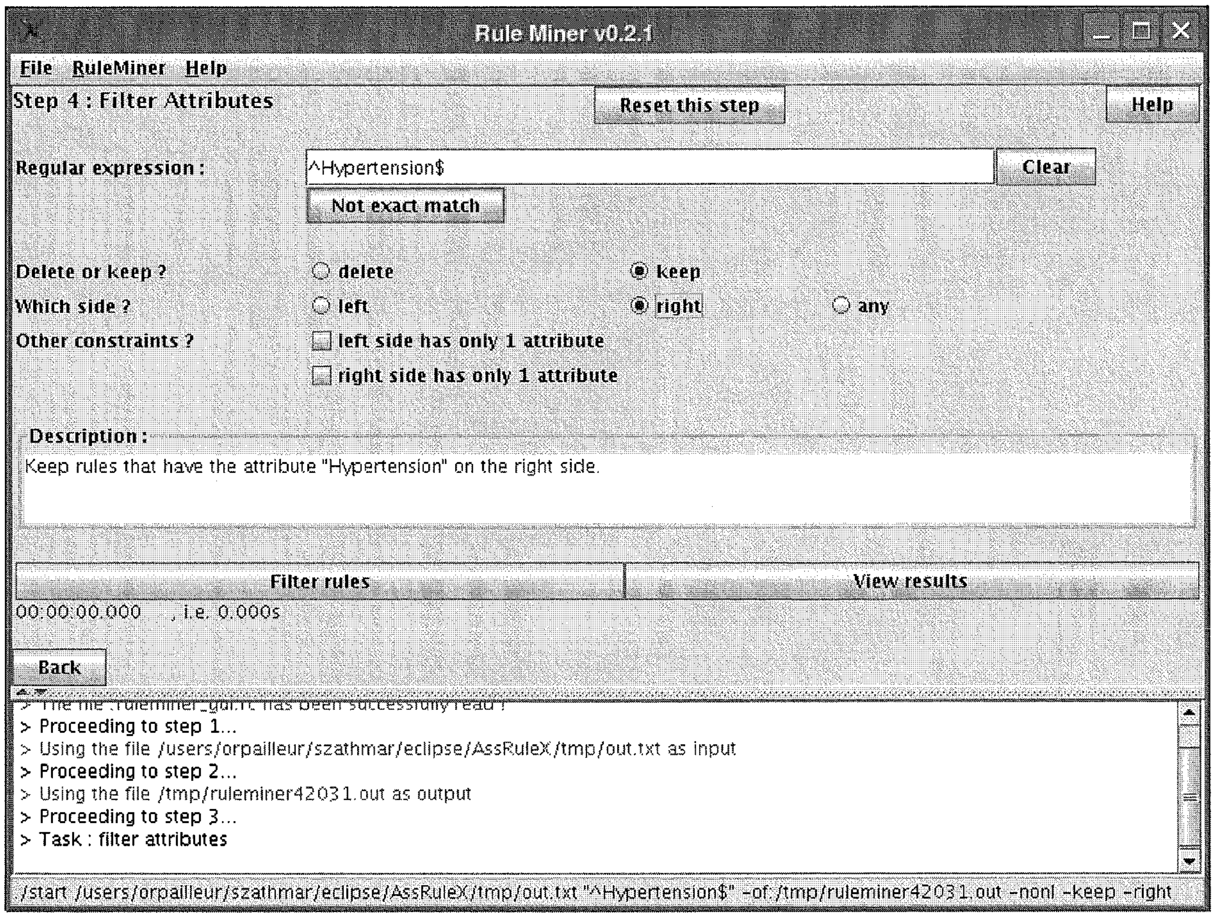
```
# Database file name:                  test/in.rcf
# Database file size:                  247 bytes
# Number of lines:                     5
# Largest attribute:                   5
# Number of attributes:                5
# Number of attributes in average:     3.2
# min_supp:                            2, i.e. 40%
# min_conf:                            50%
# Chosen algorithm:                    Zart
# Rules to extract:                    all association rules
```

1.	{b} => {a} (supp=2 [40.00%], conf=0.500 [50.00%], suppLeft=4 [80.00%])
2.	{a} => {b} (supp=2 [40.00%], conf=0.667 [66.67%], suppLeft=3 [60.00%])
3.	{█} => {a} (supp=3 [60.00%], conf=0.750 [75.00%], suppLeft=4 [80.00%]) +
4.	{a} => {█} (supp=3 [60.00%], conf=1.000 [100.00%], suppLeft=3 [60.00%]) +
5.	{e} => {a} (supp=2 [40.00%], conf=0.500 [50.00%], suppLeft=4 [80.00%])

G.4.3 RuleMiner: a GUI for Post-processing Modules

The user interface of the CORON system has a module called RULEMINER, which integrates the GUIs of the post-processing modules. RULEMINER consists of three parts:

1. a graphical frontend for Filter-Rules (Figure G.13)
2. a graphical frontend for Rule-Coloring (Figure G.14)
3. a graphical interface for filtering association rules by support and/or confidence values (Figure G.15).



Step 4 : Filter Attributes

Regular expression : ^Hypertension$

Delete or keep ? delete [✓ keep]

Which side ? left [✓ right] any

Other constraints ? ☐ left side has only 1 attribute

☐ right side has only 1 attribute

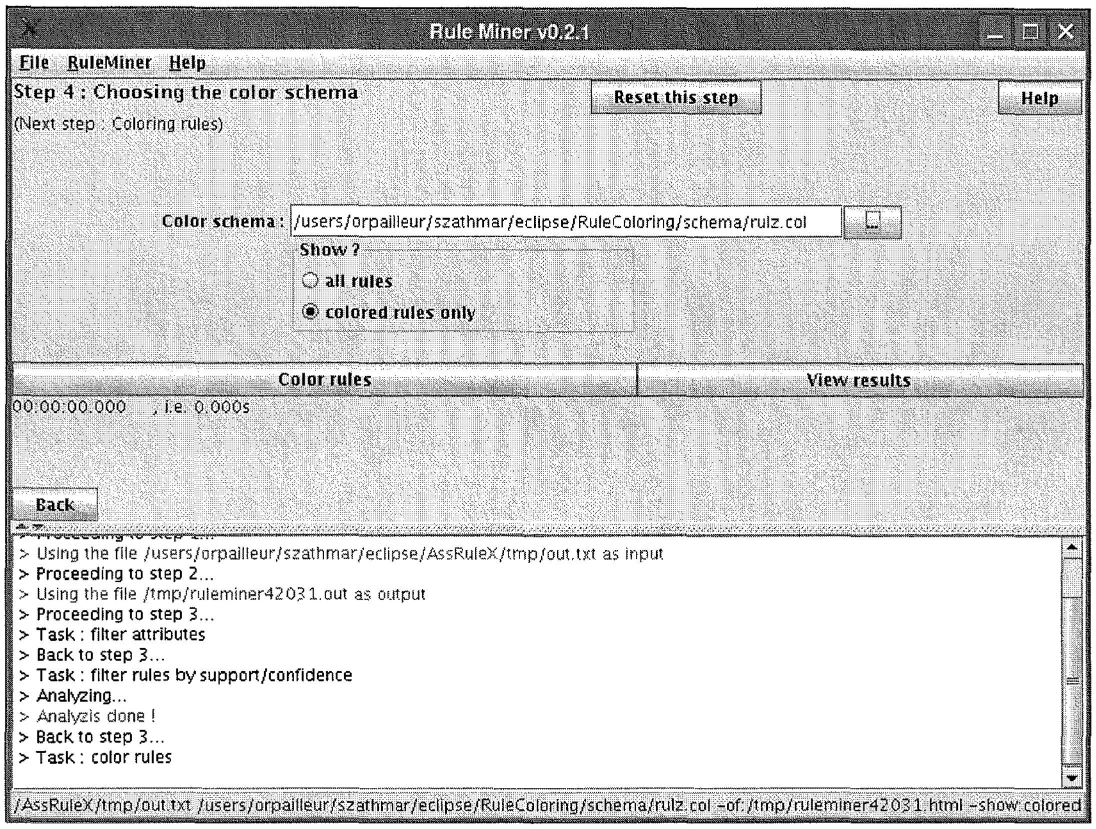
Description :
Keep rules that have the attribute "Hypertension" on the right side.

00:00:00.000 , i.e. 0.000s

```
> The file .ruleminer_gui.rc has been successfully read !
> Proceeding to step 1...
> Using the file /users/orpailleur/szathmar/eclipse/AssRuleX/tmp/out.txt as input
> Proceeding to step 2...
> Using the file /tmp/ruleminer42031.out as output
> Proceeding to step 3...
> Task : filter attributes
```

```
./start /users/orpailleur/szathmar/eclipse/AssRuleX/tmp/out.txt "^Hypertension$" -of:/tmp/ruleminer42031.out -nonl -keep -right
```

Figure G.13: Graphical frontend for Filter-Rules.



Step 4 : Choosing the color schema

(Next step : Coloring rules)

Color schema : /users/orpailleur/szathmar/eclipse/RuleColoring/schema/rulz.col

Show ?

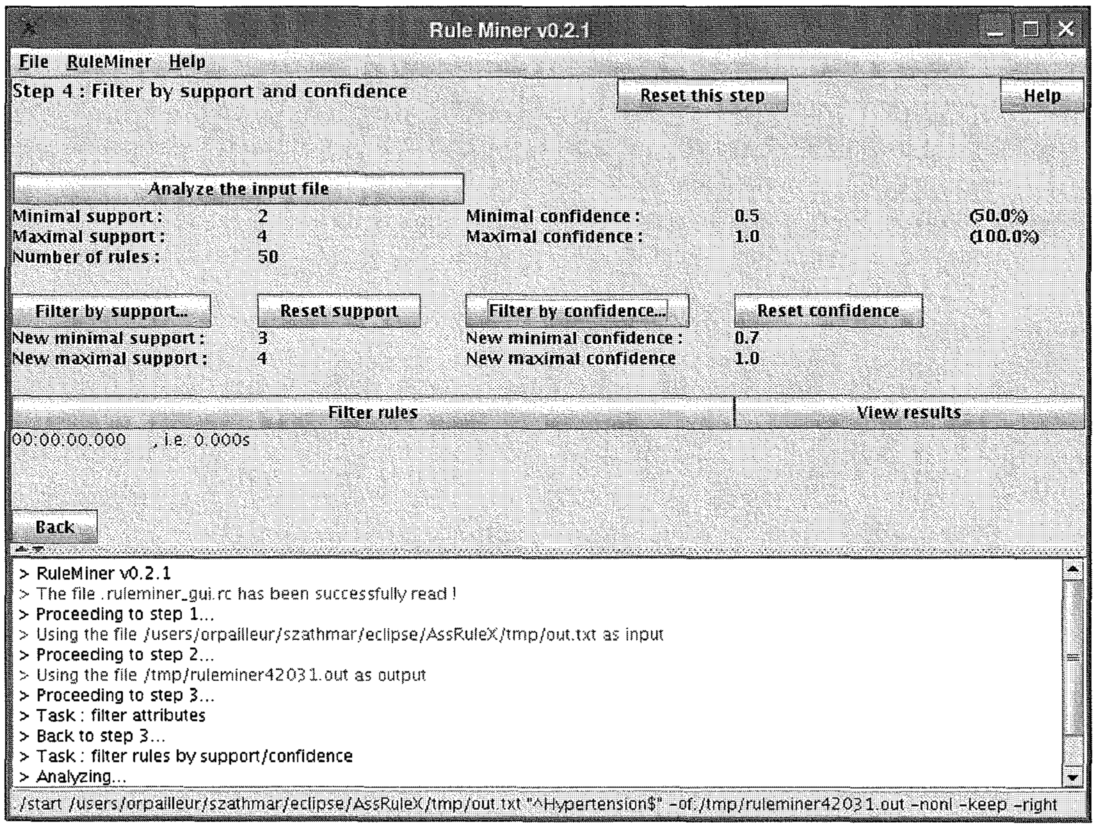
all rules [✓ colored rules only]

00:00:00.000 , i.e. 0.000s

```
> Using the file /users/orpailleur/szathmar/eclipse/AssRuleX/tmp/out.txt as input
> Proceeding to step 2...
> Using the file /tmp/ruleminer42031.out as output
> Proceeding to step 3...
> Task : filter attributes
> Back to step 3...
> Task : filter rules by support/confidence
> Analyzing...
> Analyzis done !
> Back to step 3...
> Task : color rules
```

```
/AssRuleX/tmp/out.txt /users/orpailleur/szathmar/eclipse/RuleColoring/schema/rulz.col -of:/tmp/ruleminer42031.html -show:colored
```

Figure G.14: Graphical frontend for Rule-Coloring.



Step 4 : Filter by support and confidence

Minimal support :	2	Minimal confidence :	0.5	(50.0%)
Maximal support :	4	Maximal confidence :	1.0	(100.0%)
Number of rules :	50			
New minimal support :	3	New minimal confidence :	0.7	
New maximal support :	4	New maximal confidence	1.0	

00:00:00.000 , i.e. 0.000s

```
> RuleMiner v0.2.1
> The file .ruleminer_gui.rc has been successfully read !
> Proceeding to step 1...
> Using the file /users/orpailleur/szathmar/eclipse/AssRuleX/tmp/out.txt as input
> Proceeding to step 2...
> Using the file /tmp/ruleminer42031.out as output
> Proceeding to step 3...
> Task : filter attributes
> Back to step 3...
> Task : filter rules by support/confidence
> Analyzing...
```

```
./start /users/orpailleur/szathmar/eclipse/AssRuleX/tmp/out.txt "^Hypertension$" -of:/tmp/ruleminer42031.out -nonl -keep -right
```

Figure G.15: Filtering rules by support and/or confidence.

G.5 Other Utilities

G.5.1 Analyze-Itemsets

Analyze-Itemsets is a utility for analyzing itemsets produced by Coron-base. The itemsets are analyzed by their *support* values. The input file can contain arbitrary itemsets (FIs, FCIs, etc.).

Usage:

```
analyzeItemsets.pl <input_file>
```

Example:

```
analyzeItemsets.pl btb_c73d10k.65
```

Output:

```
-----------------------------------
min_supp:          1 [0.01%]
max_supp:          6,499 [64.99%]
-----------------------------------
Number of itemsets:    2,953
-----------------------------------
Number of itemsets with support in interval ]0.0%,  10.0%[:  1,418
Number of itemsets with support in interval [10.0%, 20.0%[:  33
Number of itemsets with support in interval [20.0%, 30.0%[:  29
Number of itemsets with support in interval [30.0%, 40.0%[:  16
Number of itemsets with support in interval [40.0%, 50.0%[:  32
Number of itemsets with support in interval [50.0%, 60.0%[:  103
Number of itemsets with support in interval [60.0%, 70.0%[:  1,322
-------------------------------------------------------------------
```

This means: there are 2,953 itemsets in the input file. The itemset with the smallest support value has support 1 (0.01%), while the largest support value among the 2,953 rules is 6,499 (64.99%). At the end, it is indicated how many itemsets there are in the intervals. For instance, the interval '[10.0%, 20.0%[' contains the itemsets p where $supp(p) \geq 10\%$ and $supp(p) < 20\%$.

G.5.2 Analyze-Rules

Analyze-Rules is very similar to Analyze-Itemsets with the difference that it analyzes association rules. Rules are analyzed by their *support* values.

Usage:

```
analyzeRules.pl <input_file>
```

Example:

```
analyzeRules.pl btb_rules_genotypgroup308.95
```

Output:

```
-----------------------------------
min_supp:           1 [0.32%]
max_supp:           288 [93.51%]
-----------------------------------
Number of rules:    114
Number of RR rules: 48
Number of RF rules: 66
-----------------------------------
Number of rules with support in interval ]0.0%,  10.0%[:  17
Number of rules with support in interval [10.0%, 20.0%[:  22
Number of rules with support in interval [20.0%, 30.0%[:  25
Number of rules with support in interval [30.0%, 40.0%[:  16
Number of rules with support in interval [40.0%, 50.0%[:  9
Number of rules with support in interval [50.0%, 60.0%[:  6
Number of rules with support in interval [60.0%, 70.0%[:  7
Number of rules with support in interval [70.0%, 80.0%[:  2
Number of rules with support in interval [80.0%, 90.0%[:  5
Number of rules with support in interval [90.0%, 100.0%]: 5
-------------------------------------------------------------------
```

This means: there are 114 rules in the input file of which 48 are in the class RR (a rare itemset implies a rare itemset) and 66 are in the class RF (a rare itemset implies a frequent itemset). The rule with the smallest support value has support 1 (0.32%), while the largest support value among the 114 rules is 288 (93.51%). Then we see in the output 10 intervals, and the number of rules in each interval is displayed. For instance, the interval '[10.0%, 20.0%[' contains the rules r where $supp(r) \geq 10\%$ and $supp(r) < 20\%$.

G.5.3 getInterval

The utility "getInterval" is a Perl script that filters rules whose *support* value falls in a given interval. The interval can be open and/or closed on both sides.

Usage:

```
getInterval.pl <rule_file> <lower_limit> <upper_limit>
```

Example:

```
getInterval.pl btb_rules_genotypgroup308.95 1% 10%
```

Output:

```
{197} => {43, 51, 84, 96, 108, 114, 158, 182, 184, 186, 206, 214}
(supp=10 [3.25%]; conf=1.000 [100.00%]; suppL=10 [3.25%]; suppR=116 [37.66%];
class=RR) +
...
#
# Number of rules with support in interval [1%, 10%]: 15
#
```

Other examples:

```
./getInterval.pl btb_rules_genotypgroup308.95 [1% 10%]
```
both sides of the interval are closed

```
./getInterval.pl btb_rules_genotypgroup308.95 [1% 10%[
```
left side closed, right side open

```
./getInterval.pl btb_rules_genotypgroup308.95 ]1% 10%]
```
left side open, right side closed

```
./getInterval.pl btb_rules_genotypgroup308.95 ]1% 10%[
```
both sides open

```
./getInterval.pl btb_rules_genotypgroup308.95 1% 10%
```
is equivalent to:
```
./getInterval.pl btb_rules_genotypgroup308.95 [1% 10%]
```

G.5.4 rules2sql

rules2sql creates an SQL script from a file that contains association rules. Currently, this SQL script works only on PostgreSQL 8, but it can be easily modified for other DB systems.

Usage:	`rules2sql.pl <input_file>`
Example:	`rules2sql.pl short.txt > script.psql`

Suppose that the content of the file `short.txt` is the following:

```
{237} => {108} (supp=76 [24.68%]; conf=1.000 [100.00%]; suppL=76 [24.68%];
suppR=306 [99.35%]; class=RF) +
```

Output (`script.psql`):

```
DROP TABLE RULES;
CREATE TABLE RULES
(
leftPart       text,
rightPart      text,
suppInt        integer check (suppInt >= 0),
suppPerc       real check (suppPerc >= 0.0 AND suppPerc <= 100.0),
confReal       real check (confreal >= 0.0 and confReal <= 1.0),
confPerc       real check (confPerc >= 0.0 AND confPerc <= 100.0),
suppLeftInt    integer check (suppLeftInt >= 0),
suppLeftPerc   real check (suppLeftPerc >= 0.0 AND suppLeftPerc <= 100.0),
suppRightInt   integer check (suppRightInt >= 0),
suppRightPerc  real check (suppRightPerc >= 0.0 AND suppRightPerc <= 100.0),
class          char(2) check (class in ('RR','RF','FR','FF')),
closed         boolean,
primary key    (leftPart, rightPart)
);

INSERT INTO RULES VALUES ('{237}', '{108}', 76, 24.68, 1.000, 100.00, 76, 24.68,
306, 99.35, 'RF', true);
```

Working with PostgreSQL

Here we list some of the most important commands of PostgreSQL and we give some sample queries on the rules stored in the database.

`psql -U <user_name>`
connect to the server with a given username

`psql -U <user_name> -d <database>`
connect with a given username to a given database

Tip: create a database called "assrulex", and store association rules in it:

```
psql=> \l
```
list of databases

```
psql=> create database assrulex;
```
create a database called "assrulex"

```
psql -U <user_name> -d assrulex
```
easy connection (it is recommended to put it in a batch file)

```
psql=> \d
         List of relations
 Schema | Name  | Type  |  Owner
--------+-------+-------+----------
 public | rules | table | szathmar
(1 row)
```

Some sample SQL queries on the database:

(1) number of rules:

```
select count(*)
from rules;
```

(2) number of RR rules:

```
select count(*)
from rules
where class='RR';
```

(3) which rule(s) has (have) the smallest support?

```
select *
from rules
where suppInt = (select min(suppInt)
                 from rules);
```

(4) minimum and maximum support values:

```
select min(suppInt)  as min_supp,
       min(suppPerc) as "min_supp %",
       max(suppInt)  as max_supp,
       max(suppPerc) as "max_supp %"
from rules;
```

(5) number of rules whose support is between 0% and 10%:

```
select count(*)
from rules
where suppPerc between 0 and 10;
```

G.6 LeCo

LECO (Lattice Constructor) is a module of the CORON platform. The goal of LECO is to construct Hasse diagrams of formal concept lattices.

Usage: `leco [switches] <database> <min_supp> [-alg:<alg>]`

There are two compulsory parameters: **(1)** the database file (in .basenum, .bool or .rcf format), and **(2)** minimum support (in absolute or relative value, e.g. 2 or 40%).

There are two kinds of switches:

1. `-option` (example: `-names`)
2. `-key:value` (example: `-alg:charm`)

The algorithm to be used can be specified with the `-alg:<alg>` switch. The available algorithms are *Charm* (default) and *Close*. We recommend using *Charm* since in most cases it is more efficient than *Close*.

Other options:

`--help`	help information
`--version`, `-V`	version information
`--update`	check for a new version

Verbosity options:

`-v:m`	memory usage
`-v:f`	function information (which function is called)
`-v:t`	time information (runtime)

These options can be combined with `-vc`:
`-vc:mf` equivalent to `-v:m␣-v:f`

The verbosity options allow the displaying of some additional information while the program is running. These kind of feedbacks are always redirected to the standard error, and these lines start with a '>' sign. Because of the redirection to stderr, we can avoid cluttering the normal result with this information. Statistical information is sent to the standard output, and these lines start with a '#' sign. This way, these lines can be easily filtered.

If we only want to analyze the input dataset without any calculation, use the `-stat` option. In this case the program terminates after showing the database statistics.

The `-names` option is highly recommended. It works only with .rcf files. With this option, attribute numbers can be replaced with their names.

LECO works in the following way: as input, it takes the frequent closed itemsets (FCIs) that are generated by Coron-base. FCIs are the intent parts in the concepts of a Hasse diagram. LECO can, in addition: **(1)** find extents (i.e. find concepts), **(2)** find order (construct the Hasse diagram), and **(3)** visualize the Hasse diagram.

Examples

(1) Find concepts (extent + intent).

```
./start test/laszlo.rcf 2 -names -ext
```

Note that there is no order yet. The result is sent to the standard output.

(2) Find order among concepts.

```
./start test/laszlo.rcf 2 -names -ext -order
```

The result is sent to the standard output.

(3) Visualize the Hasse diagram of the concept lattice.

```
./start test/laszlo.rcf 2 -names -ext -order -dot:ext -dot:int
```

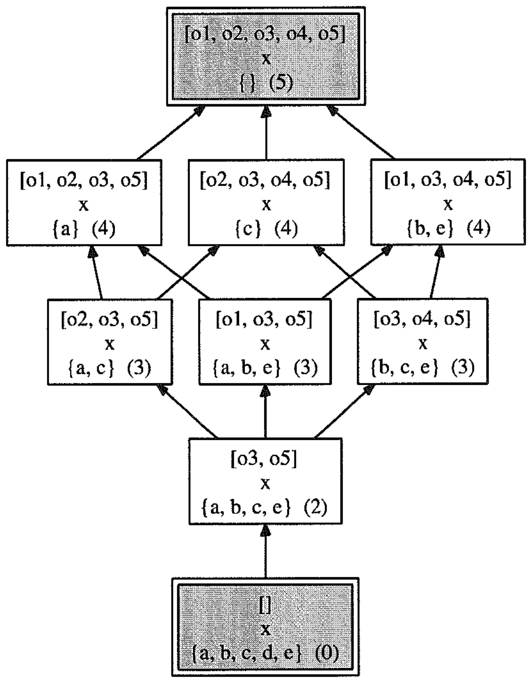
In this case two kinds of results are written:

- the first result is written to the standard output
- the second result (.dot description of the diagram) is written to the file `./graphviz/lattice.dot`

The second result can be processed with the GraphViz utility:

```
dot -Tgif lattice.dot -o lattice.gif
```

The result is a GIF image of the Hasse diagram:



In each node (for instance the node "`[o3, o5] x {a, b, c, e} (2)`") the following information is shown: **(1)** extent (objects), **(2)** intent (attributes) and **(3)** support.
The two gray nodes are the Top and Bottom elements. LECo always shows them in the Hasse diagram, even if the support of the Bottom element is less than the specified minimum support.

(4) Suppressing the standard output.

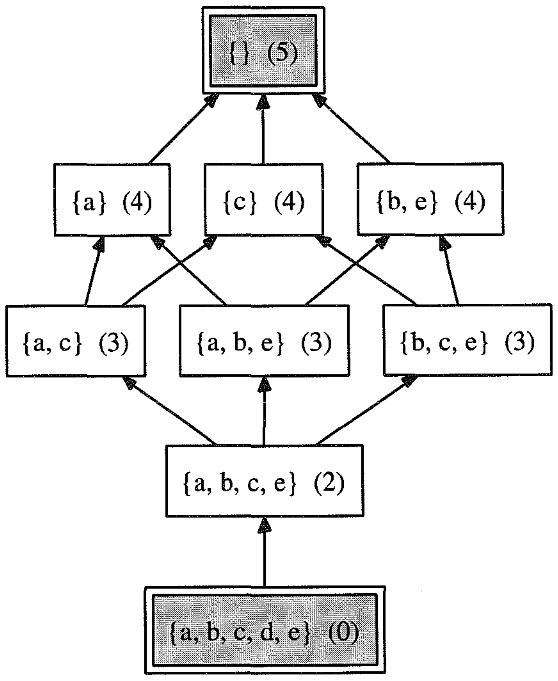
If we only want the .dot file, and we do not want to display the result on the standard output, use the `-null` switch:

```
./start test/laszlo.rcf 2 -names -ext -order -dot:ext -dot:int -null
```

(5) Hiding the extents.

The calculation and visualization of the extents can be disabled:

```
./start test/laszlo.rcf 2 -names -order -dot:int -null
```



That is: do not use the switches `-ext` and `-dot:ext`.

G.7 Using the Coron API

Here we demonstrate the ease with which CORON can be integrated into other software written in Java. The algorithms encapsulated in CORON that compute the necessary itemsets, can be called from any Java software by means of the API library provided by CORON.

In the example below we simply specify the input dataset and the minimum support threshold. Then, we call the *Charm* algorithm of CORON that extracts frequent closed itemsets. These itemsets are saved in a vector.

```
import java.util.Enumeration;
import java.util.Vector;

import coron.algorithm.Charm4b;
import coron.datastructure.charm.HashElem;
import coron.helper.C;
import coron.helper.Database;
import coron.helper.FileFormatHandler;

public class Calling_Charm
{
   /**
    * Input dataset.
    */
   private String rcf_file = "test/laszlo.rcf";

   /**
    * Minimum support.
    */
   private int min_supp = 2;

   /**
    * Entry point.
    */
   public static void main(String[] args)
   {
      Calling_Charm main = new Calling_Charm();
      main.start(args);
   }

   /**
    * The real "main" function.
    */
   public void start(String[] args)
   {
      Database.setDatabaseFileStr(this.rcf_file);
      Database.setDatabaseFileType(C.FT_RCF);

```

```
        FileFormatHandler.readFile();

        Charm4b charm = new Charm4b(Database.getDatabase(),
                                    this.min_supp);
        charm.start();   // start calculation
        Vector fciV = charm.getFCIs();

        processFCIs(fciV);
    }

    /**
     * Process FCIs.
     *
     * @param fciV Vector of FCI vectors.
     */
    private void processFCIs(Vector fciV)
    {
        Vector list;
        Enumeration in;
        HashElem elem;

        for (Enumeration e = fciV.elements(); e.hasMoreElements(); )
        {
            list = (Vector) e.nextElement();
            if (list == null) continue;
            // else
            for (in = list.elements(); in.hasMoreElements(); )
            {
                elem = (HashElem) in.nextElement();
                System.out.println(elem.toStringName());
            }
        }
    }
}
```

The output of this program is the following:

```
{a, b, c, e} (2) +
{a, b, e} (3) +
{a, c} (3) +
{a} (4) +
{b, c, e} (3) +
{b, e} (4) +
{c} (4) +
```

Bibliography

[AAP00] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *KDD '00: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 108–118. ACM Press, 2000.

[AHU85] A.V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data structures and algorithms.* Addison Wesley, 1985.

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings, ACM SIGMOD Conference on Management of Data, Washington, D.C.*, pages 207–216, 1993.

[AMS$^+$96] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in knowledge discovery and data mining*, pages 307–328. American Association for Artificial Intelligence, 1996.

[AS94] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[AS95] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. on Data Engineering (ICDE '95)*, pages 3–14, Mar 1995.

[BA96] R. J. Brachman and T. Anand. The Process of Knowledge Discovery in Databases. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 37–57, Menlo Park, California, 1996. AAAI Press / MIT Press.

[Bay98] R. J. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 85–93. ACM Press, 1998.

[BBJ$^+$02] C. Becquet, S. Blachon, B. Jeudy, J-F. Boulicaut, and O. Gandrillon. Strong-association-rule mining for large-scale gene-expression data analysis: a case study on human SAGE data. *Genome Biology*, 3(12):research0067.1–research0067.16, 2002.

[BBR00] J-F. Boulicaut, A. Bykowski, and C. Rigotti. Approximation of Frequency Queries by Means of Free-Sets. In *Proceedings of PKDD 2000, Lyon, France*, pages 75–85. Springer Berlin / Heidelberg, 2000.

[BBR03] J-F. Boulicaut, A. Bykowski, and C. Rigotti. Free-Sets: A Condensed Representation of Boolean Data for the Approximation of Frequency Queries. *Data Mining and Knowledge Discovery*, 7(1):5–22, Jan 2003.

[BGKM02] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino. On the Complexity of Generating Maximal Frequent and Minimal Infrequent Sets. In *STACS '02: Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, pages 133–141, London, UK, 2002. Springer-Verlag.

[BHC$^+$95] J. Bouyer, D. Hémon, S. Cordier, F. Derriennic, I. Stücker, B. Stengel, and J. Clavel. *Epidémiologie : Principes et méthodes quantitatives.* Editions INSERM, 1995.

[BHS02] B. Berendt, A. Hotho, and G. Stumme. Towards Semantic Web Mining. In I. Horrocks and J. Hendler, editors, *The Semantic Web - ISWC 2002*, Lecture Notes in Artificial Intelligence 2342, pages 264–278. Springer, Berlin, 2002.

[BLNN04a] S. Berasaluce, C. Laurenço, A. Napoli, and G. Niel. An Experiment on Knowledge Discovery in Chemical Databases. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *Knowledge Discovery in Databases: PKDD 2004, Pisa*, Lecture Notes in Artificial Intelligence 3202, pages 39–51. Springer, Berlin, 2004.

[BLNN04b] S. Berasaluce, C. Laurenço, A. Napoli, and G. Niel. Data mining in reaction databases: extraction of knowledge on chemical functionality transformations. Technical Report A04-R-049, LORIA, Nancy, 2004.

[BM70] M. Barbut and B. Monjardet. *Ordre et classification.* Hachette, 1970.

[BMUT97] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 255–264. ACM Press, 1997.

[BR95] P. S. Bachorik and J. W. Ross. National Cholesterol Education Program recommendations for measurement of low-density lipoprotein cholesterol: executive summary. The National Cholesterol Education Program Working Group on Lipoprotein Measurement. *Clin. Chem.*, 41(10):1414–1420, 1995.

[BR01] A. Bykowski and C. Rigotti. A condensed representation to find frequent patterns. In *PODS '01: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 267–273. ACM Press, 2001.

[BTP$^+$00a] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal. Mining frequent patterns with counting inference. *SIGKDD Explor. Newsl.*, 2(2):66–75, 2000.

[BTP$^+$00b] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal. Mining Minimal Non-Redundant Association Rules Using Frequent Closed Itemsets. In J. Lloyd *et al.*, editor, *Proc. of the Computational Logic (CL'00)*, volume 1861 of *Lecture Notes in Artificial Intelligence - LNAI*, pages 972–986. Springer, 2000.

[BTP$^+$02] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal. Pascal : un algorithme d'extraction des motifs fréquents. *Technique et science informatiques*, 21(1):65–95, 2002.

[Car83] J. G. Carbonell. Learning by analogy: Formulating and generalizing plans from past experience. In Michalski, R. S. and Carbonell, J. G. and Mitchell, T. M., editor, *Machine Learning, An Artificial Intelligence Approach*, chapter 5, pages 137–161. Tioga Press, 1983.

[CCH03] T. J. Chen, L. F. Chou, and S. J. Hwang. Application of a data-mining technique to analyze coprescription patterns for antacids in Taiwan. *Clin. Ther.*, 25(9):2453–2463, 2003.

[CDF$^+$01] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J.D. Ullman, and C. Yang. Finding Interesting Associations without Support Pruning. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):64–78, 2001.

[CG02] T. Calders and B. Goethals. Mining All Non-derivable Frequent Itemsets. In *PKDD '02: Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 74–85, London, UK, 2002. Springer-Verlag.

[CG05] T. Calders and B. Goethals. Depth-first non-derivable itemset mining. In *Proc. SIAM Int. Conf. on Data Mining SDM '05, Newport Beach (USA)*, Apr 2005.

[CH03] C. Creighton and S. Hanash. Mining gene expression databases for association rules. *Bioinformatics*, 19(1):79–86, 2003.

[CNT03] H. Cherfi, A. Napoli, and Y. Toussaint. Toward a text mining methodology using frequent itemset and association rule extraction. In M. Nadif, A. Napoli, E. San-Juan, and A. Sigayret, editors, *Journées de l'informatique Messine (JIM-2003), Knowledge Discovery and Discrete Mathematics, Metz*, pages 285–294. INRIA, 2003.

[CNT05] H. Cherfi, A. Napoli, and Y. Toussaint. Towards a Text Mining Methodology Using Association Rules Extraction. *Soft Computing*, 10(5):431–441, 2005.

[Col01] R. Cole. Automated Layout of Concept Lattices Using Layered Diagrams and Additive Diagrams. In *24th Australasian Computer Science Conference (ACSC '01), Queensland, Australia*, pages 47–60. IEEE Computer Society, 2001.

[CR04] C. Carpineto and G. Romano. *Concept Data Analysis: Theory and Applications.* John Wiley & Sons, Chichester, UK, 2004.

[CRB05] T. Calders, C. Rigotti, and J-F. Boulicaut. A Survey on Condensed Representations for Frequent Sets. In J-F. Boulicaut, L. de Raedt, and H. Mannila, editors, *Constraint-Based Mining*, volume 3848 of *LNCS*. Springer-Verlag, 2005.

[CWS$^+$04] Hoppe C., Klitz W., Cheng S., Apple R., Steiner L., Robles L., Girard T., Vichinsky E., Styles L., and CSSCD Investigators. Gene interactions and stroke risk in children with sickle cell anemia. *Blood*, 103:2391–2396, 2004.

[dBL$^+$06] M. d'Aquin, F. Badra, S. Lafrogne, J. Lieber, A. Napoli, and L. Szathmary. Adaptation Knowledge Discovery from a Case Base. In G. Brewka, editor, *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI-06), Trento*, pages 795–796. IOS Press, 2006.

[DL01] S. Dzeroski and N. Lavrac, editors. *Relational Data Mining.* Springer, Berlin, 2001.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, Cambridge, UK, 1990.

[DT99] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3:7–36, 1999.

[Dun03] M. H. Dunham. *Data Mining – Introductory and Advanced Topics.* Prentice Hall, Upper Saddle River, NJ, 2003.

[Duq99] V. Duquenne. Latticial structures in data analysis. *Theoretical Computer Science*, 217:407–436, 1999.

[FHLW03] D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, editors. *Spinning the Semantic Web.* The MIT Press, Cambridge, Massachusetts, 2003.

[FPSSU96] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* AAAI Press / MIT Press, Menlo Park, California, 1996.

[GD86] J. L. Guigues and V. Duquenne. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences Humaines*, 95:5–18, 1986.

[GGKS04] B. Ganter, P. A. Grigoriev, S. O. Kuznetsov, and M. V. Samokhin. Concept-Based Data Mining with Scaled Labeled Graphs. In K. E. Wolff, H. D. Pfeiffer, and H. S. Delugach, editors, *Conceptual Structures at Work: Proceedings of the 12th International Conference on Conceptual Structures, ICCS 2004, Huntsville, AL*, Lecture Notes in Artificial Intelligence 3127, pages 94–108. Springer, Berlin, 2004.

[GMM95] R. Godin, G. W. Mineau, and R. Missaoui. Méthodes de classification conceptuelle basées sur les treillis de Galois et applications. *Revue d'intelligence artificielle*, 9(2):105–137, 1995.

[Goe03] B. Goethals. Survey on Frequent Pattern Mining. Manuscript, 2003.

[GR01] B. Ganter and S. Rudolph. Formal Concept Analysis Methods for Dynamic Conceptual Graphs. In H. S. Delugach and G. Stumme, editors, *Conceptual Structures: Broadening the Base – 9th International Conference on Conceptual Structures, ICCS-2001, Stanford*, Lecture Notes in Artificial Intelligence 2120, pages 143–156. Springer, Berlin, 2001.

[Gue90] A. Guenoche. Construction du treillis de Galois d'une relation binaire. *Mathématiques, Informatique et Sciences Humaines*, 109:41–53, 1990.

[GVM93] A. Guenoche and I. Van Mechelen. Galois Approach to the Induction of Concepts. In I. Van Mechelen, J. Hampton, R. S. Michalski, and P. Theuns, editors, *Categories and Concepts: Theoretical Views and Inductive Data Analysis.* Academic Press Ltd, 1993.

[GW99] B. Ganter and R. Wille. *Formal concept analysis: mathematical foundations.* Springer, Berlin/Heidelberg, 1999.

[GZ01] K. Gouda and M. J. Zaki. Efficiently Mining Maximal Frequent Itemsets. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 163–170, Washington, DC, USA, 2001. IEEE Computer Society.

[HDST01] S.K. Harms, J.S. Deogun, J. Saquer, and T. Tadesse. Discovering Representative Episodal Association Rules from Event Sequences Using Frequent Closed Episode Sets and Event Constraints. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 603–606, Washington, DC, USA, 2001. IEEE Computer Society.

[HH99] R.J. Hilderman and H.J. Hamilton. Knowledge discovery and interestingness measures: A survey. Technical Report, Department of Computer Science, University of Regina (Canada), Oct 1999.

[HK96] K. Hanney and M. T. Keane. Learning Adaptation Rules From a Case-Base. In I. Smith and B. Faltings, editors, *Advances in Case-Based Reasoning – Third European Workshop, EWCBR '96*, LNAI 1168, pages 179–192. Springer Verlag, Berlin, 1996.

[HK01] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, San Francisco, 2001.

[HMS01] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining.* The MIT Press, Cambridge (MA), 2001.

[HPY00] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM Press, 2000.

[IBM98] IBM. *Intelligent Miner for Text.* International Business Machine (IBM), 1998.

[Jay03] N. Jay. Recherche et interprétation de motifs séquentiels fréquents dans une base de données médicales, 2003. Mémoire de DEA, Université Henri Poincaré Nancy 1.

[JB02] B. Jeudy and J-F. Boulicaut. Using condensed representations for interactive association rule mining. In *Proceedings PKDD '02, volume 2431 of LNAI, Helsinki (Finland)*, pages 225–236. Springer-Verlag, 2002.

[JCK$^+$04] D. Janetzko, H. Cherfi, R. Kennke, A. Napoli, and Y. Toussaint. Knowledge-Based Selection of Association Rules for Text Mining. In *Proc. of the 16th European Conference on Artificial Intelligence – ECAI '04, Valencia, Spain*, pages 485–489. IOS Press, 2004.

[KB00] R. Kosala and H. Blockeel. Web Mining: A Survey. *SIGKDD Explorations*, 2(1):1–15, 2000.

[KG02] M. Kryszkiewicz and M. Gajek. Why to Apply Generalized Disjunction-Free Generators Representation of Frequent Patterns? In M.-S. Hacid, Z.W. Ra, D.A.

Zighed, and Y. Kodratoff, editors, *Proceedings of Foundations of Intelligent Systems: 13th International Symposium, ISMIS 2002, Lyon, France*, pages 383–392. Springer-Verlag Berlin / Heidelberg, June 2002.

[KH95] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. In *Proc. 4th Int. Symp. Large Spatial Databases (SSD '95)*, pages 47–66, Aug 1995.

[KO01] S. O. Kuznetsov and S. A. Obiedkov. Algorithms for the Construction of Concept Lattices and Their Diagram Graphs. In L. De Raedt and A. Siebes, editors, *Principles of Data Mining and Knowledge Discovery: 5th European Conference, PKDD 2001, Freiburg, Germany*, Lecture Notes in Computer Science 2168, pages 289–300. Springer-Verlag Heidelberg, 2001.

[KO02] S. O. Kuznetsov and S. A. Obiedkov. Comparing performance of algorithms for generating concept lattices. *J. Exp. Theor. Artif. Intell.*, 14(2–3):189–216, 2002.

[KR05] Y.S. Koh and N. Rountree. Finding Sporadic Rules Using Apriori-Inverse. In *PAKDD '05: Proceedings of the 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, Hanoi, Vietnam*, volume 3518 of *Lecture Notes in Computer Science*, pages 97–106. Springer, May 2005.

[KRO06] Y.S. Koh, N. Rountree, and R. O'Keefe. Mining Interesting Imperfectly Sporadic Rules. In *PAKDD '06: Proceedings of the 10th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, Singapore*, volume 3918 of *Lecture Notes in Computer Science*, pages 473–482. Springer, April 2006.

[Kry98] M. Kryszkiewicz. Representative Association Rules. In *PAKDD '98: Proceedings of the Second Pacific-Asia Conference on Research and Development in Knowledge Discovery and Data Mining*, pages 198–209, London, UK, 1998. Springer-Verlag.

[Kry01] M. Kryszkiewicz. Concise Representation of Frequent Patterns Based on Disjunction-Free Generators. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 305–312, Washington, DC, USA, 2001. IEEE Computer Society.

[Kry02] M. Kryszkiewicz. Concise Representations of Association Rules. In *Pattern Detection and Discovery*, pages 92–109, 2002.

[Kuz04] S. O. Kuznetsov. Machine Learning and Formal Concept Analysis. In P. W. Eklund, editor, *Concept Lattices, Second International Conference on Formal Concept Analysis, ICFCA 2004, Sydney, Australia*, Lecture Notes in Computer Science 2961, pages 287–312. Springer, 2004.

[LFZ99] N. Lavrac, P. A. Flach, and B. Zupan. Rule Evaluation Measures: A Unifying View. In S. Dzeroski and P. A. Flach, editors, *Inductive Logic Programming, 9th International Workshop, ILP-99, Bled, Slovenia*, Lecture Notes in Computer Science 1634, pages 174–185. Springer, 1999.

[LHM99] B. Liu, W. Hsu, and Y. Ma. Mining association rules with multiple minimum supports. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 337–341, New York, NY, USA, 1999. ACM Press.

[LK98] D.-I. Lin and Z. M. Kedem. Pincer Search: A New Algorithm for Discovering the Maximum Frequent Set. In *EDBT '98: Proceedings of the 6th International Conference on Extending Database Technology*, pages 105–119, London, UK, 1998. Springer-Verlag.

[LLFH99] H. Liu, H. Lu, L. Feng, and F. Hussain. Efficient Search of Reliable Exceptions. In *PAKDD '99: Proceedings of the Third Pacific-Asia Conference on Methodologies for Knowledge Discovery and Data Mining*, pages 194–203, London, UK, 1999. Springer-Verlag.

[Lux91] M. Luxenburger. Implications partielles dans un contexte. *Mathématiques, Informatique et Sciences Humaines*, 113:35–55, 1991.

[Man97] H. Mannila. Methods and Problems in Data Mining. In F. Afrati and P. Kolaitis, editors, *Database Theory - ICDT'97, 6th International Conference, Delphi, Greece*, Lecture Notes in Artificial Intelligence 1186, pages 41–55. Springer, Berlin, 1997.

[MBK98] R. S. Michalski, I. Bratko, and M. Kubat, editors. *Machine Learning and Data Mining.* John Wiley & Sons LTD, Chichester, 1998.

[MCHSV02] M. Mansour-Chemaly, N. Haddy, G. Siest, and S. Visvikis. Family studies: their role in the evaluation of genetic cardiovascular risk factors. *Clin. Chem. Lab. Med.*, 40(11):1085–1096, 2002.

[Mit97] T. M. Mitchell. *Machine Learning.* McGraw-Hill, Boston, Massachusetts, 1997.

[MNSVS05a] S. Maumus, A. Napoli, L. Szathmary, and S. Visvikis-Siest. Exploitation des données de la cohorte STANISLAS par des techniques de fouille de données numériques et symboliques utilisées seules ou en combinaison. In *Workshop on Fouille de Données Complexes dans un Processus d'Extraction des Connaissances - EGC 2005, Paris, France*, pages 73–76, Feb 2005.

[MNSVS05b] S. Maumus, A. Napoli, L. Szathmary, and S. Visvikis-Siest. Fouille de données biomédicales complexes : extraction de règles et de profils génétiques dans le cadre de l'étude du syndrome métabolique. In G. Perrière, A. Guénoche, and C. Gourgeon, editors, *Journées Ouvertes Biologie Informatique Mathématiques - JOBIM 2005, Lyon, France*, pages 169–173, Jul 2005. contrat de plan Etat-Région.

[MT96] H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD '96), Portland (USA)*, pages 189–194. AAAI Press, 1996.

[MT97] H. Mannila and H. Toivonen. Levelwise Search and Borders of Theories in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, September 1997.

[MTV94] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In U. M. Fayyad and R. Uthurusamy, editors, *Proc. of the AAAI Workshop on Knowledge Discovery in Databases (KDD '94)*, pages 181–192, Seattle, Washington, USA, July 1994. AAAI Press.

[NLD94] A. Napoli, C. Laurenço, and R. Ducournau. An object-based representation system for organic synthesis planning. *International Journal of Human–Computer Studies*, 41(1/2):5–32, 1994.

[oH01] National Institutes of Health. Third Report of the National Cholesterol Education Program Expert Panel on Detection, Evaluation, and Treatment of High Blood Cholesterol in Adults (Adult Treatment Panel III). NIH publ. no. 01-3670, 2001. Executive Summary. Bethesda, MD, National Institutes of Health, National Heart, Lung and Blood Institute.

[ORS98] B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proc. 1998 Int. Conf. Data Engineering (ICDE '98)*, pages 412–421, Feb 1998.

[Pas00a] N. Pasquier. *Data Mining : Algorithmes d'Extraction et de Réduction des Règles d'Association dans les Bases de Données.* Thèse de doctorat, Université Clermont-Ferrand II, Clermont-Ferrand II, 2000.

[Pas00b] N. Pasquier. Mining association rules using formal concept analysis. In *Proc. of the 8th International Conf. on Conceptual Structures (ICCS '00)*, pages 259–264. Shaker-Verlag, Aug 2000.

[Paw91] Z. Pawlak, editor. *Rough Sets. Theoretical Aspects of Reasoning about Data.* Kluwer Academic Publishers, Dordrecht, The Netherlends, 1991.

[PBTL99a] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Closed set based discovery of small covers for association rules. In *Proc. 15emes Journees Bases de Donnees Avancees, BDA*, pages 361–381, 1999.

[PBTL99b] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. *Lecture Notes in Computer Science*, 1540:398–416, 1999.

[PBTL99c] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Inf. Syst.*, 24(1):25–46, 1999.

[PBTL99d] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Pruning Closed Itemset Lattices for Association Rules. *International Journal of Information Systems*, 24(1):25–46, 1999.

[PCY95] J. S. Park, M.-S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 175–186. ACM Press, 1995.

[PHM00] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000.

[PS91] G. Piatetsky-Shapiro. *Knowledge Discovery in Databases*, chapter Discovery, Analysis, and Presentation of Strong Rules (chapter 13), pages 229–248. AAAI/MIT Press, Menlo Park, G. Piatetsky-Shapiro and W.J. Frawley edition, 1991.

[QTDDB02] J. Quentin-Trautvetter, P. Devos, A. Duhamel, and R. Beuscart. Assessing Association Rules and Decision Trees on Analysis of Diabetes Data from the DiabCare Program in France. *Studies in Health Technology and Informatics*, 90:557–561, 2002.

[RS89] C. K. Riesbeck and R. C. Schank. *Inside Case-Based Reasoning.* Lawrence Erbaum Associates, Inc., 1989.

[SAA$^+$99] G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. van de Velde, and B. Wielinga. *Knowledge Engineering and Management: the CommonKADS Methodoloy.* The MIT Press, Cambridge, MA, 1999.

[SAC$^+$99] Cheng S., Grow M. A., Pallaud C., Klitz W., Erlich H. A., Visvikis S., Chen J. J., Pullinger C. R., Malloy M. J., Siest G., and Kane J.P. A multilocus genotyping assay for candidate markers of cardiovascular disease risk. *Genome Res.*, 9:936–949, 1999.

[SBMP01] S. Stilou, P. D. Bamidis, N. Maglaveras, and C. Pappas. Mining Association Rules from Clinical Databases: An Intelligent Diagnostic Process in Healthcare. In *MEDINFO 2001: Proceedings of the 10th World Congress on Medical Informatics*, pages 1399–1403. IOS Press, 2001.

[SMP$^+$06] L. Szathmary, S. Maumus, P. Petronin, Y. Toussaint, and A. Napoli. Vers l'extraction de motifs rares. In G. Ritschard and C. Djeraba, editors, *Extraction et gestion des connaissances (EGC 2006), Lille*, pages 499–510. RNTI-E-6, Cépaduès-Éditions Toulouse, 2006.

[SN05] L. Szathmary and A. Napoli. CORON: A Framework for Levelwise Itemset Mining Algorithms. In B. Ganter, R. Godin, and E. Mephu Nguifo, editors, *Supplementary Proceedings of The Third International Conference on Formal Concept Analysis – ICFCA '05, Lens, France*, pages 110–113, Feb 2005.

[SNK05] L. Szathmary, A. Napoli, and S. O. Kuznetsov. ZART: A Multifunctional Itemset Miner Algorithm. LORIA Research Report A05-R-013, Feb 2005.

[SON95] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. of the 21th International Conf. on Very Large Data Bases (VLDB '95)*, pages 432–444, Sep 1995.

[SON98] A. Savasere, E. Omiecinski, and S. B. Navathe. Mining for Strong Negative Associations in a Large Database of Customer Transactions. In *ICDE '98: Proceedings of the Fourteenth International Conference on Data Engineering*, pages 494–502. IEEE Computer Society, 1998.

[SS04] S. Staab and R. Studer, editors. *Handbook on Ontologies.* Springer, Berlin, 2004.

[STB$^+$02] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Computing Iceberg Concept Lattices with TITANIC. *Data and Knowledge Engineering*, 42(2):189–222, 2002.

[STT98] R. H. Sloan, K. Takata, and Gy. Turan. On frequent sets of Boolean matrices. *Annals of Mathematics and Artificial Intelligence*, 24(1-4):193–209, 1998.

[STVN04] A. Salleb, T. Turmeaux, C. Vrain, and C. Nortet. Mining Quantitative Association Rules in a Atherosclerosis Dataset. In *In Proc. of the PKDD Discovery Challenge (co-located with the 6th European Conference on Principles and Practice of Knowledge Discovery in databases), Pisa, Italy*, pages 98–103, 2004.

[SVH$^+$98] G. Siest, S. Visvikis, B. Herbeth, R. Gueguen, M. Vincent-Viry, C. Sass, B. Beaud, E. Lecomte, J. Steinmetz, J. Locuty, and P. Chevrier. Objectives, Design and Recruitment of a Familial and Longitudinal Cohort for Studying Gene-Environment Interactions in the Field of Cardiovascular Risk: The Stanislas Cohort. *Clinical Chemistry and Laboratory Medicine (CCLM)*, 36(1):35–42, Jan 1998.

[SWW98] G. Stumme, R. Wille, and U. Wille. Conceptual Knowledge Discovery in Databases Using Formal Concept Analysis Methods. In J. Zytkow and M. Quafafou, editors, *Principles of Data Mining and Knowledge Discovery (Proceedings PKDD'98, Nantes)*, Lecture Notes in Artificial Intelligence 1510, pages 450–458, Berlin, 1998. Springer.

[TKS02] P. N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (KDD'02), Edmonton, Canada*, pages 183–193, 2002.

[Toi96] H. Toivonen. Sampling large databases for association rules. In *Proc. of the 22nd International Conf. on Very Large Data Bases (VLDB '96)*, pages 134–145, Sep 1996.

[VGRR03] P. Valtchev, D. Grosser, C. Roume, and Hacene M. R. Galicia: an open platform for lattices. In *Supplementary Proceedings of The 11th International Conference on Conceptual Structures (ICCS '03), Dresden, Germany*, 2003.

[VL00] P. Vismara and C. Laurenço. An abstract representation for molecular graphs. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 51:343–366, 2000.

[VMG04] P. Valtchev, R. Missaoui, and R. Godin. Formal Concept Analysis for Knowledge Discovery and Data Mining: The New Challenges. In *Proc. of the 2nd Intl. Conf. on Formal Concept Analysis*, pages 352–371. Springer Verlag, Feb 2004.

[VW94] F. Vogt and R. Wille. TOSCANA – A Graphical Tool for Analyzing and Exploring Data. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing*, volume 894 of *Lecture Notes in Computer Science*, pages 226–233. Springer-Verlag, 1994.

[Wei04] G.M. Weiss. Mining with rarity: a unifying framework. *SIGKDD Explor. Newsl.*, 6(1):7–19, 2004.

[WF99] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann Publishers, 1999.

[WF00] I. H. Witten and E. Franck. *Data Mining.* Morgan Kaufmann Publishers, San Francisco, California, 2000. (Practical machine learning tools and techniques with Java implementations – Weka).

[WHP03] J. Wang, J. Han, and J. Pei. CLOSET+: searching for the best strategies for mining frequent closed itemsets. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 236–245. ACM Press, 2003.

[Wil02] R. Wille. Why can concept lattices support knowledge discovery in databases? *Journal of Theoretical Artificial Intelligence*, 14(2/3):81–92, 2002.

[WZTS04] J. T. L. Wang, M. J. Zaki, H. T. T. Toivonen, and D. Shasha, editors. *Data Mining in Bioinformatics*. Morgan Kaufmann Publishers, Springer, Berlin, 2004.

[YHHR03] H. Yun, D. Ha, B. Hwang, and K.H. Ryu. Mining association rules on significant rare data using relative support. *Journal of Systems and Software*, 67(3):181–191, 2003.

[Zak00] M. J. Zaki. Scalable Algorithms for Association Mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.

[ZG03] M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 326–335, New York, NY, USA, 2003. ACM Press.

[ZH02] M. J. Zaki and C.-J. Hsiao. CHARM: An Efficient Algorithm for Closed Itemset Mining. In *SIAM International Conference on Data Mining SDM'02*, pages 33–43, Apr 2002.

[ZPOL97] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New Algorithms for Fast Discovery of Association Rules. In *Proceedings of the 3rd Int'l Conference on Knowledge Discovery in Databases*, pages 283–286, August 1997.

Glossary

Absolute support : The absolute support of an itemset P (denoted by $supp(P)$) indicates how many objects include P. Unless specified otherwise, by "support" we mean "*absolute* support".

Anti-monotonocity property : All supersets of a non-frequent itemset are non-frequent.

Closed association rule : An association rule r: $P_1 \rightarrow P_2$ is closed if $P_1 \cup P_2$ is a closed itemset.

Closed itemset : An itemset X is closed if it has no proper superset Y ($X \subset Y$) with the same support. A closed itemset is the maximal element (w.r.t. set inclusion) in its equivalence class.

Closure : The closure of an itemset X (denoted by $\gamma(X)$) is the largest superset of X with the same support. If X is equal to its closure ($X = \gamma(X)$) then X is a closed itemset.

Confidence : The confidence of an association rule r: $P_1 \rightarrow P_2$ is defined as the conditional probability that an object includes P_2, given that it includes P_1: $conf(r) = supp(P_1 \cup P_2)/supp(P_1)$.

Confident association rule : An association rule r is confident if its confidence is not less than the user-defined threshold min_conf, i.e. $conf(r) \geq min\_conf$.

Downward closure property : All subsets of a frequent itemset are frequent.

Empty set : The empty set (denoted by $\emptyset$) is the smallest possible itemset, i.e. an itemset that contains no items. By definition, its relative support is 1.0 (100%), i.e. it is included in every object of the database. The empty set is *always* a generator. If the database has a closed itemset P (different from the empty set) with relative support 1.0, then P is the closure of the empty set; otherwise the empty set is a closed itemset. From the point of view of rule generation, the empty set is interesting only if it is not a closed itemset.

Equivalence class : An equivalence class is a set of itemsets that all have the same closure. That is, itemsets in an equivalence class are equivalent w.r.t. closure equality.

Extent : An extent (a.k.a. extension) is a closed tidset, i.e. a closed set of transaction (or object) identifiers.

Frequent association rule : An association rule r: $P_1 \rightarrow P_2$ is frequent if $P_1 \cup P_2$ is a frequent itemset.

Frequent Itemset (FI) : An itemset P is frequent if its support is not less than a given minimum support, i.e. $supp(P) \geq min\_supp$.

Frequent valid association rule : A frequent and confident association rule. Unless specified otherwise, by "valid association rules" we mean "*frequent* valid association rules".

Generator : An itemset X is a generator if it has no proper subset Y ($Y \subset X$) with the same support. Generators are minimal elements (w.r.t. set inclusion) in an equivalence class.

Informative association rules : A set of rules that allows the determination of rules parameters such as support and confidence.
Intent : An intent (a.k.a. intension) is a closed itemset, i.e. a closed set of items (or attributes).

Lossless association rules : A set of rules that enables the derivation of all valid rules.

Maximal Frequent Itemset (MFI) : A frequent itemset such that all its proper supersets are not frequent (and necessarily, all its subsets are frequent). Note that maximal frequent itemsets are closed itemsets: MFI $\subseteq$ FCI $\subseteq$ FI.
Minimal Rare Generator (MRG) : A rare generator such that all its proper subsets are not rare. Note that minimal rare generators are minimal rare itemsets.
Minimal Rare Itemset (MRI) : A rare itemset such that all its proper subsets are not rare (and necessarily, all its supersets are rare). Note that minimal rare itemsets are minimal rare generators.
Minimal Zero Generator (MZG) : A zero itemset such that all its proper subsets are non-zero itemsets (and necessarily, all its supersets are zero itemsets).

Non-zero itemset : An itemset such that its support exceeds 0.

Potentially frequent candidate : An itemset such that all its proper subsets are frequent itemsets. Note that it does not mean that the candidate is frequent too.
Pseudo-closed itemset (PCI) : A (frequent) pseudo-closed itemset P is a (frequent) non-closed itemset that includes the closures of all (frequent) pseudo-closed itemsets included in P.

Rare association rule : An association rule r: $P_1 \rightarrow P_2$ is rare if $P_1 \cup P_2$ is a rare itemset.
Rare Itemset (RI) : An itemset P is rare if its support is less than or equal to the user-defined threshold max_supp, i.e. $supp(P) \leq max\_supp$. In the case of a single border it means that an itemset P is rare if it is not frequent, i.e. if its support is less than the user-defined threshold min_supp ($supp(P) < min\_supp$).
Rare valid association rule : A rare and confident association rule.

Sound association rules : A set of rules that forbids the derivation of rules that are not valid.
Strong association rule : See "valid association rule".

Valid association rule : See "frequent valid association rule".

Zero itemset : An itemset such that its support is 0. Not to be confused with the empty set.

Index

absolute minimum support, *see* barrier
absolute support, *see* support
all association rules, 61
analyst, *see* expert
antecedent, 30
anti-monotonocity, 30, 40
approximate association rule, 17, 30
approximate free-sets, 68
Apriori, 16, 23, 61, 65, 81, 85, 93, 161
Apriori-Close, 47, 164
Apriori-Inverse, 94
Apriori-Rare, 85, 97
Arima, 86
ascendant equivalence class, 41
association rule, 4, 15, 16, 30
association rules basis, 64, 72

barrier, 96, 100
basic theorem of concept lattices, 15
basis, *see* association rules basis
border, 82, 83, 88, 89
BtB, 97, 99

cache, 75, 100
candidate, 16, 31
Charm, 58, 172
Charm-MFI, 58
classification, 7
classification of association rules, 100
Close, 20, 65, 175
closed association rules, 61, 64–66
closed itemset, 11, 29, 65, 164
Closet, 36
Closet$^+$, 36
closure, 29
closure operator, 11
cohort, 109
complete lattice, 14
complex equivalence class, 40
computing environmental variables, 96
concept, 11
concept lattice, 12, 15
concise rule representation, 61, 64, 80
condensed itemset representation, 65
confidence, 17, 30, 73
confidence transitivity, 68
confident association rule, 30
consequent, 30
context, 3, 14, 29
conviction, 17, 74
cover operator, 37, 68
cross table, 14
cyclic association rule mining, 81

data analysis, 4
data mining methodology, 100, 107
data mining methods, 4
database management, 5
dCharm, 35
dEclat, 35
dependency, 17, 74
description logics, 24
diffset, 35
direct ascendant of an equivalence class, 41
direct subsumer of an equivalence class, 41
disjunction-free generators, 68
disjunction-free sets, 68, 81
downward closure, 30, 40
Duquennes-Guigues basis, 61, 80, 123

Eclat, 167
Eclat-Z, 51, 68
element reordering, 34
empty set, 43, 54, 84, 167, 173
environmental variables, *see* computing environmental variables
equivalence class, 39
exact association rule, 17, 30
example dataset, 31
expert, 1, 5, 15, 22, 74, 82

extension, *see* extent
extent, 9, 11, 14

FIMI workshops, 105
formal concept, 11, 12
formal concept analysis, 13, 29
formal context, *see* context
FP-growth, 36
FP-tree, 36
French summary, 127
frequent association rule, 30, 81
frequent closed itemset representation, 66, 68, 91, 100, 191
frequent itemset, 29
frequent itemset mining, 30

Galicia, 106, 119, 124
Galois closure, 11
Galois connection, 9, 11, 14
Galois correspondence, *see* Galois connection
Galois lattice, *see* concept lattice
generalized disjunction-free generators, 68
generator, 29
generators representation, 68, 91, 191
generic basis, 61

hash, 174
hash tree, 32
hierarchical order, *see* order
horizontal data layout, 34, 187
Hungarian summary, 156

iceberg concept lattice, 119
image, 15
imperfectly rare association rule, 93
imperfectly rare itemset, 93
infimum, 13
informative association rules, 37, 61, 68
informative basis, 61
infrequent itemset, *see* rare itemset
intension, *see* intent
intent, 9, 11, 14
interest, *see* lift
interestingness measures, 72, 99
item, 15
itemset, 15, 29

join, 12, 13

KDD, 1
knowledge representation, 5

largest itemset, 31, 85
lattice, 9, 14
lattice-based classification, 13
levelwise search, 16
lift, 17, 73, 100
lossless association rules, 37, 61, 68
lossless itemset representation, 65
lower bound, 9, 13
Luxenburger basis, 61, 80, 123

machine learning, 5
magyar rezümé, 156
maximal frequent itemset, 58, 65, 83
maximum support, 82, 93
meet, 12, 13
methodology, *see* data mining methodology
MIISR, 95
minimal non-redundant rules, 65, 68
minimal rare generator, 90, 96
minimal rare itemset, 32, 84
minimal zero generator, 84
minimum confidence, 17, 30
minimum item support, 94
minimum support, 16, 29, 30, 93
MRG association rules, 97
MSapriori, 94

negative association rule mining, 81
negative border, 32, 89
negative generator border, 91
non-derivable itemsets, 68
non-zero itemset, 84
novelty, 74
numerical data mining, 4

ontology, 2, 5, 24
order, 15
OWL, 24

partial order relation, 12
Pascal, 37
Pascal$^+$, 47, 49, 175
pattern, *see* itemset
pattern counting inference, 37, 39, 41
perfectly rare association rule, 93
perfectly rare itemset, 93
positive border, 89

potentially frequent candidate, 31
prefix tree, *see* trie
prefix-based equivalence class, 33
proper basis, 61, 80

quality measures, *see* interestingness measures

résumé étendu en français, 127
rare association rule, 93
rare item problem, 93, 96
rare itemset, 82
RDF(S), 25
reasoning, 5
relative support, 29
representative episode rules, 68
representative rules, 61, 68, 80, 123
RSAA, 94

satisfaction, 74
Semantic Web, 24
sequential pattern mining, 81
simple equivalence class, 40
sound association rules, 37, 61, 68
spatial association rule mining, 81
sporadic association rule, *see* rare association rule
STANISLAS cohort, 82, 96, 104, 110, 111
statistics, 4
strong association rule, *see* valid association rule
structural basis, 61, 80
subconcept, 15
subset function, 180
subsumed concept, 12
subsumer concept, 12
subsumer equivalence class, 41
subsumption check, 174
subsumption on itemsets, 174
subsumption relation on concepts, *see* partial order relation
subsumption relation on equivalence classes, 41
superconcept, 15
superset function, 182
supervised learning, 5
support, 16, 29, 30, 73
support count, 34
support count of 2-itemsets, 34, 43, 167, 189
supremum, 13
symbolic data mining, 4

text mining, 25
tidset, 29, 33
toy dataset, *see* example dataset
trie, 32, 179

unit element, 14
upper bound, 9, 13

valid association rule, 17, 30, 61
valid rare association rule, 93
vertical data layout, 34, 187

Web mining, 24
Weka, 2, 26, 106

XML, 25

Zart, 37, 68, 69
zero element, 14
zero equivalence class, 85
zero itemset, 84

Nancy-Université
Université
Henri Poincaré

Monsieur **SZATHMARY Laszlo**

DOCTORAT DE L'UNIVERSITE HENRI POINCARE, NANCY 1

en INFORMATIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER N° 1256

Nancy, le 8/11/06

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ, NANCY
LE PRÉSIDENT

[illegible]

J.P. FINANCE

Abstract

The main topic of this thesis is *knowledge discovery in databases* (KDD). More precisely, we have investigated two of the most important tasks of KDD today, namely itemset extraction and association rule generation. Throughout our work we have borne in mind that our goal is to find *interesting* association rules from various points of view: for efficient mining purposes, for minimizing the set of extracted rules and for finding intelligible (and easily interpretable) knowledge units. We have developed and adapted specific algorithms in order to achieve this goal.

The main contributions of this thesis are: **(1)** We have developed and adapted algorithms for finding minimal non-redundant association rules; **(2)** We have defined a new basis for association rules called Closed Rules; **(3)** We have investigated an important but relatively unexplored field of KDD namely the extraction of rare itemsets and rare association rules; **(4)** We have packaged our algorithms and a collection of other algorithms along with other auxiliary operations for KDD into a unified software toolkit called CORON.

Keywords: artificial intelligence, knowledge discovery in databases (KDD), data mining, symbolic data mining, frequent itemset extraction, frequent association rule generation, rare item problem, rare itemset extraction, rare association rule generation.

Résumé

Le sujet principal de cette thèse est *l'extraction de connaissances dans les bases de données* (ECBD). Plus précisément, nous avons étudié deux des plus importantes tâches d'ECBD actuelles, qui sont l'extraction de motifs et la génération de règles d'association. Tout au long de notre travail, notre objectif a été de trouver des règles d'associations *intéressantes* selon plusieurs points de vue : dans un but de fouille efficace, pour réduire au minimum l'ensemble des règles extraites et pour trouver des unités de connaissances intelligibles (et facilement interprétables). Pour atteindre ce but, nous avons développé et adapté des algorithmes spécifiques.

Les contributions principales de cette thèse sont : **(1)** nous avons développé et adapté des algorithmes pour trouver les règles d'association minimales non redondantes ; **(2)** nous avons défini une nouvelle base pour les règles d'associations appelées "règles fermées" ; **(3)** nous avons étudié un champ de l'ECBD important mais relativement peu étudié, à savoir l'extraction des motifs rares et des règles d'association rares ; **(4)** nous avons regroupé nos algorithmes et une collection d'autres algorithmes ainsi que d'autres opérations auxiliaires d'ECBD dans une boite à outil logicielle appelée CORON.

Mots-clés : intelligence artificielle, extraction de connaissances dans les bases de données (ECBD), fouille de données, fouille de données symbolique, extraction des motifs fréquents, génération des règles d'associations fréquentes, problèm des motifs rares, extraction des motifs rares, génération des règles d'associations rares.