No d'ordre 02 ISAL 0053 Année 2002

Thèse

Condensed Representations of Frequent Sets: Application to Descriptive Pattern Discovery

Présentée devant

L'Institut National des Sciences Appliquées de Lyon

Pour obtenir

Le grade de docteur

`Formation doctorale:` Extraction des Connaissances à partir des Données

`École doctorale:` Informatique et Information pour la Societé

par

Artur Bykowski

ingénieur INSA-Lyon

Soutenue le 21 octobre 2002 devant la Commission d'examen

Jury MM.

Directeur	Jean-François Boulicaut	Maître Conférences à l'INSA de Lyon
	Robert Laurini	Professeur à l'Université Claude Bernard Lyon 1 et directeur du LISI
	Christophe Rigotti	Maître Conférences à l'INSA de Lyon
Rapporteur	Nicolas Spyratos	Professeur à l'Université Paris-Sud Orsay et directeur de recherche au LRI
	Gerd Stumme	Chercheur à l'Université de Karlsruhe (Allemagne)
Rapporteur	Hannu Toivonen	Professeur à l'Université d'Helsinki (Finlande)

Cette thèse a été préparée au Laboratoire d'Ingénierie des Systèmes d'Information (LISI) de l'INSA de Lyon.

Abstract

Knowledge Discovery from Databases (KDD) is a relatively new research domain, which aims at making better use of large amounts of data collected nowadays. Interesting pattern discovery is a sub-domain of KDD.

This sub-domain has recently seen an impressive progress, due to an increasing pressure from owners of large data sets and to the response of scientists by numerous theoretical and practical results. The most of data sets addressed in the beginning of the surge were sales data and the interesting patterns were in form of association rules. Very efficient solutions to this practical problem were elaborated, the root of them was the so-called **APRIORI** algorithm.

Then, the owners of other types of data wondered if these basic methods could help them. Unfortunately, their data and/or the form of interesting patterns were different. Often, these applications could not take advantage of **APRIORI**. The research following the elaboration of the basic solution addressed the important application areas, where the basic solution could not be used.

Me and my colleagues addressed the problems with mining frequent patterns in different applicative contexts, especially the problems related to the large number of interesting frequent patterns present in data that are not similar to the sales data.

We obtained a significant improvement of the performances, due above all to the use of original condensed representations of frequent patterns. This is complementary to an optimization of the evaluation function — the approach engaged by most of the recent enhancements over the **APRIORI** algorithm. Our methods mine a collection of patterns that may be quite different from the target pattern collection, and hopefully much more efficient to be mined in some types of data. Moreover, that different pattern collection must allow a subsequent "regeneration" of the target collection in a very efficient manner, in our case without access to the original data set. The theoretical framework that enfolds such methods is proposed. Since the intermediate representation will be often smaller than the target collection, we call it a condensed representation.

The use of condensed representations is relatively novel in the field. However, as we will show in the dissertation, a few isolated preceding solutions make implicit use of condensed representations. The presented research proposes an explicit unified structure for the most significant past results.

Then new major condensed representations of simple frequent patterns are proposed, the algorithms to mine them and to derive the target pattern collections. We show the practical advantages of the proposed condensed representations over the past methods, and provide an abstract view of the proposed representations in the unified structure for condensed representations.

keywords: data mining, descriptive statements, association rules, condensed representations

Contents

Notations

Typical meaning of letters

A, B — items

X, Y, Z — itemsets

E — itemset enumeration tree

G_X, S, T — generalized itemsets

$S_X, S_A, S_T, \ldots$ — supports of itemsets or of generalized itemsets

σ, σ', γ — relative or absolute support thresholds

ρ — confidence threshold

δ — an absolute number of rows

R — binary database schema (set of items)

$r, r_1, r', \ldots$ — data sets (often binary databases)

Other Symbols

$|I|$ — cardinality of the set I, used also with multisets

$\cup, \cap$ — resp. set union and intersection, not used with multisets

$\vee, \wedge, \neg$ — resp. logical disjunction, conjunction and negation

$\bigwedge_{i \in I}$ *condition* — aggregate logical conjunction over all $i \in I$, $\forall$ not used

$\bigvee_{i \in I}$ *condition* — aggregate logical disjunction over all $i \in I$, $\exists$ not used

Chapter 1

Introduction

1.1 Knowledge Discovery in Databases — the Domain

In general, observing and analyzing often precedes the discovery of something new. This is true in several domains, and the ways of observing and analyzing are very diverse.
The development of electronic equipments and of computers suddenly increased the abilities for observation and data storage, and in the recent years multiplied by orders of magnitude the amount of actually stored facts. During the same period, the potential for processing the stored data has evolved, especially for tasks that humans handle inefficiently. These capacities (to observe, to store data and to analyze them) are growing very fast and paved the way for employing data storage and analysis for new purposes — since we have data, or can collect them easily, and can analyze them, we could make some discoveries.
People can and actually do analyse presently much more data than in the past. For example few centuries ago, who would bother to measure precisely and systematically around the globe in the hope to have a return justifying the effort:

- the thickness of glaciers and of clouds,
- the temperature and height of the sea-level,
- the amount of ozone and other low-concentration gazes in the atmosphere,
- ... and other factors?

Nowadays, it is not simply possible, but it is actually accomplished with high spatial and temporal resolutions by a single satellite (Envisat, see, e.g., [European Space Agency 02]) addressing the question of the climate change on Earth. The average amount of data collected by this satellite is of about 50 GB/day and is expected to grow during its lifetime. These data are transmitted by Envisat and stored on the ground with goal to be analyzed jointly with other observations to find possible relationships between them.

This amount of data corresponds to the present-day (*A.D.* 2002) computer storage abilities, however a full in-depth analysis of such an amount of data requires large scale effort involving many people and employing various methods[1].

Envisat data are collected with the intention of analyzing them for a very wide range of applications in mind. But in other domains, the data are typically collected routinely to support some specific activities (e.g., plane reservation system). After their exploitation (relative to their principal use) they are often stored, because the storage has a marginal cost and there is a hope to reuse them in a productive, but different way in the future. There is a wealth of such data — they have origins in various domains.

When we speak about large data sets, we mean data sets, where several thousands or millions of individual observations are put together. The description of an observation stored in the data set will be called a *data point* in the following. Such a description is a collection of values for *attributes* (numerical, categorical, Boolean, *etc.*) of the observed event or object. We assume in the following that all data points are described in a uniform way, i.e., using the same attributes.

One of the most known examples of large data sets is the class of basket data, i.e., data about the products bought by each client in a single transaction (e.g., in a supermarket data are collected at the cashiers). Each data point is typically a collection of items from a basket, which we can present as a vector of numbers, each of them corresponding to an item, stating the quantity of this item in the basket (how many pieces or weight). Beside inventories of stores and checking cashiers money, these data have served to analyze the shopping behavior of customers.

Typically, the data collected routinely are a few orders of magnitude less voluminous than the ones mentioned above for Envisat, yet the current state-of-the-art methods for analyzing data have tractability problems. Moreover, in most cases they cannot produce valuable statements without human guidance.

Knowledge Discovery from Databases (KDD) intends to analyze large data sets with human guidance and computer assistance, with the goal of eliciting valuable statements. Of special interest are the statements that are scattered, not simple facts stored in the data set, i.e., shallow knowledge. For example, the observation that a client bought milk, but did not buy beer is a simple fact stored in data. Finding such observations requires only to read the stored data. On the other hand, the observation that *every* client buying milk does not buy beer is scattered over the data set — the property concerns all data points. Neither KDD is interested in facts already known by experts, at least approximately (e.g., simple statistics).

Additionally, KDD addresses the problem of methods from the data analysis domain that do not work for large volumes of data. The computationally extensive techniques used in

[1] In [European Space Agency 02] one can read that two months before the launch of the satellite "[...] over 700 [European scientists] have applied to work with the information that will flow from Envisat".

KDD will be called *data mining* techniques.
Novelty, clarity, precision and usefulness are the key attributes of the elicited statements. Pragmatically, an interesting statement becomes a piece of knowledge (and thus is interesting to be discovered) if some adding-value action may be derived from it in a particular context (business, research, *etc.*). Since the data sets concerned by KDD are large, a good scalability with respect to the amount of data points and of output statements is a key requirement for methods.
KDD is an active research domain, and it is positioned at the intersection of database management, data analysis, statistics, machine learning and discovery science.
A KDD project is a joint effort between the KDD experts and the experts of the data domain. The former are needed, because most of the KDD techniques are complex and/or limited in applicability. The others, because an effective discovery process requires a constant evaluation of the practical value of the generated results. Indeed, data mining techniques provide properties that are observed in the data but only human experts can decide whether some of them constitute a valuable piece of knowledge in their domain.
Suitable application domains of KDD range from specific breakdown prevention in a particular system to general problems as in fundamental social or natural sciences. Prototypical application of KDD is when we have relatively little understanding about a complex system, but we suspect that, having enough data about it, some interesting relationships or simplifications can be found. Moreover, if collecting data about it is not necessary because we have them as a byproduct of another activity, the investment is smaller, but the outcomes may be important.

We decompose the KDD tasks into:

- collecting data
- preparation of data for mining (preprocessing),
- rough knowledge extraction (data mining),
- identifying the most interesting conclusions (postprocessing).

Collecting data, as previously said, may be out of control of the KDD process. The remaining part of the process is fundamentally iterative and each part can be repeated until the results satisfy the end-users (i.e. some useful knowledge has been elicited).
As mentioned before, the data set subject to KDD are usually collected without a particular technique in mind. This is either because this is a secondary use of these data, or because the data owner stores all observed details in order to try different techniques, and hopes to gain some useful elicitation involving some of stored data points.
Such data have often weaknesses such that missing values ($NULL$ values in database terminology), attributes not relevant to the domain, different values designating the same entity, *etc.*

NULL values are part of the every day practice in databases. Attributes not relevant to the domain, may be the source of problems, since they increase the load of data mining. The existence of such attributes may be accidental, or a reminiscence of the database evolution.
Since KDD process relies on data, the quality of data is crucial. Assessing the quality of data implies that the KDD expert must understand the application domain and collect some background knowledge. To this aim, the participation of the application domain expert is often necessary.
The most of the weaknesses of the data need to be remedied, depending on the problem with data and on the particular techniques employed as data mining. E.g., some techniques are resistant to the presence of a small amount of NULL values, but most of them are not. Feature selection, i.e., selecting only some aspects of data for study, maybe independently of the others, is done at this stage.
Additionally, we often want to take advantage of some background knowledge to improve the quality of mined knowledge, or at least to avoid rediscovery of already known facts. Finally, we must put the data under the form suitable for the data mining software. Typically, resolving all these problems requires procedures specific to the data set.
Then, data mining tools can be applied, producing various forms of rough knowledge, such as clusters or valid patterns. These tools basically try to give a less complex description of a system than the data set itself. Simplifying the data set is a fundamental task of data mining tools and permits to humans to make some abstraction of the data, and thus of the studied system itself. There is a lot of different ways one can present the data and the properties of the studied system in a simplified way. We are going to give the flavor of various kinds of rough knowledge in Section 1.2.
Assessing an abstraction of the data is often done as a postprocessing step. During this ultimate step in the discovery process the following aspects must be considered:

- the validity of the extracted properties for the studied system (as opposed to the validity in the data, in which the properties can be assessed by computers),
- the novelty of the properties,
- the usefulness of the properties in the potential applications.

The first question requires a domain expert to correct the errors due to the use of collected data. Indeed, the data in most cases describe a system in an inherently limited way. Moreover, these data undergo a several-step processing, and may even lose the representativeness of the studied system. The domain expert can assess directly the validity of the extracted rough knowledge, or investigate by independent means, if the potential applications are worth, but a higher confidence that the extracted properties are not erronous w.r.t. the studied system is required. On the other hand, assessing the extracted rough

knowledge does not mean to make sure it is in agreement with the expert's prior knowledge. An important possibility the domain expert must admit is that his/her background knowledge is subject to verification and adjustment, i.e., may be incorrect or incomplete. Often however, the expert will see properties already known. Indeed, data mining tools rediscover what is well-known, because assessing the novelty of a property by the computer requires we provide it with an exhaustive prior knowledge, which is virtually impossible. Without stating the prior knowledge, the most "promising" properties will include trivial ones, and the expert must filter them out.
Presenting the results in an easily readable form should also be included in the postprocessing.

1.2 Data Mining Tasks and Tools

We do not intend to list exhaustively the data mining tasks and tools, either to give details such as stating the hypotheses for applicability, but rather to give flavor of variousness.

1.2.1 Outcome of Data Mining Tasks

Well-known forms of rough knowledge can be grouped in two families — into description and prediction statements, depending on the function we assign to them. We will contrast these notions in the end of this section. Below, we briefly describe high-level forms of the outcome (which can be considered as rough knowledge) that data mining tools produce.
The family of descriptive forms of rough knowledge includes: descriptions by means of clusters, collections of interesting patterns, valid equations, sets of outliers, ...
A description by means of clusters is a partitioning of all data points into *clusters* such that there is a relatively few clusters, and data points that are similar are in a same cluster, dissimilar ones are in different clusters. A small number of outliers can appear, i.e. isolated data points (data points with no similar ones). These, very-different-than-the-others data points are also considered as interesting source of discoveries, especially if they correspond to observations breaking the laws (e.g., natural laws).
Interesting patterns are statements over a few attributes (of data points) in simple expressions. In spite of selecting among all of patterns only the most promising ones (potentially the most interesting), in general a great number of patterns are produced (say several thousands).
When we describe all data points with a single complex equation, with a number of parameters, we usually say we are describing data using a dependency model. The description is then such an equation with concrete numbers instead of parameters (an instance of the model).
The second family of forms of knowledge is the family of predictive statements, such as classification trees, quantitative laws, *etc.*

A classification tree is a complex several-case equation, with a categorical (small domain) variable on the left-hand side (*target variable*) and *explanatory variables* on the right-hand side. It permits to assign the value of this target variable, given other characteristics of the data point, presumably making rarely (or never) a mistake in the studied system.
A quantitative law is an equation between a quantitative *target variable* and an expression involving *explanatory variables*. It permits to assign the value of this target variable, given other characteristics of the data point, presumably making a small mistake (or none) on the target variable in the studied system.
As shown above, the outcomes of different data mining tasks take different forms. The domain is vast; in this thesis we focus on one form, notably on patterns and their discovery.

1.2.2 Evaluation Function

A task of mining rough knowledge is inherently linked to an evaluation function. An evaluation function states how good the descriptive or predictive statement is, given the data set.
In case the rough knowledge consists of a single "best" expression, this function will guide the algorithms to find it according to the data.
In case the rough knowledge consists of *a priori* unspecified number of expressions, the objective of this function is to let only the promising ones be output.
Here, we speak only about criteria that can be expressed explicitly in the form of computer program, and can be computed relatively fast. The later requirement is very important, given that we apply the function to large collections of data. It sometimes happens that we construct the evaluation function with the first objective of being fast in execution, and only as secondary we care about the reflecting exactly what interesting statements are. We suppose the performance gain will overwhelm the (presumably small) quality loss, and we expect that the later phases of the KDD process deal with the eventual problems.
Different evaluation functions score higher different statements, even if the same data set is used. This is generally a desired behavior, because in different contexts different statements are novel or useful. E.g., the evaluation function may differently penalize the complexity of the considered expressions depending on how simple they must be w.r.t. their intended use.

1.2.3 Data Mining Tools

Data mining tools are pieces of software capable of computing the forms of knowledge, such as presented above, in an efficient way (on rather large data sets). Ideally, they find the goal as specified by a condition on the evaluation function (e.g., the set of parameters of a model maximizing the evaluation function). However sometimes, the end-user is satisfied with an approximate or incomplete result, provided we specify how different

the result from the goal is. A typical example is guaranteeing that the produced result under the form of "best" expression is a locally best expression (a local maximum of the evaluation function), but not necessarily globally.

Clustering methods (tools providing a description of data set by means of clusters) are numerous, and differ in the used background knowledge or in the hypotheses they require to hold. Most of clustering methods is foreseen for a single family of evaluation functions. **K-means** [MacQueen 67], **hierarchical clustering** (a recent algorithm of this class may be found in [Guha 98]) are such algorithms (the latter relies on local evaluation functions to split or merge clusters). Some clustering methods produce, beside the clusters, perspicuous criteria for discrimination between these clusters.

Efficient pattern extraction tools exist for a rather limited number of pattern classes. Association rules, functional dependencies and very few other classes can be mined efficiently in some cases, but we know lot of classes of patterns for which, at the present time, there is no efficient extraction tools. Since this category of rough knowledge is the main objective of the thesis, most relevant existing tools will be described in the following chapter.

Fitting a model to the data, i.e. instantiating a model, is sometimes possible analytically, so we can produce the exact formulae for computing parameters of the model given the data set, but this happens for relatively simple models and evaluation functions. Such methods include **principal component analysis** and **parametric density models** (both well-known in statistics). Otherwise, we rely on methods searching iteratively within the space parameters the values that optimize evaluation function.

The tools for mining classification trees consider that data points belong to different classes, one categorical variable being the guide (*target attribute*, also called *class variable*). Then, they try to set a simple expression that divides data points into one partition that is as homogenous as possible (following target attribute) and the remaining data points as the second partition. The tools for mining classification trees differ especially in the form of the expressions that can split a partition into two smaller ones. Then, to each of the partitions we apply recursively the procedure, unless it is uniform w.r.t. the target attribute (possibly with some quantitative criterion for that). Thus, we obtain a way to find out the value of the target attribute given the values of other attributes. Supposing, that the system producing observations did not change since the data were collected, this permits us to predict the value of the target attribute for observations not stored in the data set.

Another interesting family of classification tools are the algorithms based on **Support Vector Machines** [Vapnik 95], which try to split observations belonging to exactly two partitions, to match the classes as well as possible using a single linear or quadratic expression.

Fitting a model to the data with Bayesian statistics help to make a formula more predictive

than descriptive. Bayesian approaches can explicitly "model" our convictions and facts that the data only partially describe the system and that some uncertainties raising from the data set itself have a quantifiable influence of the conclusions we can draw from data about the studied system.
Outlier detection tools consider that some of the tools presented above produced properties that describe most of the data points. The remaining ones, the most difficult to fit, will be considered as outliers. Outlier detection tools differ from each other in what method is employed to partition the data.

1.2.4 Terminology

Forms of rough knowledge can be grouped in two families — into description and prediction statements, depending on the function it has. Sometimes to a same expression we may assign two different functions, due to different interpretations.
Description statements are facts that are verifiable in the data. For example, in a data set describing cars by their attributes, a description statement can be "if a car is black, it has 4 doors".
Prediction statements are facts that have certain chances to be true outside of the data set, under the hypothesis that the mechanism that produced the data set is somehow related to the one to which we want to apply the prediction. For example, "if a car is black, it has 4 doors" can also be a prediction statement.
If we try to contrast these notions, we can see that the techniques mining some predictive forms of knowledge are in trouble when the studied system changes — the purpose of the knowledge explicitly being the prediction, the quality of the knowledge is generally lost.
The techniques mining descriptions simply report some potentially useful simplification about the provided data. This kind of information is typically used for decision support.

1.3 Applications of Descriptive Patterns

In the mid-1990s, researchers from KDD domain came up with a method to extract and analyze numerous associations within basket data sets, which are of particular interest of marketing sciences.
A so-called association rule is an expression of the form:

$$\{P_1, P_2, \ldots, P_n\} \Rightarrow \{R_1, \ldots, R_k\} \quad [s, c]$$

where $\{P_1, \ldots, P_n\}$ and $\{R_1, \ldots, R_k\} \neq \emptyset$ are disjoint sets of product identifiers (product identifiers will be further called items). As evaluation functions, we consider s, c — two values in $[0, 1]$, called respectively *support* and *confidence*. The meaning of this rule is that if the confidence is high a recorded transaction (a data point) with all products P_i present

in the basket often contain also all products R_i. The observed conditional probability (observed in the data set) of the fact that a transaction that contains all products P_i will also contain all products R_i is given by the confidence c. The observed probability that a transaction has all P_i and all R_i is given by the support s[2]. Association rule mining task consists in extracting from a data set all such statements that verify $s \geq min_s \wedge c \geq min_c$. min_s, min_c are fixed by a user, and we will refer to them respectively as *support threshold* and *confidence threshold*.

Several basket data sets could have been analyzed with an efficient technique called **APRIORI** [Agrawal 94, Mannila 94], extracting such association rules. We will present this basic technique in Chapter 2.

Mining association rules has applications in marketing (analyzing the shopping behavior, which is the prototypical application of association rules). The data sets of other domains did not have some particularities as basket data sets did have. Moreover, the other domains aimed at different class of patterns. As stated in [Hand 01], beyond the exploratory data analysis, successful primary applications of association rules are not numerous.

The application to other domains required further research, and the most successful extensions to the basic technique for mining association rules were techniques for mining so-called episode rules. Episode rules are similar in form to association rules, the difference being that they concern ordered data such as text or timed event sequences. The expression of episode rules reflects the ordered nature of data and possible associations between data points close to each other following the order. Applications of episode rules include constructing text summaries, predicting system breakdowns and Web usage mining[3] (also called clickstream mining).

Descriptive patterns, such as association rules, are not trivial to use. Mining them typically requires sophisticated tools that can cope with large search space and data set sizes. Moreover, as said previously, a knowledge discovery project requires a KDD expert to participate, because combinatorial aspects of data mining tools are often not well understood by the end-users[4].

For example, a naïve belief in pattern mining is to think that enumerating and evaluating all possible patterns is a right way to find the interesting ones. In real-life pattern spaces, this method would be undoubtedly intractable.

Let us take the example of association rules. Suppose we have mere 200 products in a supermarket. Then the number of possible association rules (ignoring the thresholds of support and confidence) is $3^{200} - 2^{200}$, i.e. around 10^{95}. This is more than the number of

[2]The term "probability observed in the data set" refers to the ratio of data points stored in the data set having a property over all data points stored in the data set.

[3]Finding typical patterns of navigation in documents structured with hypertext.

[4]Nonetheless, it is anticipated by the writer that with the development of the KDD domain the KDD tools will be used more and more by the end-users, and KDD projects will be assisted by computer system administrators, rather than KDD experts, same way as database management systems are nowadays.

atoms in the Universe, making it hard to imagine that some computers could enumerate or store them exhaustively even in a far future.
Not only good tools are needed, but even very advanced tools need to be used appropriately. Setting too low support threshold or too rich pattern language often leads to intractable computations even with sophisticated tools. On the other hand, too high support threshold or inadequate pattern language may produce valid patterns, but none of them may be, e.g., novel. We will come back to this question in Section 2.1.5 illustrating potential problems using a real-life example.
We address the question of extracting collections of descriptive patterns when interesting patterns are too numerous. The following section describes briefly our approach.

1.4 Informal Presentation of the Main Results — Condensed Representations of Frequent Sets

In this thesis, we propose a framework of condensed representations of patterns and study its two specific instances on patterns called frequent sets. These two instances reflect our idea to remove some kinds of redundancy in pattern collections.
Below, we introduce the collection of patterns to which we will apply the idea of condensed representations, i.e., frequent sets. Just after, we give an informal description of the condensed representations of frequent sets that we will investigate in details in this thesis.

1.4.1 Frequent Sets

In the context of basket analysis itemsets are expressions of the form:

$$\{P_1, P_2, \ldots, P_n\} \quad [s]$$

where $\{P_1, \ldots, P_n\}$ is a set of product identifiers (items), $n \geq 0$ and $s \in [0, 1]$ is called *support* of the itemset $\{P_1, \ldots, P_n\}$. The observed probability that a transaction includes all items P_i is given by the support s. Frequent set mining task consists in extracting from a data set all such statements that verify $s \geq min_s$ (min_s is fixed by a user).
The close relationship between frequent sets and association rules, which were defined in the previous section, will be given in Chapter 2. Here, we mention that association rules with a support threshold min_s can be computed from frequent sets mined with the same support threshold. In fact, various forms of knowledge that can be computed from frequent sets.
The main result of the thesis concerns alternative, condensed representations of frequent sets. The goal of condensed representations is to require less frequent sets in practical data sets and to be able to efficiently derive the forms of knowledge mentioned above. To

remedy the problem of the existence of too many interesting patterns holding in practical data sets, we propose an approach by condensed representations.
First, in Chapter 2, we shortly introduce frequent sets as prototypical patterns. We will show in details how frequent set mining is accomplished, with the state-of-the-art methods. We give an emphasis on the performance issues. We show that computing the evaluation function of itemsets, applied in practical data sets, is linearily dependent on the data set size (i.e., the evaluation function has a large cost when working with large data sets). The large cost of testing a pattern combined with an exponential growth of the number of patterns leads easily to too heavy or untractable tasks. The basic idea is that if the evaluation cost of a pattern that belongs to a condensed representation is about the same as of an ordinary pattern, we can cut the mining cost along with the reduction of the size of the collection of patterns. Besides the performance gains, the semantics of these patterns may be interesting in some contexts.
An original framework of condensed representations is provided in Section 3.1.
In Sections 3.2 and 3.3, we propose original condensed representations of the full collection of frequent sets. They may be used to represent all frequent sets without or with a small loss to the overall knowledge discovery process.
Two original representations will be based respectively on so-called frequent δ-free sets and frequent disjunction-free sets. We provide the necessary toolkit: important theoretical results, algorithms to mine them, experimental validation, and so forth.
Let us informally introduce these concepts.

1.4.2 Frequent δ-Free Sets

X, a frequent set, is a frequent δ-free set if and only if any 2 disjoint subsets $Y, Z \subseteq X$ are not related by any very strong positive correlation. We will examine the strenght of such a correlation using the number of exceptions (data points) of the rule $Y \Rightarrow Z$ (since every rules with $Z = \emptyset$ are trivially without exceptions, we exclude this possibility from the examination). Every time a data point has not all items from Z, while having all items from Y, we count 1 exception. X will be frequent δ-free set if for every such Y, Z there are more than δ exceptions.
We will show that if at least one such a rule has less than δ exceptions, the support of X can be approximated by the support of one of its subsets, and thus (in most of cases) outputting X may be omitted. This way, a lot of frequent sets may be omitted, and important gains may be realized.
A question of an eventual error on support will be addressed and tight error bounds (proportional to δ) will be provided.
The publications related to the development of this condensed representation include [Boulicaut 00a, Boulicaut 00c, Boulicaut 03].

1.4.3 Frequent Disjunction-Free Sets

The notion of frequent disjunction-free set is based on a different kind of relationships between supports of itemsets. When for some $A, B \in X$ (A, B are items) the exceptions of rules $X \setminus \{A\} \Rightarrow \{A\}$ and $X \setminus \{B\} \Rightarrow \{B\}$ hold on *disjoint* data points, the support of X may be computed exactly using the supports of some subsets of X. Otherwise, we say that X is disjunction-free.
We will show that the practical amount of frequent sets that can be "regenerated" from the others can exceed 90%, or even more, thus leading to important pruning and the associated gains.
The basic idea of this condensed representation has been presented in [Bykowski 01]. A completed study can be found in [Bykowski 03].

1.4.4 Organization of the Thesis

In Chapter 2, we present a state-of-the-art patterns and focus on methods for computing frequent sets.
In Chapter 3, we propose the condensed representation framework. We give a synthetic view on the state-of-the-art techniques described in Chapter 2, which are assimilable to that framework (of condensed representations). Then, we define formally both condensed representations introduced informally above. We show that they are more interesting from the performance point of view than the preceding results assimilable to the framework of condensed representations.
Since frequent sets can be used to mine different forms of rough knowledge, in Chapter 4, we detail how frequent sets and other closely related patterns may be efficiently obtained from the condensed representations described in Chapter 3.
Chapter 5 shortly concludes and presents directions of future works.

Chapter 2

State of the Art

In this chapter, we present some existing results relevant to this thesis. The contents of the chapter presents an original, not published elsewhere, viewpoint on past studies.
We start with more formal presentation of pattern mining, with a focus on patterns that have been investigated in the literature. At that point, we will focus on frequent sets, and introduce formally convenient notations for them. Then, we describe the techniques for mining frequent sets. Most relevant families of techniques include:

- levelwise: **APRIORI** [Agrawal 94, Mannila 94] (the basic successful technique, foreseen for basket data sets), and derivatives, e.g., **DIC** [Brin 97a] (optimizing the support counting procedure),
- depth-first: **FP-Growth** [Han 00], **Tree-Projection** [Agarwal 99, Agarwal 01],
- border mining: **Max-Miner** [Bayardo, Jr. 98], **Pincer Search** [Lin 98],
- guess and correct: **sampling** [Toivonen 96],
- mining closed frequent sets: **CLOSE** [Pasquier 99c] (levelwise), **CLOSET** [Pei 00] (depth-first).

At least one technique of each family will be detailed in this chapter.
At the end of this chapter, we will give the direction of research that we will exploit.

2.1 Descriptive Patterns

2.1.1 Well-known Examples of Descriptive Patterns

As presented in Chapter 1, models and patterns are different directions of expressing diffused properties of data. No sharp boundary exists and we tried to contrast these notions in Section 1.2.4.
Below, we briefly present the patterns that have been largely investigated in the literature. It is meaningless to speak about the form of the expressions alone, without giving the

semantics of the expression and stating what kind of conditions they must satisfy. E.g., the way we introduced association rules in Section 1.3 makes immediately reference to the support and confidence evaluation functions. They are not the only evaluation functions for these patterns (a single "best" evaluation function does not exist, as suggested in the introduction, because different contexts make a pattern unequally "interesting"). We are going to mention few alternatives in Section 2.1.4. Not only different evaluation functions have been proposed for association rules, but the term "association rules" has been used by different authors either for the form of the expression or for the collection of such expressions satisfying support and confidence criteria in a data set. The distinction is in our feeling important.

Association rules have been proposed in [Agrawal 93] as interesting analytical tool for transactional data sets. Closely related to association rules are *frequent sets.*

In order to extract possible additional associations between data points in ordered data, Agrawal *et al.* proposed in [Agrawal 95] *large sequential patterns* and Mannila *et al.* proposed in [Mannila 95] *frequent episodes.* They are both analogous to frequent sets in the sense that they can produce rules similar to association rules, but dissimilar in taking additionally into account that the data points are ordered. The resulting classes of patterns require an ordering, at least partial, of data points that match it. We present one class of such patterns, namely frequent episodes.

Episode rules, which follow frequent episodes in this presentation, can be derived from frequent episodes in a similar way that we can derive association rules from frequent sets.

Finally we will present *functional dependencies* and *inclusion dependencies.* We tend to put these expressions in the category of patterns, because in our opinion, they have more characteristics of patterns than of models. Nonetheless, they have characteristics of both.

Now, let us go one-by-one with the patterns investigated in the literature. We give each pattern class together with its original evaluation function.

Frequent Sets

Frequent sets for transactional data sets have been proposed in [Agrawal 93] as an intermediate pattern for mining association rules. We already introduced frequent sets for transactional data in Section 1.4. Let us now see an example.

Example 1. *If we consider the baskets represented in Table 2.1 and a support threshold value of* $2/8$*, then all frequent sets and their corresponding supports are:*

$$\emptyset \ [8/8], \quad \{A\} \ [4/8], \quad \{A,B\} \ [2/8], \quad \{C\} \ [5/8], \quad \{A,D\} \ [2/8],$$
$$\{B\} \ [4/8], \quad \{B,C\} \ [4/8], \quad \{A,C\} \ [2/8], \quad \{D\} \ [4/8], \quad \{A,B,C\} \ [2/8].$$

Frequent sets for data sets based on categorical attributes (attributes with few valuations) can be generalized in the following way. Itemset expression is of the form:

$$A_1 = a_1 \wedge A_2 = a_2 \wedge \ldots \wedge A_n = a_n \ \ [s]$$

A	B	C	D
x	x	x	
	x	x	x
x			x
			x
		x	
x			x
	x	x	
x	x	x	

Table 2.1: Baskets of customers

where $A_1, \ldots, A_n$ are n attributes, $n \geq 0$, each a_i is a value in the domain corresponding to A_i, and $s \in [0, 1]$ is a value of *support* (the evaluation function, not part of the pattern). Analogously to frequent sets for basket data, s is the observed probability that the data point satisfies the conjunction.

Frequent sets w.r.t. a data set are all such statements that verify $s \geq min_s$ (min_s is fixed by a user).

Some techniques have been proposed for data sets that contain several valuations of attributes (e.g., real-valued attributes). They tend to map the domains of these attributes into categorical domains, i.e., domains with few distinct values. For example, the real-valued attribute *age* (for records about people) may be mapped into categorical attribute taking values $age0\_20, age20\_40, age40\_60, age60\_80, age80+$ (mapping for values of *age* is suggested here by the names of categorical constants).

Rarely, but possibly, an attribute may be mapped into more than 1 categorical variable. For example, day_of_year may be mapped into $week\_of\_year$ and day_of_week to be able to detect associations particular to a day of the week (e.g., Sunday patterns) or a particular week (e.g., Easter week patterns).

Association Rules

Association rules for transactional data sets have already been introduced in Sections 1.3.

Example 2. *With the example data set depicted on Table 2.1, we may find the following association rule and its scores:*

$$\{C\} \Rightarrow \{D, A\} \quad [0/8, 0/5]$$

The collection of all association rules for the support and confidence thresholds respectively of 2/8 and 1/2 is:

$$\begin{array}{lll}
\emptyset \Rightarrow \{A\} \ [4/8, 4/8], & \{B\} \Rightarrow \{A\} \ [2/8, 2/4], & \{B, A\} \Rightarrow \{C\} \ [2/8, 2/2], \\
\{A\} \Rightarrow \{C\} \ [2/8, 2/4], & \{D\} \Rightarrow \{A\} \ [2/8, 2/4], & \{A\} \Rightarrow \{B, C\} \ [2/8, 2/4], \\
\emptyset \Rightarrow \{B\} \ [4/8, 4/8], & \emptyset \Rightarrow \{B, C\} \ [4/8, 4/8], & \{B\} \Rightarrow \{A, C\} \ [2/8, 2/4], \\
\{C\} \Rightarrow \{B\} \ [4/8, 4/5], & \emptyset \Rightarrow \{C\} \ [5/8, 5/8], & \{C, A\} \Rightarrow \{B\} \ [2/8, 2/2], \\
\{A\} \Rightarrow \{D\} \ [2/8, 2/4], & \{B\} \Rightarrow \{C\} \ [4/8, 4/4], & \{C, B\} \Rightarrow \{A\} \ [2/8, 2/4], \\
\emptyset \Rightarrow \{D\} \ [4/8, 4/8], & \{A\} \Rightarrow \{B\} \ [2/8, 2/4]. &
\end{array}$$

For data sets based on attributes with few valuations, association rules are generalized as follows. Such an association rule is an expression of the form:

$$A_1 = a_1 \wedge A_2 = a_2 \wedge \ldots \wedge A_n = a_n \Rightarrow B_1 = b_1 \wedge \ldots \wedge B_k = b_k \ \ [s, c]$$

where $\{A_1, \ldots, A_n\}, \{B_1, \ldots, B_k\}$ are disjoint sets of attributes, $n \geq 0$, $k > 0$, each a_i (resp. b_i) is a value in the domain corresponding to A_i (resp. B_i), and s, c are two values in $[0, 1]$, respectively *support* and *confidence*.

The meaning of this rule when the value of the confidence is high is following. When all attributes A_i of a data point are valued to the corresponding values a_i, the attributes B_i of that data point have often the corresponding values b_i. The observed conditional probability (observed in the data set) of the fact that a data point satisfying all $A_i = a_i$ equalities will satisfy also all $B_i = b_i$ is given by c (the confidence). The observed probability that a data point satisfies all $A_i = a_i$ and all $B_i = b_i$ is given by s (the support). As in case of basic association rules a user fixes min_s, min_c and the task consists in extracting from a data set all such statements that verify $s \geq min_s \wedge c \geq min_c$.

Further extensions of association rules include quantitative association rules [Srikant 96], association rules generalized for including hierarchies [Srikant 95], association rules generalized for arbitrary Boolean expressions [Mannila 96b].

Frequent Episodes

Frequent episodes [Mannila 95] are patterns similar to frequent sets, with the difference that they are designed for unidimensionally-ordered data, especially for temporal data. In this case, data are ordered using a dedicated attribute, designated in the following as *time_stamp*. This attribute may be implicit (e.g., data point number) or explicit (a specific attribute of data points with a total order defined on its values). We often use the term *event* instead of *data point*, to emphasize that there is a distinguished attribute for ordering.

Examples of implicit ordering include textual documents (when we consider for example words as events) and DNA base sequences (base symbols, C, G, T, A, arranged in a long string, are then events).

Explicit ordering may be based on attributes giving a precise coordinate, in time or in space, of an event.

An episode is a partially ordered collection of event constraints. An example of *arbitrary* episode could be C_1 *and* C_2 *both followed by* C_3 (note that there is no assumed order between C_1 and C_2, unlike between two other pairs). This episode imposes both: the kinds of events and their relative partial order of occurrence. It matches any sequence of events that contains an event satisfying the constraint C_1 and one satisfying the constraint C_2, in any order, but both followed by an event satisfying the constraint C_3 (order of events is given by their *time_stamps*).
Of special interest are:

- *serial* episodes, e.g., C_1 *then* C_2 *then* C_3 — the "matching" events must satisfy the individual constraints and their *time_stamps* must additionally follow the order corresponding to the episode (i.e., this order is defined totally),
- *parallel* episodes, e.g., C_1 *and* C_2 *and* C_3 — the "matching" events must satisfy individual constraints, but their *time_stamps* do not have to follow any order.

The language of episodes is often reduced in a particular extraction to a class of patterns respecting some conditions of their structure, e.g., to serial episodes only, to guide the tool in the discoveries.
We can formalize the pattern with a graph-like structure $\alpha = (V, \preceq, g)$, where V is a collection of nodes, $\preceq$ is an ordering relation between the nodes and g is a mapping assigning to a node from V the constraint it represents.

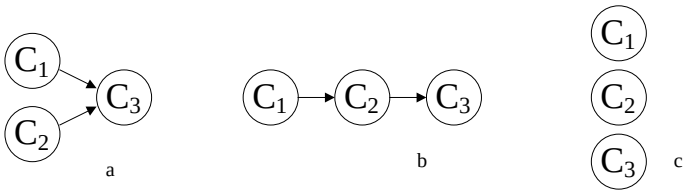

Figure 2.1: Examples of episodes: arbitrary (a), serial (b) and parallel (c).

We can conveniently visualize episodes using oriented graphs, with vertices denoting elements of V and the edges corresponding to the partial order $\preceq$. Figure 2.1 shows the graphs corresponding to the episodes given as examples so far (we show only the transitive reduction).
In [Mannila 95], for simplicity, the constraints on events are restricted to involve only one attribute, the same for all constraints, and the only allowed form of constraint is that this attribute and a constant are equal. That attribute is called an *event type*.
The evaluation function proposed in [Mannila 95] uses the the concept of *window*, through which we will match episodes with data. We define it after [Mannila 95]. A window is characterized by its *beginning* and *end*. Given a data set, the *events in such a window* (a subset of all events in the data set) is made of events whose *time_stamps* take the values in the half-open interval $[beginning, end)$.

Then an episode α *matches* the events in a window W if and only if for every node n of α we assign an event e in W satisfying $g(n)$ (recall that $g(n)$ is a constraint of an event), and the $time\_stamps$ in the resulting collection of events (corresponding to all nodes of α) respect the order $\preceq$.
The *support* of an episode is the ratio of windows in which the events match this episode. If this ratio is at least min_s (the support threshold, fixed by the user), then the episode is called *frequent*.
The population of windows is produced by "sliding" a window of width w (also fixed by the user) over an interval $[T_S, T_E]$, with the beginning of the first window overlapping with T_S, and the end of the last window overlapping with T_E. T_S and T_E are coordinates from the same dimension as $time\_stamps$. They should be fixed also by the user, and should be (respectively) not greater than the least $time\_stamp$ in the data set and greater than the greatest $time\_stamp$ in the data set.

Episode Rules

Rules obtained from the frequent episodes presented above have been formalized in [Mannila 97c]. To define the expression of such rules we need to define first the concept of *sub-episode* of an episode. A sub-episode of an episode $\alpha = (V, \preceq, g)$ is an episode $\alpha' = (V', \preceq', g')$ with:

- $V' \subseteq V$,
- for every node $n \in V'$, $g(n) \equiv g'(n)$ (each node of V' expresses the same constraint as in V),
- for all $n_1, n_2 \in V'$, $n_1 \preceq' n_2 \Rightarrow n_1 \preceq n_2$ (the pairs of nodes that are ordered in α' are ordered as well in α).

α' is a proper sub-episode of the episode α if additionally α is not a sub-episode of α'.
Episode rule is then an expression of the form:

$$\alpha' \Rightarrow \alpha \quad [s, c]$$

where α' is a proper sub-episode of the episode α. As evaluation functions (so not a part of the pattern) called respectively *support* and *confidence*. Thier respective values, denoted s, c, are defined as follows. s is the probability of observing events in a window that match α, and c is the conditional probability of observing events matching α in a window given the events of that window match α'.

An alternative definition of episode rules is presented in [Mannila 96a]. It allows better handling of multiple window sizes and a more intuitive semantics of evaluation functions.

Data Dependencies (Functional and Inclusion Dependencies)

The relational database theory gives quite a lot of attention to so-called functional and inclusion dependencies, which were informally introduced in [Codd 70] and formally studied in many textbooks like [Ullman 80, Abiteboul 95]. We shortly present them now, slightly adapting the description to our vocabulary.

Functional dependencies across a data set (a universal relation, in the original context) are expressions of the form:

$$\{A_1, A_2, \ldots, A_n\} \rightarrow \{B_1, \ldots, B_k\}$$

where $\{A_1, \ldots, A_n\}$ and $\{B_1, \ldots, B_k\}$ are disjoint sets of attributes, $n > 0$, $k > 0$. Let us consider a pair of data points r, q such that the value of attribute A_i is the same for both points, for all $i \in \{1, \ldots, n\}$. We say that this functional dependency holds in a data set if and only if for every such a pair of data points they also have the same corresponding values for all attributes $B_1, \ldots, B_k$.
Extensions of functional dependencies include approximate functional dependencies [Kivinen 92, Huhtala 99, Novelli 01].

Inclusion dependencies are expressions like:

$$R\ [<R_1, R_2, \ldots, R_n>] \subseteq S\ [<S_1, \ldots, S_n>]$$

where R and S are two data sets (two relations, in the original context), $<R_1, \ldots, R_n>$ and $<S_1, \ldots, S_n>$ are sequences of attributes without duplicates, respectively from R and S, with $n > 0$. We say that this inclusion dependency holds in the two data sets if and only if for every data point r from the data set R, there is a data point s from the data set S with the value of every attribute S_i ($i \in \{1, \ldots, n\}$) of s being equal to the value of attribute R_i of r.

Summary of Different Kinds of Patterns

A summary of the classes of patterns described above could lead to a better insight into the particularities of each pattern class.
One of the most important characteristics of the classes is the extent of a particular pattern, i.e., how many points are potentially concerned by a descriptive or predictive semantics of a pattern. Some patterns, called further *local patterns*, state directly properties of a limited number of data points and the other points are not concerned directly. If a pattern directly states a property of all data points, we call it *global*[1].

[1] Concerning the whole data set is one of the characteristics of models, but it is not a sufficient condition, in our opinion.

Pattern class	local	incorporating order
frequent sets	yes	no
association rules	yes	no
frequent episodes	yes	yes
episode rules	yes	yes
functional dependencies	no	no
inclusion dependencies	no	no

Table 2.2: Summary of characteristics of patterns presented in the section.

Locality of a pattern raises the question of its statistical significance. For example, if a property that has been extracted from a large data set concerns only one data point of that data set, how significant the property is for the studied system? Actually, quite significant if the property is of pure qualitative kind ("for all" or "there exists"), because it can disprove the former or prove the latter. But it is not very significant in the case of a quantitative law derived only from that single point. It could also point out erroneous data points (isolated points in large databases are often due to errors, and rarely to a real feature of the studied system).

As a simple substitute of the statistical significance of a local pattern one may use the observed probability of the fact that the pattern is supported by the data[2]. If the statistical significance of a property is monotonic with that probability, mining properties of a guaranteed minimum statistical significance may be reduced to applying a threshold to the observed probability of fact that a pattern holds. This leads to the family of *frequent* patterns, e.g., association rules, frequent sets, frequent episodes and episode rules.

Functional dependencies are global — each instance requires the corresponding property to hold for the entire set of data points, to be exact for all data point pairs. Inclusion dependencies, which involve 2 data sets, assign different roles to the data points of each of them. Nonetheless, an inclusion dependency directly concerns all data points of the left-hand-side data set, and in that sense it is global.

Another important feature is whether a pattern states a property resulting from an eventual ordering of data points. We call such patterns *order incorporating patterns*. This family includes frequent episodes and episode rules. The remainder of the patterns presented above are patterns not order-incorporating (note that inclusion dependencies make appear an explicit order of attributes, but it is not related to an ordering of points).

The discussed characteristics are summarized in Table 2.2.

2.1.2 Why Focus on Frequent Sets?

From now on, we will focus on frequent sets, and only occasionally mention other patterns. The reasons for choosing frequent sets include:

[2]We do not precise here what is the whole population corresponding to the discussed probability nor what it means that the pattern is "supported", because it depends on the semantics of a pattern.

- frequent sets have simple and precise meaning, they are an educational class of patterns — relatively easy to understand and to observe the disparity between the simplicity of the task definition and the practical difficulties related to mining them; the simplicity makes also research results easily assimilable by the other researchers,
- they are generic, i.e., some concepts and techniques defined for frequent sets can be adapted to other patterns,
- they share several properties with other patterns, but seem to be the most basic of them,
- different kinds of knowledge discovery tasks can make use of frequent sets (besides association rules, similarity measures [Das 98], clustering techniques [Das 98, Han 98], ...),
- frequent sets are used for exploratory analysis in a wide range of application domains.

Similar reasons must have lead other researchers to investigate this class of patterns, because most of recent publications in pattern mining focus on frequent sets.

2.1.3 Notations

In this section, we formalize the notions that we will use along this thesis. When possible, we follow the notational conventions and definitions of [Mannila 96b, Mannila 97b]. In particular, we use multisets to represent collections of rows and given such a multiset r, we write $t \in r$ to denote that a particular row t belongs to r.

Definition 1. (binary database) Let R be a finite set of symbols called *items*. A *row* is a subset of R. A *binary database* r *over* R is a multiset of rows.
Let $\prec$ be a linear order for the items from R.

A row in a binary database is typically the mapping of a single data point. For transactional data sets, the mapping is straightforward. As we suggested in the section devoted to frequent sets (see Section 2.1.1), for non-transactional data sets an item usually represents a particular value (resp. a range of values) of an attribute, and its inclusion in a row means that the attribute of the data point takes the corresponding value (resp. takes a value in the corresponding range).
In the following, we will rarely make reference to the real property represented by an item, so the results are applicable in both cases.

Definition 2. (itemset and its support) Let r be a database over R. Let X be any subset of R. Any such a set of items will be called an *itemset* and $|X|$ its *size*.

We note $\mathcal{M}(r, X) = \{t \in r : X \subseteq t\}$ the multiset of rows matched by the itemset X and $Sup(r, X) = |\mathcal{M}(r, X)|$ the *support* of X in r, i.e., the number of rows matched by X[3].

Definition 3. (frequent sets) Let σ be a support threshold (σ is an absolute number of rows), $Freq(r, \sigma) = \{X : X \subseteq R$ and Sup(r,X) $\geq \sigma\}$ is the set of all so-called *σ-frequent itemsets in r* (*frequent sets*, for short, when σ and r are clear in the context).

The following lemma is a fundamental property for the frequent itemset extraction. It states that the functions $\mathcal{M}$ and Sup are *anti-monotone* w.r.t. itemset inclusion. The anti-monotonicity of Sup has been demonstrated in [Agrawal 94, Mannila 94] as an efficient pruning criterion.

Lemma 1. *Let r be a binary database over R, and X, Y be sets of items such that $Y \subseteq X \subseteq R$. Then $\mathcal{M}(r, X) \subseteq \mathcal{M}(r, Y)$ and $Sup(r, X) \leq Sup(r, Y)$.*

Proof. Let t be any row in r that belongs to $\mathcal{M}(r, X)$. From Definition 2, a row t belongs to $\mathcal{M}(r, X)$ if and only if $X \subseteq t$. $Y \subseteq X \subseteq t$ implies $t \in \mathcal{M}(r, Y)$, and thus $\mathcal{M}(r, X) \subseteq \mathcal{M}(r, Y)$. Consequently, $Sup(r, X) \leq Sup(r, Y)$. □

Corollary (of Lemma 1) *Let X, Y be the same as in the lemma. If X is σ-frequent in r, Y is σ-frequent in r. If Y is not σ-frequent, so neither is X.*

For notational convenience, we also need the following specific definitions.
Let us adapt the concept of positive and negative border from [Mannila 97b] to our context.

Definition 4. (negative border) Let $\mathcal{C}$ be a downward-closed collection of itemsets[4] over R. The *negative border of $\mathcal{C}$* is noted $\mathcal{B}d^-(\mathcal{C})$ and is defined as follows: $\mathcal{B}d^-(\mathcal{C}) = \{X : X \subseteq R, X \notin \mathcal{C} \wedge \bigwedge_{Y \subset X} Y \in \mathcal{C}\}$.

Definition 5. (positive border) Let $\mathcal{C}$ be a downward-closed collection of itemsets over R. The *positive border of $\mathcal{C}$* is noted $\mathcal{B}d^+(\mathcal{C})$ and is defined as follows: $\mathcal{B}d^+(\mathcal{C}) = \{X : X \subseteq R, X \in \mathcal{C} \wedge \bigwedge_{Y \supset X} Y \notin \mathcal{C}\}$.

Definition 6. (collection of frequent sets and their supports) $FreqSup(r, \sigma)$ is the set of all pairs containing a frequent itemset and its support, i.e., $FreqSup(r, \sigma) = \{<X, Sup(r, X)> : X \subseteq R$ and $Sup(r, X) \geq \sigma\}$.

We wish to work sometimes on association rules in a formal way, so we define now the related concepts formally.

[3]Note that we use the absolute number of rows, unlike in previous sections, where we used the observed probability, i.e., the fraction of data points from the data set. Given a non-empty binary database, both are equivalent, but for all reasonings in this thesis, we prefer using the absolute supports.

[4]I.e., $\mathcal{C}$ is such a collection that $\bigwedge_{C \in \mathcal{C}} \bigwedge_{Y \subset C} Y \in \mathcal{C}$.

Definition 7. (association rule) Let X, Y, Z be three itemsets (subsets of R) such that $X \subset Z$, $Y \subseteq Z \setminus X$ and $Y \neq \emptyset$. An *association rule over* Z is an expression of the form $X \Rightarrow Y$. We call Y the *head* of the rule and X its *body*.
Let $Sup(r, X \Rightarrow Y) = Sup(r, X \cup Y)$ be the *support* and $Conf(r, X \Rightarrow Y) = \frac{Sup(r, X \cup Y)}{Sup(r, X)}$ be the *confidence* of rule $X \Rightarrow Y$ in the database r.

Definition 8. (frequent and confident association rules) Let r be a binary database. Let $\sigma \in (0, |r|]$ be a frequency threshold, $\rho \in [0, 1]$ a confidence threshold. $X \Rightarrow Y$ is a *σ-frequent and ρ-confident association rule in r* iff $Sup(r, X \Rightarrow Y) \geq \sigma$ and $Conf(r, X \Rightarrow Y) \geq \rho$. We use also the term *frequent and confident association rule*, when r, σ, ρ are clear from the context.

Definition 9. (collection of frequent and confident association rules with their supports and confidences) Let r, σ, ρ be as in Definition 8. Let Z be an itemset. We denote $RulesSuppConf(r, \sigma, \rho, Z)$ the collection $\{X \Rightarrow Y \ [Sup(r, X \Rightarrow Y), Conf(r, X \Rightarrow Y)] : X \Rightarrow Y$ *is a frequent and confident association rule over* Z *in* $r\}$.

To keep the presentation concise, we consider notational conventions to handle *negative form of items* as well as *accessing individual items of an itemset.*

Definition 10. (generalized item and full clause over an itemset) Let $R = \{A_1, A_2, \ldots, A_{|R|}\}$ be a set of symbols. The symbols in R will be called *positive* items, and for each positive item $A_i \in R$ we consider a *negative* item noted $\overline{A_i}$. $GenIt(R) = R \cup \{\overline{A_i} : A_i \in R\}$ is the set of generalized items based on R. We assume non-ambiguity of the symbols, i.e., that for every $A_i \in R, \overline{A_i} \notin R$.
Let $FullClauses(R)$ denote the set $\{A_1, \overline{A_1}\} \times \{A_2, \overline{A_2}\} \times \ldots \times \{A_{|R|}, \overline{A_{|R|}}\}$. A *full clause* over R is any member of the set $FullClauses(R)$.

In a full clause over R each item of R appears exactly once, either in a negative or a positive form. For example, the set $\{A, B, \overline{C}, D, \overline{E}\}$ is a full clause over the set $\{A, B, C, D, E\}$, whereas $\{B, \overline{C}, \overline{E}\}$ and $\{B, \overline{B}, \overline{C}, E, \overline{E}\}$ are not.

Definition 11. (generalized itemset) Let R be a set of symbols. A subset of a full clause over R is called a *generalized itemset based on* R. Formally, the set of all generalized itemsets is $\{T : T \subseteq F_R \wedge F_R \in FullClauses(R)\}$. Note that a full clause F_R over R is also a generalized itemset.
We denote the items appearing positively and negatively in a generalized itemset T as follows, $Pos(T) = \{A \in R : A \in T\}$ and $Neg(T) = \{A \in R : \overline{A} \in T\}$.

The semantics of these concepts are given by the following definition.

Definition 12. (generalized itemset support) Let r be a binary database over R and X be a generalized itemset based on R. Then we extend the functions $\mathcal{M}$ and Sup defined previously, as $\mathcal{M}(r, X) = \{t \in r : Pos(X) \subseteq t \wedge Neg(X) \cap t = \emptyset\}$ is the multiset of rows matched by X and the support of X in r is $Sup(r, X) = |\mathcal{M}(r, X)|$.

Intuitively, the rows matched by a generalized itemset X are the rows that contain all positive items in X but none of the items appearing under a negative form in X. For example, in Table 2.1 the generalized support of $\{\overline{A}, B, C\}$ is 2.
The matching and support functions on generalized itemsets are also anti-monotone w.r.t. the generalized itemset inclusion. This is stated by the following lemma.

Lemma 2. *Let r be a binary database over R and X, Y be generalized itemsets based on R. If $Y \subseteq X$ then $\mathcal{M}(r, X) \subseteq \mathcal{M}(r, Y)$ and $Sup(r, X) \leq Sup(r, Y)$.*

Proof. Let t be any row in $\mathcal{M}(r, X)$. $t \in \mathcal{M}(r, X)$ implies $Pos(X) \subseteq t$ and $Neg(X) \cap t = \emptyset$. Since $Y \subseteq X$ then $Pos(Y) \subseteq Pos(X)$ and $Neg(Y) \subseteq Neg(X)$, and then we have $Pos(Y) \subseteq t$ and $Neg(Y) \cap t = \emptyset$. Thus, $t \in \mathcal{M}(r, Y)$. The second conclusion is immediate. □

Definition 13. (index-based access of items from itemsets) We make use of a notation for accessing individually the items in the itemsets as follows. Let X be an itemset, $ordered\_list(X)$ be the list of all items in X sorted in ascending order according to the linear order $\prec$ for items. Then, $X[i]$ denotes the i^{th} element in the list $ordered\_list(X)$.

2.1.4 Main Uses of Frequent Sets

We mentioned in the previous section that frequent sets have been proposed as an intermediate pattern for mining association rules. Deriving association rules remains the main use of frequent sets, so we describe it first. Few other uses will be given at the end of this section.
Deriving all association rules from frequent sets is accomplished in the following way. Let σ, ρ be respectively the support and confidence thresholds for the association rules. Let FS be the collection of all σ-frequent itemsets and their corresponding supports in a data set r.
Algorithm 1 shows the pseudo code of a program performing the derivation of the corresponding association rules from FS. It considers all pairs X, Y such that X is frequent and Y is a proper subset of X (lines 2–3). The statement of line 4 finds in FS the support S_Y of Y. $Y \subset X$ (line 3), so S_X/S_Y (line 5) is the confidence of the rule $Y \Rightarrow X \setminus Y$. Observe that the algorithm does not make any use of σ. This threshold does not need to be checked, because the support of the rule $Y \Rightarrow X \setminus Y$ (lines 5 and 6) is, by definition, the support of the union of Y and of $X \setminus Y$, i.e., the support of X, which is known to be at least σ, because it is in FS (line 2).

Theorem 1. *Algorithm 1 finds all and only association rules with support and confidence values greater of equal to the corresponding thresholds.*

Proof. Let $U \Rightarrow V \quad [s, c]$ be an association rule with $s \geq \sigma$ and $c \geq \rho$. We show that $U \Rightarrow V \quad [s, c]$ will be in the output of that program.
The support of the itemset $U \cup V$, which is equal to the support of the rule $U \Rightarrow V$, equals to s. $s \geq \sigma$ implies that $U \cup V$ is a frequent set, so $<U \cup V, s>$ must be in FS and will be considered in line 2. Since $U \subset U \cup V$ (recall $V \neq \emptyset$), U is considered in line 3, and we are sure to find it and its support in FS (line 4), because every subset of a frequent set is frequent too (see Corollary of Lemma 1).
So all σ-frequent ρ-confident association rules are generated. And by construction only σ-frequent ρ-confident are output. □

We know how to derive all association rules with support and confidence values greater or equal to their corresponding thresholds. These rules fulfil the stated criteria on the evaluation functions, but it does not necessarily mean they are all interesting in the application domain. Some problems may occur, such as redundancy of the discovered rules, or extraction of rules that are already known to the expert. In a few words, the measures of support and confidence are not monotonic with the interest of a particular rule in some particular application, i.e., higher support and confidence values do not necessarily guarantee a higher derived added-value in that application.
It is well known that only a relatively small number of association rules is ultimately used to derive knowledge (i.e., small fraction of the collection of rules brings the majority of the added-value in a particular context). Once, they have been identified and the actions have been derived, the remainder of the collection, even if it contains high-confidence rules, is discarded.

Algorithm 1. (Deriving association rules from frequent sets [Agrawal 94])

Input: *FS the collection of all σ-frequent itemsets and their corresponding supports in a data set r (i.e., $FreqSup(r, \sigma)$), ρ a confidence threshold.*
Output: *The collection of all frequent and confident association rules with their supports and confidences (i.e., $RulesSuppConf(r, \sigma, \rho, R)$).*

1: **let** $C := \emptyset$;
2: **for all** $<X, S_X> \in FS$ **do**
3: **for all** $Y \subset X$ **do**
4: **let** S_Y *be a natural* **such that** $<Y, S_Y> \in FS$;
5: **if** $\frac{S_X}{S_Y} \geq \rho$ **then** // *confidence of* $Y \Rightarrow X \setminus Y$
6: **let** $C := C \cup \{Y \Rightarrow X \setminus Y \quad [S_X, \frac{S_X}{S_Y}]\}$;
7: **fi**
8: **od**
9: **od**
10: **output** C;

Two principal directions of research have been proposed: interactive tools assisting an expert in browsing the collections of discovered rules [Klemettinen 94] or trying to find other, more pertinent evaluation functions. Let us recall a few of them.

Definition 14. (Piatetsky-Shapiro's measure) In [Piatetsky-Shapiro 91] an unnamed measure was proposed, which we call PS. It is defined as: $PS(r, X \Rightarrow Y) = Sup(r, X \cup Y)/|r| - Sup(r, X)Sup(r, Y)/|r|^2$.

The main idea is to consider how the properties X and Y depart from the independence. If X and Y are independent[5], $PS(r, X \Rightarrow Y)$ will be equal to 0. The more statistically dependent they are, the higher is the value of the measure.
A serious drawback of the Piatetsky-Shapiro's measure is that it is symmetrical, scoring in the same way rules $X \Rightarrow Y$ and $Y \Rightarrow X$, which have in general different intuitive meaning.

Definition 15. (conviction) Brin *et al.* in [Brin 97a] proposed the measure called *conviction*, which is defined for a rule $\{A\} \Rightarrow \{B\}$ as:

$$conviction(r, \{A\} \Rightarrow \{B\}) = \frac{Sup(r, \{A\})Sup(r, \{\overline{B}\})}{Sup(r, \{A, \overline{B}\}\ |r|}.$$

The intuition behind the formula is that we can measure the strength of the connection between C and D (statistical events) using the so-called departure from independence, but unlike in the Piatetsky-Shapiro's measure we take the ratio $P(CD)/(P(C)P(D))$ (the so-called statistical dependence[6]), a measure monotonic with the strength of the connection.
We should also observe that the logical implication $A \Rightarrow B$ is equivalent to $\neg(A \wedge \neg B)$ so the strength of the connection between A and $\neg B$ is the opposite of the strength of the "implication" $A \Rightarrow B$. The authors propose a combination of these two ideas, first to obtain $P(A \wedge \neg B)/(P(A)P(\neg B))$, and secondly to inverse it to obtain measure (given in Definition 15) monotonic with the strength of the "implication".
Unfortunately, the authors do not precise how to compute them for a rule with head and/or the body involving more than one item. Bayardo *et al.* generalized this definition in [Bayardo, Jr. 99a].

Definition 16. (conviction for longer rules) Conviction of the rule $X \Rightarrow Y$ is defined as:

$$conviction(r, X \Rightarrow Y) = \frac{Sup(r, X)(1 - Sup(r, Y)/|r|)}{Sup(r, X) - Sup(r, X \cup Y)}.$$

[5]Two statistical events C and D are independent iff $P(CD) = P(C)P(D)$.

[6]The statistical dependence $P(CD)/(P(C)P(D))$ measures how dependent two events are. Its value is 1 in the case of independence.

Definition 17. (improvement) Bayardo *et al.* in [Bayardo, Jr. 99b] proposed *improvement*, which is defined as:

$$improvement(r, X \Rightarrow Y) = \min_{X' \subset X}(Conf(r, X \Rightarrow Y) - Conf(r, X' \Rightarrow Y)).$$

In [Bayardo, Jr. 99b], the extracted rules are aimed at the prediction. Therefore, a rule with lesser confidence is considered as potentially less useful. If a rule with subset body (more general) has a higher confidence, e.g., $\{A, B\} \Rightarrow \{C\}$ $[s_1, 95\%]$ for $\{A, B, D\} \Rightarrow \{C\}$ $[s_2, 93\%]$, the improvement is negative. Thus, the authors consider that such a rule may be pruned, and suggest to use for prediction the rule involving as body X' that is the subset of X corresponding to the highest confidence.

There is a multitude of proposed interestingness measures, providing emphasis on different aspects. Since this thesis is not focused on scoring association rules, we do not enumerate all proposed measures, we only give few more pointers to the papers presenting them. Other recognized measures include *interest* [Brin 97a], *intensity of implication* [Suzuki 98, Guillaume 98] and *J-measure* [Smyth 92]. The statistical significance can also be used for that matter.
Note that if the framework is constructed to extract rules interesting in a context, the authors may call differently the extracted rules, to emphasize different evaluation functions or the use made of these rules.
Interested reader can find an independent comparative summary of statistical measures in [Tan 02], including detailed qualitative and quantitative properties.

Frequent sets are also used for other tasks than pattern mining. E.g., in [Das 98] Das *et al.* use frequent sets to define *external similarity measures for binary attributes* and clustering techniques using frequent sets are considered in [Das 98, Han 98].

Frequent sets have inspired mining forms of patterns other than association rules or episode rules. E.g., Brin *et al.* in [Brin 97b] proposed correlation rules, Mannila *et al.* in [Mannila 96b] proposed mining generalized Boolean rules.
Generalized Boolean rules are similar to association rules, but both sides of the rule, instead of being sets of items, are Boolean expressions involving items as operands.
The correlation rules are less intuitive, so we recall them more formally.

Definition 18. (correlation rules) Given min_{χ^2} (a significance threshold), $\sigma \in [0, |r|]$ (a support threshold), $\gamma \in [0, 1]$ (a population ratio threshold), a *correlation rule* is an itemset X such that $|\{G_X \in FullClauses(X) : Sup(r, G_X) \geq \sigma\}| \geq \gamma \, |FullClauses(X)|$ and

$$\sum_{G_X \in FullClauses(X)} \frac{(Sup(G_X) - E[G_X])^2}{E[G_X]} \geq min_{\chi^2},$$

where $E[G_X] = |r| \prod_{I \in G_X} \frac{Sup(\{I\})}{|r|}$.

Intuitively a correlation rule is an itemset X such that the full clauses based on X (see Definition 10) are not independent. The independence of all full clauses based on X is assessed in this framework using a chi-squared test. The consideration is further restricted to itemsets such that at least a fixed fraction (γ) of full clauses over X has a minimum support σ.
Note that all the uses presented in this section are generic for frequent patterns, e.g., the mentioned interestingness measures may be applied in episode rule mining, episode rule derivation from frequent episodes is analogue to association rule derivation from frequent sets, and so forth.
Note however that the investigations on all combinations of these have not necessarily been reported in the literature.

2.1.5 Potential Problems Associated with Pattern Discovery

For pattern mining, the most important potential problems are related to the choice of pattern language and attributes.
Pattern languages with high expressive power tend to make numerous patterns look interesting in practical data sets, even between attributes that are unrelated in reality. This is due to the fact that within a huge number of patterns, some may happen to fit the data accidentally. This is a situation in pattern mining equivalent to overfitting for models. Thus, a statistical significance assessment must be carried out for patterns with high expressive power (often in the postprocessing step).
High expressive power may be due to the complexity of the language itself (e.g., rich syntax), or simply to the abundance of terminals[7] of the language (e.g., items in association rules), or both.
Therefore, a KDD process often starts preprocessing with so-called feature selection step, where we select for example attributes and the relationships that we want to study. A combined effect of language expression power, number of features, data set, evaluation function, ... can lead to two extreme situations:

- None or too few patterns are interesting,
- Too many patterns are interesting.

Both situations give no significant insight in data.
The first situation may happen if we are unlucky, and the selected features cannot be related by patterns. More frequently however, the situation will arise if the evaluation function is too demanding.
The second situation may happen if we are unlucky, and the selected features are strongly related, and furthermore if several interesting relationships can be combined to give new

[7] Also called terminal symbols, or tokens. It denotes an atomic symbol in a grammar.

number of attributes	number of items	number of interesting association rules
1	2	0
2	8	0
3	12	1
4	22	22
5	24	207
6	33	1390
7	35	5492
8	37	18 728
9	39	63 240
10	51	167 197
11	53	534 246
12	58	1 770 927
⋮	⋮	
23	119	?

Table 2.3: Number of multi-valued attributes, attribute-value pairs (items) and corresponding association rules.

interesting ones. A combinatorial explosion may happen, leading to an untractable task, or (at best) a result which cannot be reasonably analyzed in a postprocessing step (e.g., if we cannot ask the domain expert about each individual pattern out of a collection of 1 million statements). We will refer to such a situation by saying that with respect to a task the data set is *highly correlated.*

The multitude of interesting patterns will also arise if the evaluation function is too feeble (unselective), or the number of selected features is too high (leading to myriads of "accidental" relationships, as explained above).

Coming in-between these extremes may require a number of tries on the data, trading the evaluation function, the pattern language or the number of selected features.

Example 3. *Let us make an experiment concerning the number of actual interesting patterns within a practical data set. Let us suppose we analyze* **Mushroom** *data set*[8]*. It describes 8124 various mushroom species using 23 attributes for each species. Altogether 119 attribute-value pairs (called "items" further on) are actually present in this data set. In this data set we look for association rules with support and confidence thresholds set to* $min_s = 0.01$ *and* $min_c = 0.9$*, i.e., we consider as interesting the rules with support of at least 1% and with confidence of at least 90%.*

Now, let us observe in Table 2.3 the growth of the number of extracted association rules, as we increase the number of attributes (we included attributes in the order they appear in the data set). For the first and two first attributes (resp. 2 and 8 items) the result is empty. One association rule is extracted when considering three first attributes (12 items). Further on, their number grows rapidly. We stopped the experiment when we judged that

[8]A benchmark data set well-known in machine learning community. A preprocessed version of the data set is available at http://www.almaden.ibm.com/cs/quest/data/long_patterns.bin.tar

going any further would produce only less readable insight in the data set[9]*. At that time, we have included only about a half of the attributes.*
One cannot postprocess any number of rules. E.g., if the postprocessing consists in an individual assessment of each rule with respect to the expert's knowledge, not even 1 770 927 rules, but 1000 rules will probably be too many. Then, at most five first attributes can be included in the study. Moreover, considering less than five attributes brings almost no rules, so the study risks to be unproductive (the postprocessing may filter out all extracted rules).

As it can be inferred from the above example, a tough compromise is needed. When the combinatorial explosion is possible, the choice of the number of attributes included in a study, for example, is particularly slim.

2.2 State of the Art in Frequent Set Discovery

Let us now focus on recent algorithms that compute frequent sets.

2.2.1 APRIORI

The first algorithm effectively fulfilling the task of extraction of all frequent sets in a transactional data set is called **APRIORI** [Agrawal 94, Mannila 94].
This algorithm uses a simple but very efficient safe pruning strategy: supersets of an infrequent set can not be frequent (see Corollary of Lemma 1). **APRIORI** searches in the lattice of itemsets (ordered by set inclusion) starting from singletons and identifies level by level larger frequent sets until the largest frequent sets are found. A join-based procedure has been proposed in [Agrawal 94] to generate candidates efficiently.
In the following, we describe important aspects **APRIORI**. The presentation is slightly different than in the original papers. We take into account that the algorithm must output frequent sets and their supports to be able, e.g., to derive and score association rules[10].

We present a version of **APRIORI** enhanced with the prefix-tree data structure inspired from [Mueller 95].

Algorithm 2 shows the pseudo-code of **APRIORI**. This algorithm creates the first candidate itemsets in line 1. Each item potentially present in the database (each item belonging to R) becomes a singleton itemset in $\mathcal{C}_1$. Then, the algorithm enters a loop (lines 2–7). i^{th} iteration of the loop corresponds to itemsets of size i. As long as the current collection

[9]Here, as well as in all further experiments in this thesis, we stop varying the experiment parameters when the number of patterns goes over 1 000 000. We set this condition as a convention, because depending on the actual postprocessing step, the saturation may happen for a few orders of magnitude less patterns, or (rarely) for a few orders of magnitude more.
[10]See Algorithm 1.

of candidate itemsets $\mathcal{C}_i$ is not empty (line 2), the program interleaves the following 2 operations:

- checking for each individual candidate itemset of size i if it is frequent (lines 3 and 4),
- constructing the collection of candidates of size $i+1$ (candidates for the following iteration) from the frequent sets of size i (line 5).

In line 8, the program reports all discovered frequent sets and their corresponding supports in r. The empty set is also output; it is the only frequent set for which the support test may be skipped, because $Sup(r, \emptyset) = |r| \geq \sigma$ (recall: $\sigma \in (0, |r|]$).
The collection of candidate itemsets $\mathcal{C}_i$ is stored in a prefix-tree[11] structure of height i. This structure implements an efficient storage of (ordered) sequences of elements. Each edge of such a tree stores an element of the sequence (an item for itemsets), and a sequence is obtained as the juxtaposition of path elements from the root of the tree (a *path* for short). Since the tree is supposed to represent itemsets of size i, a sequence in the collection $\mathcal{C}_i$ corresponds only to nodes at depth i, and there is no leaves in the tree at other depths. Sequences obtained as paths to an intermediate node are *prefixes* of (possibly multiple) actual sequences stored in the tree. In other words, the tree aggregates the same sequence prefixes into one sequence, the first differing element starts a distinct sub-tree.
To make an itemset correspond to a path in a non-ambiguous way, we take advantage of the linear order of items $\prec$ (see Definition 1) and actually store the sequence of items $ordered\_list(X)$ for an itemset X (see Definition 13).
An example prefix-tree, storing itemsets of size 2 is depicted in Figure 2.2. On that figure, the itemsets stored in the tree correspond to rightmost nodes (leaves of the tree). The

[11] We will use the common terms related to trees, such as *height*.

Algorithm 2. (APRIORI)

Input: *r a binary database over a set of items R, and $\sigma \in [0, |r|]$ an absolute support threshold.*
Output: $FreqSup(r, \sigma)$.

1: **let** $\mathcal{C}_1 := \{\{A\} : A \in R\}$, $i := 1$;
2: **while** $\mathcal{C}_i \neq \emptyset$ **do**
3: *Scan r and compute supports of itemsets in $\mathcal{C}_i$;*
4: $\mathcal{FS}_i := \{Y : Y \in \mathcal{C}_i$ *and* Y *is σ-frequent in $r\}$;*
5: $\mathcal{C}_{i+1} := \{X : X \subseteq R \wedge |X| = i+1 \wedge \bigwedge_{Y \subset X} |Y| = i \Rightarrow Y \in \mathcal{FS}_i\}$;
6: **let** *$i := i + 1$;*
7: **od** *;*
8: **output** $\bigcup_{j<i}\{<X, support\ of\ X> : X \in \mathcal{FS}_j\} \cup <\emptyset, |r|>$;

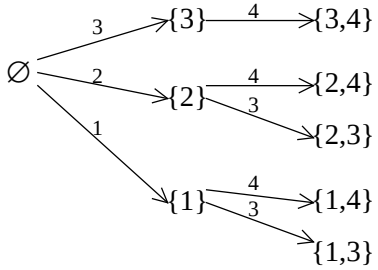

Figure 2.2: Example prefix-tree representing itemsets $\{3,4\}$, $\{2,4\}$, $\{2,3\}$, $\{1,4\}$ and $\{1,3\}$, assuming $\prec$ is the order $<$ of natural numbers.

itemsets corresponding to the prefixes also shown. Neither leaves nor the remaining nodes contain this information — it is shown for clarity, but in the structure only the values associated with the edges need actually be stored.

The prefix-tree structure also allows an efficient support counting. To explain that claim, let us first observe on Algorithm 3 how **APRIORI** counts the individual supports of all itemsets in a collection $\mathcal{C}_k$ within one database scan (the operation occurring in line 3 of Algorithm 2). Algorithm 3 processes one row t of r at a time. For each row t, **APRIORI-Supports** considers every itemset of $\mathcal{C}_k$ (line 2), and if the row matches the candidate itemset, the support count of that candidate is incremented by 1 (line 4).
The advantage of the prefix-tree in this task is the following. The itemset C can be tested for inclusion in t (line 3) by testing successively each item of t against C and finishing when $|C|$ items of t are found in C or there is no more items in t. Thus, the testing does not have to be repeated for itemsets that share a prefix with C for the items of t that have been already found in the shared prefix. Note that t is fixed each execution of lines 2–6 and that we must check all possible itemsets from $\mathcal{C}_k$ (they all have size k), i.e., there is no fixed C as suggested above.
Therefore, in order to consider all candidates of the prefix-tree we start at the root node

Algorithm 3. (APRIORI-Supports)

Input: *$\mathcal{C}_k$ collection of itemsets of size k, r a binary database.*
Output: *Computed supports of all itemsets in $\mathcal{C}_k$ according to the contents of r.*

1: **for all** $t \in r$ **do**
2: **for all** $C \in \mathcal{C}_k$ **do**
3: **if** $C \subseteq t$ **do**
4: *increase by 1 the support counter of C;*
5: **fi**
6: **od**
7: **od**

corresponding to an empty prefix. We take the first item of t and see if with this item $t[1]$ in the prefix we still can find itemsets in $\mathcal{C}_k$. For that, we look up for an edge labeled with the item $t[1]$. If it is found, the tree contains itemsets with $t[1]$ in the prefix, so we move to the corresponding child node. We keep adding items from t to the prefix as long as we keep finding itemsets in $\mathcal{C}_k$ with that prefix, i.e., as long as we can find the edge from the current node labeled with the latest item. If we do not find any, we skip the item from t and try with the following one until we accumulated k items in the prefix or none of items from t is the label of an outgoing edge. If we accumulated k items in the prefix, we found actually an itemset in $\mathcal{C}_k$, because the current node is a leaf (so we increment the corresponding support counter). Then, we backtrack and consider remaining items of t with the prefix corresponding to the parent.
That way, the cost of the test of a single item added to a prefix (the edge look-up) is shared among all itemsets matching the prefix. In general there is a high overlap between prefixes of itemsets, so the cost per itemset is low. The exception is the last element of a path, which is not shared, because it corresponds to exactly one itemset.

Algorithm 4 shows how to implement efficiently line 5 of Algorithm 2. The operation starts with the production of a superset of the desired collection (lines 1–4). **APRIORI-Gen** obtains potential candidate itemsets of size $k+1$ by performing the union of every pair of itemsets of size k that share first $k-1$ items (according to $\prec$). Taking additionally advantage of the properties of sequences representing itemsets (based on the linear order

Algorithm 4. (APRIORI-Gen)

Input: *$\mathcal{FS}_k$ collection of all frequent sets of size k (where $k \geq 1$).*
Output: *$\mathcal{C}_{k+1}$ collection of itemsets of size $k+1$ such that for every of them C all its subsets of size k are in $\mathcal{FS}_k$.*

1: **insert into** $\mathcal{C}_{k+1}$
2: **select** $N[1], N[2], \ldots, N[k-1], N[k], R[k]$
3: **from** $\mathcal{FS}_k\ N, \mathcal{FS}_k\ R$
4: **where** $N[1] = R[1]$ **and** $N[2] = R[2]$ **and** $\ldots$ **and** $N[k-1] = R[k-1]$
and $N[k] \prec R[k]$;

5: **for all** $C \in \mathcal{C}_{k+1}$ **do**
6: **for all** $A \in C$ **do**
7: **if** $C \setminus \{A\} \notin \mathcal{FS}_k$ **then**
8: *delete C from $\mathcal{C}_{k+1}$;*
9: **fi**
10: **od**
11: **od**

12: **output** $\mathcal{C}_{k+1}$;

of items $\prec$), **APRIORI-Gen** avoids the multiple generation of a same candidate. It also partly realizes the pruning, which is completed in lines 6–10, where **APRIORI-Gen** checks if a generated candidate C of size $k+1$ has all its subsets of size k in the collection of frequent sets (all frequent sets of size k are already known), and if not, C may be safely removed from the candidate set. Checking all subsets of size k for a candidate C of size $k+1$ guarantees (by recurrence) that all proper subsets of that candidate are frequent. This is the so-called **APRIORI** *trick*.
As described in [Mueller 95], the prefix-tree is also useful to optimize the candidate generation step. We must recall that an instance of the prefix-tree $\mathcal{C}_k$ in **APRIORI-Gen** may only represent a collection of itemsets that all have the same size. Furthermore, the selection condition on a pair of itemsets (line 4) enforces that they share the same first $k-1$ items. Therefore, the program recursively enters the prefix-tree, until it reaches the last-but-one level of nodes (last non-leaves). For each node N of that level, it combines the leaves that are children of N into pairs and adds each pair separately to the common prefix. Having a common prefix ensures without actual test that the first $k-1$ conjuncts of the **where** clause in line 4 are satisfied.

There are many improved techniques based on the **APRIORI** algorithm. We describe in the rest of the chapter some of them. We also present existing approaches that differ considerably from **APRIORI**.

2.2.2 Sampling Algorithm

APRIORI makes multiple scans of the data set, typically 5 to 10. The cost associated with these scans is the main load of the extraction of frequent sets. Reducing the number of scans has been the driving motivation for Toivonen to propose a **sampling algorithm** in [Toivonen 96].
The **sampling algorithm** aims at reducing to 1 the number of scans of the data set (concluding with a second scan in the worst case). First, it tries to make a *guess* of which itemsets are frequent in the data set, and then to complete one full scan of the data set to check if these itemsets are indeed frequent. Checking an itemset consists in counting its support and comparing it with the support threshold σ.
The proposed method realizing a guess, described later, is imperfect — the actual collection of frequent sets may be different than in the guess[12]. Discovering that some of the itemsets that were supposed to be frequent actually are not is a not a big problem, unless they are so numerous that the support counting procedure is intractable. The bigger problem being the itemsets that were not "suspected" to be frequent, but they are. If the **sampling algorithm** detects after the first scan of data set that some actual frequent sets could have been missed (the guess was not enough "large"), a second scan of the data

[12]Otherwise, it would not be called a guess.

set is necessary, checking itemsets that are the potentially missed frequent sets. Therefore, the **sampling algorithm** prefers to consider a candidate collection that contains extra candidates rather than to miss actual frequent sets.
Let denote by r the binary database and σ the support threshold. The goal is to extract $Freq(r, \sigma)$.
The itemsets likely to be frequent in r (the guess) are obtained as actual frequent sets present in a sample of the binary database. First, **sampling algorithm** draws a random sample of rows of the binary database, called further r', then it computes actual frequent sets present in the sample, i.e., itemsets from $Freq(r', \sigma')$, where σ' denotes the absolute support threshold to be used on the sample.
Given that a little fluctuation of the relative support of an itemset is likely to occur between the sample of the data set and the data set itself, a slightly lower relative support threshold is used on the sample than on the data set itself, i.e., $\frac{\sigma'}{|r'|} < \frac{\sigma}{|r|}$. The goal is to lead to a guess that is a super-collection of the collection frequent sets of r (i.e., we wish to have $Freq(r, \sigma) \subseteq Freq(r', \sigma')$). On the other hand, the support threshold σ' should not be too low, in order to avoid considering unnecessarily big number of candidates.
In [Toivonen 96] the reader may find the details on how to choose the right support threshold σ' and size of r' to make (almost) sure that $Freq(r', \sigma')$ will not miss any frequent set from $Freq(r, \sigma)$. As a basic rule, we may find that reducing the probability of the second scan requires increasing size of the sample r' or decreasing the value of σ'.
The basic **APRIORI** is used on the sample r' to obtain $Freq(r', \sigma')$, but given that the sample is relatively small, it may be read from the disk once and stored in memory. Thus, one reading of the sample is necessary and the several scans required by **APRIORI** can be handled without actual disk scans.
Summarizing the technique, the goal of reducing number of disk scans is achieved by performing a random sample reading (a single reading of a fraction of the data set), one full scan of the data set (more exactly of the remainder of the data set) and a possible-but-unlike second scan of the data set. Reducing the probability of the second scan is traded-off against a larger random sample size used by the **APRIORI**-like frequent set computation technique (with a relatively high CPU load) in the first step or against a smaller σ' leading to a larger candidate collection for the full-scan support counting (higher memory and CPU loads).

2.2.3 DIC

Brin *et al.* proposed in [Brin 97a] an algorithm called **DIC** (for *Dynamic Itemset Counting*). Like **APRIORI**, this algorithm identifies larger and larger frequent sets starting from singletons and until the maximal frequent sets are found. Though the order in which itemsets are considered respects the partial order of itemsets defined by the set inclusion, the itemsets of a same size do not have to be considered simultaneously, as it was the case

in **APRIORI**.

The essence of the approach is to consider a collection of candidates that dynamically changes while scanning a binary database. Observe that **APRIORI** considers an itemset as a candidate when all its subsets are known to be frequent. **DIC** considers an itemset as frequent if its support counter shows a value greater or equal to σ. **DIC** tests if that condition is satisfied earlier than at the end of a scan, notably at few preset points of a scan.

If a new candidate is detected (all its immediate subsets turned out to be frequent), it will be considered from that point of the scan by the support counting procedure. This may lead to a simultaneous counting of its support and of the supports of its subsets (at that time the latter are known to be frequent).

For the itemsets whose support counting started inside a data set scan, at the end of a scan **DIC** wraps and continues from the beginning of the data set until all rows of the binary database have been taken into account. The scan (or scans) of the data set continues until the supports of all candidates are counted over the whole data set and there is no further candidates.

The important advantage of the method over **APRIORI** is a reduced number of data set scans — in most of real data sets a couple of them is enough. Observe that at least one full scan is needed and that above one full scan fractions of a scan (to one of the preset points of the scan) may lead the algorithm to finish. Therefore, when the number of scans is the primary cost factor, the speed-up will be remarkable.

On the other hand, **DIC** requires some additional memory related to each node and each candidate — the starting point of the support counting for a candidate (to be able to find out when a full scan for that candidate has been achieved and the counter must become inactive).

2.2.4 Max-Miner

Max-Miner [Bayardo, Jr. 98] is developed on the base of the following observation. From Corollary of Lemma 1, we know that if X is a frequent set and $Y \subset X$, then Y is a frequent set, too. Therefore, Bayardo proposes in [Bayardo, Jr. 98] to extract only the maximal frequent sets, i.e., the frequent sets not having other frequent sets as proper supersets for a given σ.

Note that this collection can also be described as the positive border of the collection of all frequent sets, $\mathcal{B}d^+(Freq(r,\sigma))$ (see Definition 5). Therefore, **Max-Miner** can be seen as a tool to find the (positive) border of frequent sets.

Pincer Search [Lin 98] is an independently proposed algorithm for mining the maximal frequent sets. In the following, we focus on one of them, namely on **Max-Miner**.

The point of mining maximal frequent sets only is that this collection is very attractive compared to the collection of all frequent sets, because the sizes of these collections typi-

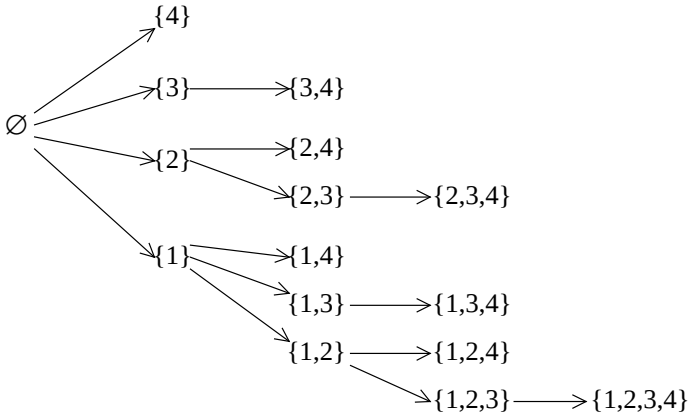

Figure 2.3: Example itemset-enumeration-tree representing all itemsets over $\{1, 2, 3, 4\}$.

cally differ by orders of magnitude, but information frequent/infrequent for each itemset is preserved. In other words, given all maximal frequent sets we may find out if any given itemset Y is frequent. The procedure is sketched in terms of the following theorem.

Theorem 2 (determining frequent/infrequent status of an itemset). *Let r be a binary database over R and σ be a frequency threshold. Y is frequent iff among maximal frequent sets there is a superset of Y.*

Proof. If among maximal frequent sets there is a superset X of Y we justify that Y is frequent by the already-mentioned corollary of Lemma 1.
We prove the forward-directed implication by contraposition. If Y was frequent, either it would be a maximal frequent set, or one of its proper supersets would have to be frequent. If the latter is not a maximal frequent set, we recursively search for larger and larger frequent sets, until we find one without frequent proper superset (R, the itemset of maximal size, is finite, so moving to larger frequent sets must finish at some point). At the end, we get a maximal frequent set that is a superset of Y ($\supset$ is transitive). So, if among all maximal frequent sets, there is no superset of Y, Y cannot be frequent. □

In this case, the proposed extraction algorithm counts the supports of much less itemsets (in general) than there is in the collection of all frequent sets.
The algorithm uses a heuristic to skip checking large portion of the itemset space. It is described below.
Max-Miner stores the candidates in a prefix-tree. In the i^{th} scan of data in the prefix-tree is of height i and candidates correspond only to nodes at depth i (in which **Max-Miner** is similar to **APRIORI**). However, for a single node N we count the support of two itemsets: of the one represented by the path to the node (referred to as *head*) and of the union of the itemsets that are children of the current node in a complete set-enumeration-tree. Let denote this union by $N.U$ and the head of the node $N.H$. Obviously, the value of $N.U$ depends directly on the applied set-enumeration-tree. Let us denote by E this tree.

E is a different tree, not to be confused with the candidate storage prefix-tree. E can be defined in advance of the mining — in that sense it is independent of the candidate storage prefix-tree.
To simplify the following discussion, we neglect that the candidates are stored in a prefix-tree, and the only tree we refer to is going to be the set-enumeration-tree ("nodes", "children" and so on will refer to this tree). Consequently, instead of a node for a candidate itemsets $N.H$ and $N.U$, we use the term a group, and we reserve the term node to nodes of the set-enumeration-tree.

Example 4. *If the head of the group $G.H$ is $\{2\}$, and we use the set-enumeration-tree depicted on Figure 2.3, then the children of node $G.H$ are $\{2,3\}$, $\{2,4\}$ and their union is $G.U = \{2,3,4\}$.*

$G.U$ is the superset of all descendants of $G.H$, so if it is frequent, all itemsets that are descendants of $G.H$ in E will be frequent as well. In this case, **Max-Miner** does not enumerate them and skips counting their supports, i.e., it prunes the sub-tree and keeps $G.U$ as a potential maximal frequent set. Some other sub-trees may also be pruned, because $G.U$ may be a superset of the maximal itemset represented in another sub-tree. Thus, we prune all groups G_j such that $G_j.U \subset G.U$ because we are interested only in maximal frequent sets, and counting any subset of any known frequent set is useless in this context. Pruning of a group G means that we do not probe the children nodes of $G.H$ nor their descendants.
Following Example 4, if $G.U = \{2,3,4\}$ (corresponding to $G.H = \{2\}$) is frequent, the group G' with $G'.H = \{3\}$ and $G'.U = \{3,4\}$ may be pruned, too.
Additional optimizations are integrated in **Max-Miner**, such that:

- dynamic construction of the set-enumeration-tree between scans of the data set, defining level by level the nodes which are required for that particular scan. Defining this tree influences the $G.U$ itemset of each created group and lets enhance the pruning described above,
- to compute the supports of $G.H$ and $G.U$, a row is first matched with $G.H$, and then every additional item present $G.U$ is checked if it is in the current row. Each of these individual checks may be reused separately to efficiently count the supports of the heads of the node's children (one level ahead),
- given the supports gathered for ancestors of a node **Max-Miner** uses a simple support bounding to determine whether some candidates are frequent without counting their supports.

The last of the above optimizations and the group pruning described previously could have been employed, because **Max-Miner** considers maximal frequent sets as self-sufficient description of all frequent sets. The idea to consider only some frequent sets that provide the

necessary information to determine all frequent sets (in this case the frequent/infrequent status of every itemset) is important.
There is however a major drawback of maximal frequent sets — they do not provide the actual support values for all frequent sets, which are required, for example, by Algorithm 1 to score the association rules. Bayardo proposes in [Bayardo, Jr. 98] as remedy to scan the data set and count the supports of all frequent sets (this latter collection is derived from the maximal frequent sets). The cost of this final scan is in the same order of magnitude as of running **APRIORI**, and that part of computation dominates the running time ([Bastide 00b] offers, among other results, an experimental comparison of two steps of frequent set computation using **Max-Miner**).
As for the number of scans, **Max-Miner** reduces it slightly, not as much as by employing **DIC** or the **sampling algorithm**.

2.2.5 FP-Growth

Recently, Han *et al.* proposed in [Han 00] **FP-Growth** — a technique for mining frequent sets that is more different from **APRIORI** than the techniques described hitherto. **FP-Growth** explores the search space in a depth-first manner. We should also mention here an independently proposed **Tree-Projection** [Agarwal 99, Agarwal 01], an early bird of depth-first techniques for frequent set mining.
FP-Growth considers an itemset as candidate only if its parent itemset is frequent. The parent is determined as the parent in a complete set-enumeration-tree (denoted E in the following)[13]. This allows a divide-and-conquer approach. The set-enumeration-tree is used to never generate twice the same candidate.

Algorithm 5 shows the top-level pseudo-code of **FP-Growth**. The pseudo-code represents a recursive function $FindFS$ that explores the set-enumeration-tree E in a depth-first manner. A call $FindFS(X, r', \sigma, E)$ finds in r' the σ-frequent itemsets (and their supports) in a well-defined partition of the search space — the function will consider only the part of E corresponding to the descendants of X. X itself is supposed to be processed by the function calling $FindFS$. **FP-Growth** makes sure that the supports of descendants of X mined in r' are the same as in r, so the frequent sets found in these r' (different in each call of $FindFS$) are identical to frequent sets in r. We justify it shortly, after the description of the top-level pseudo-code of **FP-Growth**.
Since $\emptyset$ is the root of E, all itemsets except the empty set itself are its descendants, so the call $FindFS(\emptyset, r, \sigma, E)$ returns $\mathcal{FS}$ and we have $FreqSup(r, \sigma) = \mathcal{FS} \cup \{<\emptyset, |r|>\}$. As in case of other algorithms, the empty set can be added without computing its support, because $Sup(r, \emptyset) = |r| \geq \sigma$ (recall: $\sigma \in [0, |r|]$).

[13] An example set-enumeration-tree, depicted in Figure 2.3, accompanies the description of **Max-Miner**.

In line 1, the algorithm initializes a temporary variable for the result (returned in line 10). Then, it considers successively all children Y of the entry point node (i.e., of X) in line 4. If Y is frequent (line 5) descendants of Y are considered (recursive call in line 6) and all frequent sets from the sub-tree rooted at Y are stored in the buffer (line 7).

A particular operation is based on the following remark. Notice that the rows matching every descendant of Y must also match Y, so the rows matching Y (i.e., $\mathcal{M}(r, Y)$) are sufficient to determine all frequent sets that are descendants of Y. Moreover, given that for every X, Y such that $X \subset Y$ we have $\mathcal{M}(r, Y) \subseteq \mathcal{M}(r, X)$ (see Lemma 1), we may compute $\mathcal{M}(r, Y)$ incrementally[14] as $\mathcal{M}(r, Y) = \mathcal{M}(r', Y)$, where $r' = \mathcal{M}(r, X)$. So the trick works in the recursive manner for nested calls (in line 6).

Another good property linked to this depth-first manner of computing frequent sets is that the difference between Y and X (i.e., the difference between a child and its parent in E) contains 1 item. Let us denote this item by A. Notice that $\mathcal{M}(r, Y) = \mathcal{M}(r, X \cup \{A\}) = \{t \in r : t \in \mathcal{M}(r, X) \wedge t \in \mathcal{M}(r, \{A\})\}$. Therefore, obtaining $\mathcal{M}(r, Y)$ from $r' = \mathcal{M}(r, X)$ consists in applying an elementary filter defined as: $t \in \mathcal{M}(r', Y)$ iff $t \in r' \wedge A \in t$.

It is necessary to determine the support of a candidate itemset to compare it to σ (line 5) and eventually to store it in the temporary variable (line 7) the latter serving as buffer for output (line 10). It is accomplished in line 3.

As explained above, the rows matching Y may be obtained as rows matching 1 item in

[14] The term "incremental" may be misleading, because the collection of rows matching the itemset never grows, and typically decreases, and it is the matched itemset which grows.

Algorithm 5. (FP-Growth)

Function $FindFS(X, r', \sigma, E)$
Input: *X the itemset considered as the current starting point in the search space, r a binary database over a set of items R, σ an absolute support threshold, and E is a complete set-enumeration-tree.*
Output: *All pairs $<Y, Sup(r', Y)>$ such that Y is a descendant of X in E and it is σ-frequent in r'.*

1: $\mathcal{FS} := \emptyset$;
2: **let** $\mathcal{C} := \{Y : Y \in children\ of\ X\ in\ E\}$;
3: *Scan r' and compute supports of itemsets in $\mathcal{C}$;*
4: **for all** $Y \in \mathcal{C}$ **do**
5: **if** *Y is a σ-frequent item in r'* **then**
6: $\mathcal{F}' := FindFS(Y, \mathcal{M}(r', Y), \sigma, E)$;
7: $\mathcal{FS} := \mathcal{FS} \cup \mathcal{F}' \cup \{<Y, support\ of\ Y>\}$;
8: **fi**
9: **od**
10: **return** $\mathcal{FS}$;

r', so the support counting for Y (counting the number of rows matching Y) is computationally a very simple operation on r'. However, one scan of a disk resident r' per call to $FindFS$ is not reasonable, because it would result in the same number of disk scans as the number of frequent sets (likely to be several thousands in real contexts). Even if the size of r' decreases for recursive calls of $FindFS$, the resulting Input/Output load[15] would be unacceptable.

Therefore **FP-Growth** uses a single disk scan of the data set[16] and stores the entire binary database in memory. Further accesses (all scans by $FindFS$) are then redirected to a memory resident binary database.

An efficient storage of a binary database is then necessary to fit it in memory, which is a critical question for typically large data sets processed in KDD. The proposed solution is... again a prefix-tree. It is not unreasonable, given that a binary database is a collection of itemsets (see Definition 1) and several advantages of that structure were made clear in the section devoted to **APRIORI**. However, the use of the structure is now quite different — it is used to store the rows of the binary database and not the candidates. In this case, all rows of a binary database are stored in a single prefix-tree.

Each node (recall: it corresponds to a unique prefix) has as attribute a positive integer. It stores the number of rows in the binary database starting with the corresponding prefix. Different prefixes are thus mapped into different nodes (and their paths) and same prefixes (of different rows) are aggregated into 1 node (and its path), the node attribute reporting the corresponding number of rows. This way, the itemsets of different sizes or possible several occurrences of same rows are handled without loss of relevant information.

Then, the support counting of an itemset $Y = X \cup \{A\}$ in the (incrementally updated) $r' = \mathcal{M}(r, X)$ consists in traversing the prefix-tree representing the rows of r' and summing up the number of all rows containing A. It may be accomplished by summing up the attributes of the nodes that have the incoming edge labeled with A, the explanation is the following. Every row containing an item A may be split into the prefix of the row ending with the item A and the reminder of the row. All the rows starting with the same such a prefix P are aggregated in the prefix tree, and share that prefix. Therefore, their number is counted by the attribute of the node N_P corresponding to the prefix P (ending with A, so the incoming node is labeled with A). Then, it is sufficient to sum up the attributes of these nodes (no need to enter the sub-tree below N_P because all rows represented in there have been taken into account by the attribute of N_P). On the other hand, collections of rows not having a common prefix ending with A are necessarily disjoint, so the summing counts exactly once each row containing A.

The support counting of line 3 performs a simultaneous counting for all candidates in $\mathcal{C}$ (candidates produced by the current call of $FindFS$). Therefore, we may create an array

[15] I/O load, from now on.

[16] Two scans, if we take into account of an optimization described later.

of $|\mathcal{C}|$ counters, each corresponding to a different item A_i (each A_i is an item that must be added to X to obtain one of X's children). Then, the scan would traverse completely the prefix-tree representing r'. For each traversed edge labeled with an item A_i and leading to a node with an attribute value S, we would increment by S the counter corresponding to A_i (if $X \cup \{A_i\}$ is one of the candidates in $\mathcal{C}$).
Further optimizations in **FP-Growth** (described briefly, in the order of successive consequences) include:

- instead of a full tree E, each call to $FindFS$ uses only an ordered set of items $Tail_X$ (disjoint from X) such that the itemsets X and $X \cup Tail_X$ delimit the partition of the search-space to be considered by $FindFS$[17],
- $Tail_X$ is further reduced to the items that are frequent in r', which eliminates parts of E that are simply known to contain only infrequent sets,
- when calling $FindFS(Y, \mathcal{M}(r', Y), \sigma, Tail_Y)$ in line 6, $\mathcal{M}(r', Y)$ preserves only the items contained in $Tail_Y$, because the presence of all items Y is already checked by computing $\mathcal{M}(r', Y)$ and the exploration of the corresponding partition of the set-enumeration-tree neither contains items other than $Y \cup Tail_Y$ nor requires supports involving them,
- by dynamically adapting the order in $Tail_X$ for each X, **FP-Growth** dynamically adapts E and defines it level by level, on the when-needed basis. When defining the sub-tree of a node representing X in the no-longer-materialized set-enumeration-tree E, most frequent children Y are given a least $Tail_Y$ (small sub-tree), which enhances sharing prefixes for parts of the binary database containing lot of rows. The counterpart is to allow bigger $Tail_Y$ to the least frequent children Y, but the corresponding prefix-trees are likely to be small because the part of the database contains less rows,
- dynamic definition of the order of items that is used to sort the items in rows before the construction of a particular prefix-tree (representing an r'). Redefining this order each time a prefix-tree corresponding to a collection of rows is constructed may enhance the overlap between the prefixes of the rows. On the other hand, it requires a preliminary disk scan over the binary database to determine the order of items to be used during the second disk scan when the first prefix-tree, corresponding to the full binary database, is constructed (bringing the number of scans to 2), and
- when the current prefix-tree (for an r') consists of a single path (the tree has only one leaf), a dedicated very fast procedure directly enumerates the frequent sets that are descendants of the current node.

[17] If the order of items of an ordered $Tail_X$ is respected for the full sub-tree, X and $Tail_X$ uniquely define that sub-tree of a set-enumeration-tree.

Summarizing, **FP-Growth** is a technique very different from **APRIORI** for mining frequent sets and their supports. Several optimizations are employed to improve CPU performances. The binary database is hold in memory to reduce otherwise excessive I/O load.

2.2.6 CLOSE

Pasquier *et al.* proposed in [Pasquier 99c] **CLOSE** — an algorithm apparently similar to **APRIORI**, but with fundamental differences making it an important innovation in frequent set mining. The fundamentals of the method are in Galois connections [Ore 44] and formal concept analysis [Wille 82].
The algorithm aims at mining highly correlated binary databases, i.e., databases where many strong correlations between items hold and the resulting frequent sets are long and numerous.
This technique is based on the so-called *frequent closed sets*. So, we first recall the necessary definitions. For proofs of lemmas and of claims as well as for a detailed description see [Pasquier 99c].
We defined in Section 2.1.3 the multiset $\mathcal{M}(r, X)$ of rows matched by the itemset X in r. Now, we give the definition of $\mathcal{I}(r')$, the set of items matching a multiset of rows r'.

Definition 19. (itemset matching a set of rows) Given r' a binary database over R, $\mathcal{I}(r') = \{A \in R : \bigwedge_{t \in r'} A \in t\}$.

Definition 20. (closed set) Given r a binary database over R, X is a closed set w.r.t. r iff $X = \mathcal{I}(\mathcal{M}(r, X))$. $Closed(r) = \{X : X \subseteq R \text{ and } X = \mathcal{I}(\mathcal{M}(r, X))\}$ denotes the set of all closed sets in r.

Definition 20 states that an itemset X is closed w.r.t. r if and only if it contains every item that is present in all rows of r matched by X. In other words, for each item A not in a closed set X, there is an object in the database that has all items of X, and does not have the item A.

Example 5. *In Table 2.1 (Section 2.1.1), the itemsets $\{A\}$ and $\{B, C\}$ are closed, whereas the itemset $\{B\}$ is not. Observe that the rows matching $\{B\}$ are the first two and the last two, all containing also the item C. So, $\{B\} \neq \mathcal{I}(\mathcal{M}(r, \{B\})) = \{B, C\}$ and thus $\{B\}$ is not closed.*

The compound function $\mathcal{I}(\mathcal{M}(r, X))$ returns the so-called *closure* of X w.r.t. r, and the corresponding operator is denoted in the following as $\mathcal{I} \circ \mathcal{M}$. Relevant properties of this operator are stated in the following lemmas.

Lemma 3. *Let r be a binary database over R, and X, Y be itemsets such that $X \subseteq Y \subseteq R$. $\mathcal{I} \circ \mathcal{M}(r, X) \subseteq \mathcal{I} \circ \mathcal{M}(r, Y)$.*

Lemma 4. *Let r be a binary database over R, and X be an itemset such that $X \subseteq R$.* $\bigwedge_{\substack{Y \subseteq \mathcal{I} \circ \mathcal{M}(r,X) \\ Y \supseteq X}} Sup(r,Y) = Sup(r,X)$.

Lemma 5. *Let r be a binary database over R, and X, Y be itemsets such that $X \subseteq Y \subseteq R$. $Sup(r,Y) = Sup(r,X)$ implies $Y \subseteq \mathcal{I} \circ \mathcal{M}(r,X)$.*

A closed set is called frequent closed set when its support in the database is not less than a given threshold σ. The following definition formalizes the notion.

Definition 21. (frequent closed sets) Given r a binary database over R and σ a frequency threshold, the set of all frequent closed sets w.r.t. r is defined as $FreqClosed(r,\sigma) = Freq(r,\sigma) \cap Closed(r)$.

Algorithm 6 shows the pseudo-code of **CLOSE**.
CLOSE belongs to the class of levelwise algorithms. At i^{th} iteration, corresponding to the i^{th} sacn over the binary database, it considers only the candidates of size i. They are stored as *generators* in $\mathcal{CG}_i$. The algorithm starts with $\mathcal{CG}_1$ initialized to the set of candidates of size one (line 1). Each item potentially present in the database (each item belonging to R) becomes a singleton generator in $\mathcal{CG}_1$. In each iteration, **CLOSE** makes a scan of the database, computes supports of the candidate itemsets and their respective closures (line 3). Once the scan is accomplished, the closed sets for which corresponding generators are frequent (line 4) are to be stored with their respective supports (for simplicity, we represent only the storage of generators in a temporary variable, but both their closures and their supports should be stored too). The iteration completes by generating the next iteration candidates (line 5). Once it has been confirmed that all frequent closed sets have

Algorithm 6. (CLOSE)

Input: *r a binary database over a set of items R, and $\sigma \in [0, |r|]$ an absolute support threshold.*
Output: *$FreqClosed(r,\sigma)$ and the corresponding support values (with a possible exception, see explanation in text).*

1: **let** $\mathcal{CG}_1 := \{\{A\} : A \in R\}$, $i := 1$;
2: **while** $\mathcal{CG}_i \neq \emptyset$ **do**
3: *Scan r and compute supports and closures of itemsets in $\mathcal{CG}_i$;*
4: $\mathcal{FG}_i := \{Y : Y \in \mathcal{C}_i$ *and* Y *is σ-frequent in $r\}$;*
5: $\mathcal{CG}_{i+1} := \{X : X \subseteq R \wedge |X| = i+1 \wedge \bigwedge_{Y \subset X} |Y| = i \Rightarrow (Y \in \mathcal{FG}_i \wedge X \not\subseteq$ *closure of* $Y)\}$;
6: **let** *$i := i + 1$;*
7: **od** *;*
8: **output** $\bigcup_{j<i}\{<$*closure of* X, *support of* $X> : X \in \mathcal{FG}_j\}$;

been computed, i.e., there are no more generators (line 2), the program finishes iterating, outputs the frequent closed sets removing duplicates and terminates.
The algorithm **CLOSE** does not consider the empty set the same way as other itemsets. This is a similar behavior to other algorithms, because the empty set is know to be frequent. However in case of **CLOSE**, only frequent closed sets should be output, so we cannot simply add $\{<\emptyset, |r|>\}$ to the result — we know that $\emptyset$ is frequent, but we do not know if it is closed without looking at the data. Fixing the error is straightforward, e.g., one may consider separately $\emptyset$ as a candidate generator, compute its closure along with the first scan of the binary database and add to the result the pair consisting of the empty set's closure and support. In the following, for simplicity we suppose that the error is fixed that way[18].

Algorithm 7 shows how to realize efficiently line 3 of Algorithm 6.
As compared to Algorithm 3, Algorithm 7 has the additional task to compute the closure of each generator. It is accomplished in lines 5–9, according to Lemma 5. That lemma states that if an item A is present in all rows matching an itemset C, it is a member of the closure of C. A is present in all rows matching C also means that A belongs to the intersection of all rows matching C. An item not present in a row matching C neither is in its closure (see Definition 20), so the intersection of all rows matching C contains all and only items from the closure of C.
So Algorithm 7 finds out the closure of C by computing the intersection of all rows

[18]In the experiments presented in this thesis, we used the implementation of **CLOSE** that handles all itemsets up to size 2 in a specific, optimized manner, so the empty set is processed specifically anyway.

Algorithm 7. (CLOSE-Supports-and-Closures)

Input: *$\mathcal{CG}_k$ collection of itemsets of size k (where $k \geq 1$), r a binary database.*
Output: *Computed supports of all itemsets in $\mathcal{CG}_k$ following the contents of r. The closures of itemsets in $\mathcal{CG}_k$ matched by at least 1 row are computed as well.*

1: **for all** $t \in r$ **do**
2: **for all** $C \in \mathcal{CG}_k$ **do**
3: **if** $C \subseteq t$ **do**
4: *increase by 1 the support counter of C;*
5: **if** *closure of* $C = \emptyset$ **then**
6: **let** *closure of* $C := t$;
7: **else**
8: **let** *closure of* $C := t \cap$ *closure of* C;
9: **fi**
10: **fi**
11: **od**
12: **od**

matching C, and that for each itemset from $\mathcal{CG}_k$.

Algorithm 8 shows how to realize efficiently line 5 of Algorithm 6. In this line there is a major difference with **APRIORI** on which **CLOSE** relies to boost its performances on highly correlated data.
We shortly explain the properties of the itemset lattices on which that substantial optimization relies. First, the supports of an itemset, of its closure and of all intermediate itemsets (w.r.t. the itemset inclusion) are the same (Lemma 4). Thus, once identified the closure of an itemset to be different from this itemset, we can exclude the closure and all intermediate itemsets from the support counting procedure (their supports are all known to be the same). Pruning an itemset C that is included in the closure of its subset of size $|C| - 1$ appears in line 7 as the second disjunct. Supersets of such an itemset C are pruned as the combined effect of both conditions stated in line 7, because once an itemset is rejected to be considered as generator, the first disjunct stated in line 7 blocks its proper supersets from being generators.
But what if an itemset X is a superset of a pruned itemset C on the base of the closure of a $C \setminus \{A\}$, but X itself is not included in the closure of that $C \setminus \{A\}$? Can it be safely pruned?
Note that $X \setminus \{A\}$ is not a superset of C and it may be a generator. If so, based on Lemma 3, its closure will contain A and thus we will know the support of X (which is

Algorithm 8. (CLOSE-Gen)

Input: *$\mathcal{FG}_k$ collection of all frequent generators of size k (where $k \geq 1$).*
Output: *$\mathcal{CG}_{k+1}$ collection of itemsets of size $k+1$ such that for every of them C all its subsets of size k are in $\mathcal{FG}_k$ and none of its subsets in $\mathcal{FG}_k$ has as closure an superset of C.*

1: **insert into** $\mathcal{CG}_{k+1}$
2: **select** $N[1], N[2], \ldots, N[k-1], N[k], R[k]$
3: **from** $\mathcal{FS}_k\ N, \mathcal{FS}_k\ R$
4: **where** $N[1] = R[1]$ **and** $N[2] = R[2]$ **and** $\ldots$ **and** $N[k-1] = R[k-1]$
and $N[k] \prec R[k]$*;*

5: **for all** $C \in \mathcal{CG}_{k+1}$ **do**
6: **for all** $A \in C$ **do**
7: **if** $C \setminus \{A\} \notin \mathcal{FG}_k$ **or** $C \subseteq$ *closure of* $C \setminus \{A\}$ **then**
8: *delete C from $\mathcal{CG}_{k+1}$;*
9: **fi**
10: **od**
11: **od**

12: **output** $\mathcal{CG}_{k+1}$*;*

the same as the support of $X \setminus \{A\}$). On the other hand, if $X \setminus \{A\}$ is not a generator, either:

1. it is in the closure of one of its proper subsets, or
2. it is a superset of an itemset that had been pruned based on a closure, but $X \setminus \{A\}$ itself is not included in that closure.

If the first case holds then the support of that subset is the same as of $X \setminus \{A\}$, and consequently we know the support of X (equal to the support of $X \setminus \{A\}$).
If the second case holds then we consider recursively smaller itemsets, and repeat the reasoning until we find an itemset that has been a candidate generator. We denote that generator X', and the string of support equalities accumulated on the way permits to affirm that $Sup(r, X) = Sup(r, X')$.
Pruning due to a too low support (infrequent sets) works as in **APRIORI** and avoids considering the proper supersets of an infrequent set. This pruning works independently of whether these proper supersets have been pruned based on the closures. Therefore, infrequent sets may be correctly determined based only on infrequent generators, and no interference with the pruning on the closures occurs.

Interesting alternative techniques to mine frequent closed sets include **CHARM** [Zaki 02] and **A-Close** [Pasquier 99b].

Mining frequent closed sets is one step, but if we want to compute all frequent sets, we need to convert the frequent closed sets into frequent sets. The corresponding algorithm is given as Algorithm 9. We modified it slightly, to avoid relying on the specific order of itemset enumeration described in the example accompanying the original algorithm in [Pasquier 99c].
We take advantage of this algorithm to introduce a handy notation that we will use for some algorithms. We will be using a record-like access to data. The data record corresponding to one frequent set will be called a node. Here is its structure:

itemset	: set of items
support	: integer

Let $\mathcal{FS}_i$ denote the collection of nodes corresponding to itemsets of size i. A collection of such nodes is stored in an itemset-prefix-tree, a structure allowing efficient access, as we explained for the previous algorithms.
For a node N, the access to these fields is denoted respectively $N.itemset$ and $N.support$. $N.itemset$ corresponds to a frequent itemset[19] and $N.support$ is its support.

[19]The field *itemset* is represented in the algorithm for clarity, but using an *itemset*-based prefix-tree structure provides a much more compact storage of the items in this itemset.

In lines 2–7 **Freq-Closed-To-Freq** loads the input and converts it into the records (nodes). The frequent closed sets are partitioned following their sizes in order to access them in descending size order in the main loop (lines 8–19).
The main operation is to consider every C in line 9, either a frequent closed set or a regenerated frequent set, and to enumerate each proper subset of C of size $|C| - 1$ (each item is removed separately in line 11). If the node corresponding to that subset is already in $\mathcal{FS}_{|C|-1}$, the support is updated. Otherwise, a new node is created, and the support of that subset is set to the value of support of C (lines 12–16).

Mining frequent closed sets instead of all frequent sets is a big step towards more efficient techniques for highly correlated data sets. **CLOSE** checks against the data set only some frequent sets (generators) and stores only their closures (which are closed sets) and their supports. The remaining frequent sets may be derived from the former ones using **Freq-Closed-To-Freq** in a sound and complete manner. Moreover, that algorithm can derive

Algorithm 9. (Freq-Closed-To-Freq)

Input: *$\mathcal{FC}$ a collection of all frequent closed sets (i.e., $FreqClosed(r, \sigma)$) and their corresponding supports.*
Output: $FreqSup(r, \sigma)$.

1: **let** $max\_size := 0$;
2: **for all** $X \in \mathcal{FC}$ **do**
3: *Create new node N_X in $\mathcal{FS}_{|X|}$ with $N_X.support :=$ support of X, $N_X.itemset := X$; // partitioning $\mathcal{FC}$*
4: **if** $|X| > max\_size$ **then**
5: **let** $max\_size := |X|$;
6: **fi**
7: **od**

8: **for** $i := max\_size$ **downto** 1 **do**
9: **for all** $N \in \mathcal{FS}_i$ **do**
10: **for all** $A \in N.itemset$ **do**
11: *Find N_A in $\mathcal{FS}_{i-1}$* **such that** $N_A.itemset = N.itemset \setminus \{A\}$;
12: **if** *such N_A exists* **then**
13: **let** $N_A.support := max(N_A.support, N.support)$;
14: **else**
15: *Create new node N_A in $\mathcal{FS}_{i-1}$ with $N_A.support := N.support$, $N_A.itemset := N.itemset \setminus \{A\}$;*
16: **fi**
17: **od**
18: **od**
19: **od**

20: **output** $\bigcup_{j \leq max\_size} \{<N.itemset, N.support> : N \in \mathcal{FS}_j\}$;

the support for each frequent sets without any access to the original data set, unlike in the **Max-Miner** approach.
We studied the behavior of **CLOSE** on real data sets with our own implementation of that algorithm (see [Bykowski 99, Boulicaut 00a]). Considering less candidates reduces CPU and memory loads, typically an order of magnitude on highly correlated data. Additionally, the fact that the largest generators are typically smaller than the largest frequent sets leads to a reduced number of scans, typically by half on highly correlated data as compared to **APRIORI** (not as much as the **sampling algorithm** or **DIC**).

2.2.7 Comparative Critique of Frequent Set Mining Tools

The basic frequent set mining technique, **APRIORI**, has been quite successful on the basket-type data, for which it was designed. It makes several linear scans of data (in general less than a dozen), considers one row at the time and increments the support counters for candidate itemsets included in the row.
This technique typically does not work very well on different types of data, especially on highly correlated data. High correlation makes some long itemsets relatively highly "supported" (see [Bayardo, Jr. 98]). If the size of the largest frequent set is N, the number of its subsets is 2^N and the number of scans **APRIORI** makes is N (sometimes $N+1$). So, if the largest frequent set gets larger, it may result in an exponential increase of the number of candidates to be considered by **APRIORI** and a linear increase of the number of scans.
Early proposals after **APRIORI** were designed to reduce the number of scans of the binary database. The **sampling algorithm** reduces the number of scans to one full scan, the second full scan remains possible, but unlikely. **DIC** does not provide any worst-case improvement over **APRIORI** for the number of scans, but the average figure (typically a couple of scans) is better than for **APRIORI** and slightly worse than for the **sampling algorithm**.
Then, other types of data sets were considered, and it became obvious that the principal difficulties to mine them are related to the high number of frequent sets (leading the high CPU and memory loads) and not to the number of scans. We did not present research results that parallelize the task of frequent set mining[20]. Though these are very challenging techniques, they do not reduce the total CPU or memory load, but they split it up and assign to several computers or processors in order to make the extraction faster. We looked rather for methods that reduce the overall CPU and memory loads. Indeed, they could be studied further for parallelization if necessary.
From that view point, the **sampling algorithm** and **DIC** do not reduce the memory load, but they are likely to reduce the CPU load. We base the following explanation on

[20] An interested reader should see e.g., [Zaki 97].

the hypothesis that we use the prefix-tree structure to represent candidate itemsets (as described in Section 2.2.1).

Let us now assume that the data set is very large. Due to large size of the data set all operations except scans have negligible loads as well, which is approximately what we observe typically using **APRIORI** on large data sets. Moreover, on current platforms, the marginal I/O load of reading one row is negligible before the marginal load of support counter updates corresponding to the row (see, e.g., [Mueller 95] for details).
So, we compare exclusively the support counting procedures of **APRIORI**, of the **sampling algorithm** (after the guess step) and of **DIC**. Then, the major factors contributing to the CPU load are: the inclusion test (corresponding to line 3 of Algorithm 3) and unit support increments (corresponding to line 4 of Algorithm 3).
Note that the number of unit support increments in **DIC** is exactly the same as in **APRIORI**, because the same overall collection of candidates is considered by both algorithms and there is the same elementary support counter update (increment by 1). Therefore, this part of the cost remains unchanged.
In the **sampling algorithm**, the number of candidates is at least the same and typically is larger than in **APRIORI**. The elementary support counter update remains the same, so this part of the cost typically increases when compared to **APRIORI**.
The second retained operation with assumed major cost is the inclusion test of a candidate in a row. Recall that **APRIORI** stores the candidates in a prefix-tree and searching for candidates matching a row is performed by edge look-ups, going to child nodes and backtracking (see the description accompanying Algorithm 3).
Both the **sampling algorithm** and **DIC** consider more candidates at the time than **APRIORI**. This difference has an effect on the performances of the inclusion test.
We would not achieve any gain if all three algorithms stored the candidates in prefix-trees partitioned according to the candidate sizes:

- the **sampling algorithm** would make additional edge look-ups (as compared to **APRIORI**) due to larger number of candidates. Since the additional candidates are not frequent, the overhead is low,

- **DIC** would lose more efficiency — it would have to make additional edge look-ups, not because of infrequent sets, but because a particular prefix-tree would likely be accessed during more than one scan due to asynchronous creation of candidates of the same size (a prefix may be shared between 2 itemsets, but if the support counting of one itemset is finished before the beginning of the support counting of the other, the sharing of the prefix does not benefit the support counting and there is a duplication of the work to do as compared to **APRIORI**). Since this happens for prefixes with high support (prefixes are never less frequent than frequent sets starting with these prefixes), the overhead would be significant.

We can possibly achieve some CPU load gain in this context using a unique prefix-tree for all candidates. This gain might come because more candidates than in **APRIORI** may share a same prefix, so edge look-ups in prefixes shared by several candidates would be shared too (however the asynchrony of candidate generation in **DIC** and thus of support counting goes up against).
Under our assumptions, an optimistic upper bound for relative CPU gain over **APRIORI** we might achieve with **DIC** is therefore the ratio:

$$gain\ limit = \frac{\text{number of edge look-ups leading to a leaf in } \textbf{APRIORI}}{\text{total number of edge look-ups in } \textbf{APRIORI}}$$

which means, we must have at least 1 edge look-up per candidate itemset (edge look-up leading to a node with the corresponding support counter) and per support counter increment. The above algorithms cannot overcome this limit.
But for highly correlated data sets, this ratio is indeed quite high and thus the resulting gain may be fine.

Max-Miner reduces CPU load at a much higher level than **DIC** and the **sampling algorithm** when mining frequent sets on highly correlated data, but it does not provide the support values for all frequent sets.
One additional scan of the data set performed by **Max-Miner** to count the supports of all frequent sets takes advantage of storing all candidates in a single prefix-tree, as do **DIC** and the **sampling algorithm**, but cannot break through the limit of one edge look-up per unit support increment. Therefore, neither **Max-Miner** cannot overcome the CPU gain limit indicated above, when computing frequent sets and their supports.

A rather important advance for mining frequent sets in highly correlated data comes with **FP-Growth**. It aggregates different rows into one and thus can increment support counters by more than 1 in a single operation. Several other optimizations accompanying this idea make it quite profitable in general.

A different idea is the essence of the algorithm **CLOSE**, which increments the support counters 1 by 1, but considers much less candidates than **APRIORI**, **DIC**, the **sampling algorithm** and **FP-Growth**[21]. **CLOSE** processes less candidates without loss of information by detecting the most obvious form of redundancy. Since the algorithm only works directly on selected itemsets (i.e., on closed sets; the definition is in the section devoted to **CLOSE**), it typically considers much less candidates, which saves CPU and memory resources.

[21] **Max-Miner** also considers much less candidates than these algorithms, but only to find maximal frequent sets, which can be seen as an important loss of information about all frequent sets. In order to compute the supports of all frequent sets, **Max-Miner** retreats and considers them all for the support counting.

Trait \ Algorithm	**APRIORI**	**sampling algorithm** (after guess)	**DIC**	**Max-Miner** (maximal frequent sets)
search space traversal	levelwise	none	ordered by inclusion	levelwise
candidate processed together	all having a same size	all in the guess	dynamically changing	all having a same size
candidate collection structure	prefix-tree	prefix-tree	dynamic prefix-tree	prefix-tree
candidate generation	join	guess on a sample	not specified (join?)	children in E
pruning	supersets of infrequent	none	supersets of infrequent	descendants of infrequent, subsets of frequent
support counting primitive	increment by 1	increment by 1	increment by 1	increment by 1
intended itemsets	all frequent	all frequent	all frequent	maximal frequent
data access when support counting	linear scans	linear scans	linear scans	linear scans
typical disk scan number	N	1 (rarely 2)	$\ll N$	$< N$

Trait \ Algorithm	**FP-Growth**	**CLOSE**	**CLOSET**
search space traversal	depth-first	levelwise	depth-first
candidate processed together	siblings in E	all having a same size	siblings in E
candidate collection structure	array	prefix-tree	array
candidate generation	children in E	join	children in E
pruning	descendants of infrequent	supersets of infrequent, closure subsets except generator	descendants of infrequent, descendants of child with same support as parent
support counting primitive	aggregated	increment by 1	aggregated
intended itemsets	all frequent	frequent closed	frequent closed
data access when counting support	in memory	linear scans	in memory
typical disk scan number	2	$< N$	2

Table 2.4: Selected characteristics of algorithms (N is the size of the largest frequent set, E is the set-enumeration-tree)

Table 2.4 resumes the selected characteristics of frequent set mining algorithms presented in the state-of-the-art (**CLOSET** is going to be described). Each of these algorithms can be summarized in a concise way as following:

- **APRIORI**: basic, introduces **APRIORI** trick (pruning strategy), efficient join-based candidate generation and simultaneous support counting for all candidates of a same size,
- **DIC**: support counting and data set scan number optimizations based on better-timed candidate generation,
- **Max-Miner**: (positive) border of frequent sets mining only,
- the **sampling algorithm**: frequent set collection guess and correct, data set scan number optimization, simultaneous support counting for candidates of all sizes,
- **FP-Growth**: depth-first, aggregating rows to make amassed support increments,
- **CLOSE**: mining an equivalent, less voluminous information, deriving all frequent sets in a simple and very efficient way.

An interesting remark from this comparative critique of those algorithms is that among the ideas presented in this summary some are complementary and might be combined. The most promising is to consider a depth-first, row aggregating method to mine frequent closed sets, which could be superior to any of them separately. It has been recently considered by Pei *et al.* in [Pei 00], and the resulting algorithm, called **CLOSET**, is briefly presented below.

2.2.8 CLOSET

CLOSET [Pei 00] computes frequent closed sets and their corresponding supports using a technique similar to **FP-Growth**. It explores the search space in a similar, depth-first manner and uses a complete set-enumeration-tree (denoted E in the following). An itemset is considered as candidate if its parent itemset is frequent, but an additional requirement is that the child's support is different (i.e., strictly lesser) from the parent's.

Algorithm 10 shows the top-level pseudo-code of **CLOSET**. The pseudo-code represents a recursive function $FindFCS$. A call $FindFCS(X, r', \sigma, E)$ finds in r' the frequent closed sets (and their supports) in a well-defined partition of the search space — the function will consider only the part of E corresponding to X and its descendants in E. This is a subtile difference with $FindFS$ of **FP-Growth**, which considered that X itself is processed by the function calling $FindFS$.
Since $\emptyset$ is the root of E, all itemsets are considered by the call $FindFCS(\emptyset, r, \sigma, E)$, so it returns $\mathcal{FC}$ and we have $FreqClosed(r, \sigma) \subseteq \mathcal{FC}$. A second step (not represented in

Algorithm 10) must remove all non-closed sets from $\mathcal{FC}$. It can be done by testing for inclusion every pair of itemsets in $\mathcal{FC}$, and if two itemsets have the same support and one is a proper subset of the other, the former is not closed.
In line 1, the algorithm generates the candidate itemsets for the following scan (line 2) of data. Like **FP-Growth**, **CLOSET** stores the binary database r' in memory, so the scan can be implemented relatively efficiently. In line 3, it initializes a temporary variable for the result (returned in line 10). The value stored in the buffer is the set containing a pair itemset-support: the first element is a possibly incomplete closure of X, the second is the support of X (and thus of its closure). The closure is complete only if X is an ancestor of its own closure in E or if X itself is its own closure. Since for each closed set there is at least one such an X, it follows that the algorithm is complete (unsound elements are removed in a second step mentioned above). Then, it considers successively all children Y of the entry point node (i.e., of X) in line 4. If Y is frequent, but not included in the closure of X (line 5) descendants of Y are considered (recursive call in line 6) and all frequent closed sets from the sub-tree rooted at Y are stored in the buffer (line 7).
Similar optimizations to the ones used in **FP-Growth** are integrated in **CLOSET** aiming at aggregating the rows when it can be done without loosing any relevant information (see **FP-Growth**).

Summarizing, **CLOSET** is a technique marrying the most promising ideas in frequent set mining tools of the past. It will be our reference algorithm for depth-first methods. Typically, on highly correlated data **CLOSET** should be the most efficient among the

Algorithm 10. (CLOSET)

Function $FindFCS(X, r', \sigma, E)$
Input: *X the itemset considered as the current starting point in the search space, r a binary database over a set of items R, σ an absolute support threshold, and E is a complete set-enumeration-tree.*
Output: *All pairs $<Y, Sup(r', Y)>$ such that Y is frequent and closed in r' and it is in the sub-tree of E rooted at X.*

1: **let** $\mathcal{C} := \{Y : Y \in children\ of\ X\ in\ E\}$;
2: *Scan r' and compute supports of itemsets in $\mathcal{C}$;*
3: $\mathcal{FC} := \{<X \cup \bigcup_{Y \in \mathcal{C}} \{Y : support\ of\ Y = |r'|\}, |r'|>\}$;
4: **for all** $Y \in \mathcal{C}$ **do**
5: **if** *support of Y in $r' \in [\sigma, |r'|)$* **then**
6: $\mathcal{FC}' := FindFCS(Y, \mathcal{M}(r', Y), \sigma, E)$;
7: $\mathcal{FC} := \mathcal{FC} \cup \mathcal{FC}'$;
8: **fi**
9: **od**
10: **return** $\mathcal{FC}$;

ones we presented, however in this thesis, we will also show that on some data sets and typical conditions of experiments, **CLOSE** outperforms **CLOSET**.

2.2.9 Note on the Implementations Used in the Experiments

For experiments we implemented **CLOSE** and **CLOSET**, two mining tools computing frequent closed sets. This is not only because the use of frequent closed sets leads to tools that are fastest on highly correlated large data sets, but also because the idea of mining only some frequent sets (here frequent closed sets), instead of all frequent sets presents some important advantages. This idea will be more formally stated as condensed representation framework and developed through this thesis.
Additionally, we implemented **APRIORI**, to see if the use of frequent closed sets has an adverse effect on performances in the circumstances of our experiments.
We notice however that the implementations of all algorithms (including **HLin-Ex** and **VLin-Ex**, which will be introduced in Chapter 3) use the same low-level data structures and techniques, in order to ensure a fair comparison. All prototypes have been implemented in C++, and a similar effort has been spent on specific fine-tuning of each of them.
Our implementations have been cross-validated with independently realized tools[22] and results[23], as well as cross-validated between the tools implemented by ourselves. They pass all these tests — we consider them as reliable and the presented behavior in experiments as typical.

2.3 Future Direction: Investigating Condensed Representations

Condensed representations have been actually used in the computer science for long, even if the people using them do not explicitly call them that way. Let consider informally the concept.
A condensed representation is such an alternative form of data and procedures manipulating them that from the external point of view the manipulating the alternative form of data is easier than of the corresponding original form of data (e.g., it is more pertinent to inspect or more efficient to obtain). Additionally, we wish that there were little or no loss of quality of the result (comparing to the more difficult processing of the original form of data).

[22]Thanks to Prof. H. Toivonen (from the University of Helsinki) who provided an implementation of **APRIORI** and to T. Daurel (from INSA-Lyon and Schlumberger-Clamart) who provided an implementation of **ACE3**, a specialized association rule extractor.
[23]We are grateful to the Prof. L. Lakhal's research group from the University of Clermont-Ferrand for having sent us relevant outputs of their implementation of **CLOSE**.

This informal definition is quite broad, let us take some examples.

Example 6. *In Section 2.1.1 we introduced the concept of functional dependencies (FD) across a data set. We say that the FD* $\{A_1, A_2, \ldots, A_n\} \rightarrow \{B_1, \ldots, B_k\}$ *holds when for every pair of data points* r*,* q *such that the value of attribute* A_i *is the same for both points, and that for all* $i \in \{1, \ldots, n\}$*, they also have the same corresponding values for all attributes* $B_1, \ldots, B_k$*.*
Now, observe that if the above rule holds, and two data points r *and* q *have the same values (pairwise) of all attributes* A_i*, they will have the same values (pairwise) of all attributes* B_i*, and that independently of the values of remaining attributes. So we may add an attribute* A_{n+1} *to the left-hand-side of the dependency and still be sure that any* r *and* q *having the same values (pairwise) of all attributes* A_i *(now including* A_{n+1}*) will have the same* B_i*s. This latter dependency is a logical consequence of the former, and thus is redundant.*
Therefore, computer scientists tend to use a non-redundant cover of a collection of FDs. The can be handled easier than the collection of all FDs. A non-redundant cover c *of a collection of FDs* s *(see [Bernstein 76], the term non-redundant covering of a collection of FDs is used) is the collection of FDs that permits to logically infer the same FDs as from the original collection* s*, but none of the proper sub-collections of* c *does.*
In this case, the set of all FDs is the original collection and the FD cover is its condensed representation.

The functional dependency cover is typically orders of magnitude smaller that the represented collection of functional dependencies, so it makes sense to call the cover a *condensed* representation of all FDs.
The following example recalls another typically used trick, in this case to simply increase performances.

Example 7. *Trigonometric functions* $sin(x)$ *and* $cos(x)$ *are quite costly to evaluate, in general. However, they are periodic and they vary little over a small range. Thus one could sample their values over one period (e.g.,* $[0, 2\pi)$*) and store them in tables (one for each function). When the precision is not much a concern (rapid animations using sine and cosine to define a trajectory), the sampling rate could be actually quite rough, e.g., only one value per angular degree (resulting in an array of 360 reals). Then, all input values can be transformed to integer degrees and when their sine or cosine is needed, the value may be read from the corresponding table. In general, earlier that transformation is done, better for the performances.*

Remark that in the latest example, the original representation is a real number and its "condensed" counterpart is a small integer from $\{0, 1, \ldots, 359\}$. The advantages related to the performances might be interesting and the "condensation ratio", depending on

the chosen representation for numbers, will typically be limited to a small (constant) factor. Though interesting, the example shows a rather modest advantages. Therefore, the framework is rather proposed for condensing large collections.
The last example (given later) will show a second (after FDs) application of the idea to large collections, possibly resulting in a high degree of condensation.

The use of condensed representations in data mining has been suggested in [Mannila 97a] as a promising direction of research. Such condensed representations could be gathered off-line before the interactive step of the mining process. In that case, the goal is to improve the step of processing involving humans (querying, pattern assessment, ...).
Other promissing promising directions of research have been proposed and investigated to acheive these goals. On-line behavior can be improved, e.g., reusing the result of one query for obtaining the next result in a way different from re-computing the whole answer from the data set (see e.g., [Diop 02, Bykowski 02b]). Such approaches can greatly benefit from "similarity" of successive queries to reduce the loads.
Also combinations of the above ideas are recently considered, e.g., [Jeudy 02, Giacometti 02]. In this thesis, we focus on promising condensed representations.

The condensed representations can be divided into lossless representations and representations with loss. Example 6 corresponds to the lossless class, Example 7 is a representation with loss (a bounded error).
One of the existing frameworks for representations with loss is the framework of ϵ-adequate representations. We recall it after Mannila and Toivonen [Mannila 96b] and exemplify on itemset support queries with an alternative answer mode using the collection of frequent sets and their supports.

Definition 22. (ϵ-adequate representation) An ϵ-adequate representation for S w.r.t. a class of queries $\mathcal{Q}$, is a class of structures C, a representation mapping $rep : S \rightarrow C$ and a query evaluation function $m : \mathcal{Q} \times C \rightarrow [0,1]$ such that $\bigwedge_{Q \in \mathcal{Q}} \bigwedge_{s \in S} |Q(s) - m(Q, rep(s))| \leq \epsilon$.

Table 2.5 describes all components of an ϵ-adequate representation and illustrates them with a complete example. Remark that in that example the original representation is a binary relation and its "condensed" counterpart is a collection of itemset-support pairs. The "condensation ratio" depends on the instance of binary matrix r and on σ the support threshold. For a given r, higher ϵ generally leads to more "condensed" representation (note that in the example in Table 2.5 the support threshold σ used to mine frequent sets is proportional to ϵ), but also to higher accepted errors. In extreme cases (e.g., a binary database consisting of a single row), the condensed form may actually be larger than the original one.

Note that by setting ϵ to 0, we define a framework for lossless representations.

Component	Description	Example
S	S is a class of structures, class of original representations.	A class of structures is for example the set noted $\mathcal{DB}_R$ of all possible binary databases over the set of items R.
$\mathcal{Q}$	$\mathcal{Q}$ is a class of queries for S. The value of a query $Q \in \mathcal{Q}$ on a structure $s \in S$ is assumed to be a real number in $[0,1]$ and is denoted $Q(s)$.	For a given set of items R, an example class of queries is $\mathcal{Q}_R$, the set of all queries retrieving the relative support of an itemset (subset of R). If we denote Q_X the query in $\mathcal{Q}_R$ asking for the relative support of the itemset X then $\mathcal{Q}_R = \{Q_X : X \subseteq R\}$ and the value of Q_X on a database instance $r \in \mathcal{DB}_R$ is defined by $Q_X(r) = Sup(r,X)/\|r\|$. $Sup(r,X)/\|r\|$ effectively are in $[0,1]$.
C	The class of alternative representations.	C is the set of all possible collections of frequent sets and their supports, i.e., $C = \{rep(r) : r \in \mathcal{DB}_R\}$.
rep	Permits to obtain the contents of the alternate form from the original form and is given in terms of a representation change function $rep : S \to C$.	We can represent a binary database for itemset support queries using frequent sets and their supports, i.e., $\bigwedge_{r \in \mathcal{DB}_R} rep(r) = FreqSup(r, \epsilon * \|r\|)$.
m	The query evaluation function m on the alternative representation, i.e., on the elements of C. It is a function that for an individual query Q and a stored value c returns an approximate value of $Q(s)$, where c is the transformed value of s. Formally, $m : \mathcal{Q} \times C \to [0,1]$ and $m(Q, rep(s))$ must not differ by more than ϵ from $Q(s)$ for every s.	Using this representation, the query Q_X returns α corresponding to $<X,\alpha>$ if such a pair is in C, 0 otherwise. It is easy to prove that this is an ϵ-adequate representation for $\mathcal{DB}_R$ w.r.t. $\mathcal{Q}_R$ since $\bigwedge_{Q_X \in \mathcal{Q}_R} \bigwedge_{r \in \mathcal{DB}_R} \|Q_X(r) - m(Q_X, rep(r))\| \leq \epsilon$.
ϵ	The error bound. Described together with the previous component.	Fixed by the user.

Table 2.5: ϵ-adequate representation component by component.

Chapter 3

Core of the Thesis

3.1 Original Framework of Condensed Representations

In this section, we motivate and propose a framework for condensed representations of interesting patterns. Since this thesis is focused on frequent sets, most of examples we give are issued from this particular research field. Further, we propose to abstract into this framework the existing solutions that compute alternative representations of frequent sets. It should be noticed however that the applicability of the proposed framework is not limited to frequent sets — it may be used for collections of other interesting patterns.

3.1.1 Problem Statement

In this section, we briefly introduce a specific framework for handling the collections of interesting patterns, namely the so-called framework of theories.
Let the following parameters be given: r a data set, $\mathcal{L}$ the set of all possible sentences for expressing the properties and q a selection predicate, which for a given sentence $\phi \in \mathcal{L}$ establishes if it is potentially interesting w.r.t. the data set r.
The theory $\mathcal{T} = Th(\mathcal{L}, r, q)$ is then the set of all sentences that are judged potentially interesting w.r.t. the data set r [Mannila 97b]. Of course, the judgment depends on the context of the knowledge discovery task (e.g., the intended application).
In [Mannila 97b], the reader can find many interesting results concerning theory computation when the selection predicates are *anti-monotone*[1] w.r.t. a generalization/specialization $\preceq$ relation between sentences of $\mathcal{L}$.

Definition 23. (anti-monotonicity) A predicate q is anti-monotone w.r.t. a generalization/specialization $\preceq$ iff $\bigwedge_{\phi, \psi \in \mathcal{L}} \phi \preceq \psi \wedge q(\psi) \Rightarrow q(\phi)$, i.e., if and only if for every sentence asserted by q more general sentences are also asserted by q.

The sentences of the language $\mathcal{L}$ expressing the properties will be called *patterns*. For

[1] In [Mannila 97b] the term *monotone* is used for the same concept

patterns that are accepted by the selection predicate q, we will use the term *interesting patterns*. [Mannila 97b] describes few example classes of patterns.
Now, we illustrate the use of the framework on frequent sets ($Freq(r, \sigma)$ and $FreqSup(r, \sigma)$).

Example 8. *Let us consider the collection of all frequent sets $\mathcal{T}_F$. Here, r is a single binary relation over a set of attributes R, $\mathcal{L}_F$ is the collection of all itemsets (subsets of R), and $q_F(r, X) \equiv Sup(r, X) \geq \sigma$ (q_F is the predicate* "X is σ-frequent in r"*). $\mathcal{T}_F = Th(\mathcal{L}_F, r, q_F)$ is the collection of all frequent itemsets.*

The following example incorporates into a pattern the value of the evaluation function (usually not a part of the pattern iteself).

Example 9. *Deriving correctly supports and confidences of association rules requires for each frequent set its support. The adequate patterns should take into account this connection. Here, r is a relation over binary attributes R, $\mathcal{L}_{FS}$ is the collection of pairs $<X, S_X>$, such that X is an itemset (subset of R) and S_X is an integer. Let $q_{FS}(r, <X, S_X>) \equiv Sup(r, X) = S_X \wedge S_X \geq \sigma$, i.e., the predicate stating that X is frequent and S_X is the support of X in r. $\mathcal{T}_{FS} = Th(\mathcal{L}_{FS}, r, q_{FS})$ is the collection of all frequent sets and their corresponding supports in r.*

$\mathcal{T}_{FS}$ is not anti-monotone w.r.t. itemset inclusion. Indeed, $q_{FS}(r, <Y, 100>)$ and $X \subseteq Y$ does not necessarily implies that, for example, $q_{FS}(r, <X, 99>)$ holds. If $q_{FS}(r, <Y, 100>)$ and $X \subseteq Y$, the consequence actually cannot hold, because the support of the subset of Y cannot be lower than the support of Y itself.
Assuming that for each itemset we have its exact support (e.g., we count it during a scan), we could however improperly say that the selection predicate is anti-monotone.

The difficulty of pattern mining tasks that turn out to be the hardest is due to any or a conjunction of the following characteristics:

- target pattern collections are large or infinite,
- evaluating the selection predicate on a pattern or on a set of patterns is costly.

Large pattern collections often occur when one mines frequent sets with a very low support threshold within a binary database r. E.g., if the support threshold is fixed at 1 row and the longest row of the database r has 100 items, then the number of frequent sets is at least $2^{100} \approx 10^{30}$, which is clearly out of the present-day ability, not even to compute or to store, but simply to enumerate them.
Evaluating the selection predicate is costly, for example, when the data set r is huge, say in Giga- or Terabyte orders of magnitude, and we need to scan it entirely to evaluate the selection predicate.

3.1.2 Proposed Remedies

Repetitive evaluating of the selection predicate q for numerous (candidate) patterns is often considered as the bottleneck of the mining task.
On one hand, some of proposed remedies aim at optimizing the evaluation of selection predicate or the number of scans of the data set (e.g., in case of frequent set mining algorithms, we may cite **DIC**, **FP-Growth**, both presented in Section 2.2). On the other hand, we observe that some recent advances do not aim at improving the execution time of the selection predicate on a pattern, but focus rather on mining different pattern collections (e.g., **Max-Miner**, **CLOSE**, also presented in Section 2.2). According to their authors, these collections are somehow more pertinent w.r.t. a goal.
Let us focus now on the solutions based on different pattern collections, and denote $\mathcal{C}$ such a pattern collection and $\mathcal{T}$ the target collection of patterns.
A reconstruction step to *infer* the target patterns from $\mathcal{C}$ is necessary. It is implicit in some papers (e.g., in [Bayardo, Jr. 98] for a condensed representation of frequent sets or in [Pasquier 99a] for a condensed representation of association rules) or explicitly stated in the others (e.g., in [Pasquier 99c] for a condensed representation of frequent sets). Since no additional evaluations of the selection predicate (against the data) are necessary, this inference is often very fast, and may be done on the fly, when $\mathcal{C}$ is available and $\mathcal{T}$ is needed.

Example 10. *Theorem 2 permits to derive all frequent sets (but not their supports) from the positive border of all frequent sets.*

Example 11. *Algorithm 9 shows how to derive all frequent sets and their corresponding supports from the collection of all frequent closed sets along with their supports.*

It is important to know how to infer $\mathcal{T}$ from $\mathcal{C}$. In some cases, the authors rely on the intuition and do not prove the preservation of the full information (e.g., in [Pasquier 99a, Zaki 00], both concern association rules). In extreme cases, this leads to design flaws (as shown e.g., in [Cristofor 02], the condensed representation of association rules proposed in [Pasquier 99a] leads to unsound inferred answers).
In the following, we propose a simple formalisation of condensed representations of the pattern theories in the context given in Section 3.1.1. Here, we focus only on exact representations, i.e., collections that aim at representing correctly all interesting patterns. We will show that we can model in a simple and unified framework some recent advances in pattern mining that make use of inference.

3.1.3 The Proposed Framework

Let the parameters $\mathcal{L}, r, q$ given for the theory $\mathcal{T} = Th(\mathcal{L}, r, q)$ be defined as previously. Here, we additionally consider the functions $InferentialClosure$ and q_c. q_c selects the

sentences of $\mathcal{C}$ that make up a *condensed representation of* $\mathcal{T}$, while $InferentialClosure$ infers $\mathcal{T}$ from $\mathcal{C}$.
Nothing forbids to use a different kind of patterns ($\mathcal{L}_\mathcal{C}$) for the condensed representation, but the instances of the framework in this thesis use the same pattern domain for condensed and original theory ($\mathcal{L}_\mathcal{C} = \mathcal{L}$). Formally, $InferentialClosure$ is a function from $2^{\mathcal{L}_\mathcal{C}}$ to $2^{\mathcal{L}}$ and we are interested in correct representations, i.e., representations such that $\mathcal{T} = InferentialClosure(\mathcal{C})$ for any instance r.
The *condensed theory of* $\mathcal{T}$ is then the pair $<\mathcal{C}, InferentialClosure>$ with $\mathcal{C} = Th(\mathcal{L}_\mathcal{C}, r, q_c)$ and $\mathcal{T} = InferentialClosure(\mathcal{C})$.

3.1.4 Practical Issues about Components of Condensed Theories

Elements of the condensed theory, i.e., $\mathcal{L}_\mathcal{C}, q_c, InferentialClosure$ may be chosen depending on the pattern mining context, data owner's domain knowledge, redundancy observations, model assumption, etc. One should care as well about being able to implement efficiently q_c and $InferentialClosure$, if the goal is to improve the overall efficiency of mining the full theory. Choosing above parameters in a particular context is a multi-step process. It may be inspired from past advances recast into this framework in Section 3.1.5 as well as from the original condensed representations presented later in this thesis (Sections 3.2.4 and 3.3.4).
Let $\mathcal{P}$ be a collection of patterns. Here, we are only concerned with consistent arguments $\mathcal{P}$ of the function $InferentialClosure$, i.e., there must exist an instance r of a dataset such that all and only patterns in $\mathcal{P}$ validate the selection predicate q_c. Readers interested in the topic of consistency of pattern collections may find in [Bykowski 02b] a method for checking the consistency of a set of Boolean formulae together with their corresponding supports within the context of binary databases.
An important property is the inferential monitonicity of function $InferentialClosure$, stating that one infers at least the same properties given additional assumptions. It is not required by the framework, but it may sometimes influence the implementation of the full pattern derivation ($InferentialClosure$ function).

3.1.5 Instanciating the Framework on the Recent Approaches

We limit the discussion to the recent approaches concerning frequent sets, already presented in Section 2.2. The re-examination of these approaches and instanciating the framework is conducted in four steps (denoted ①, ②, ③ and ④).
① corresponds to identifying what is the theory $\mathcal{T}$ that the approach aims to condense and what are the parameters $\mathcal{L}, r, q$ of $\mathcal{T}$.
② expresses what the authors consider as redundancy within the collection of patterns $\mathcal{T}$.

③ gives $\mathcal{L}_{\mathcal{C}}$ and $InferentialClosure$ needed to reconstruct the theory $\mathcal{T}$.
④ defines q_c and combines all parameters to form the condensed representation as $\mathcal{C} = Th(\mathcal{L}_{\mathcal{C}}, r, q_c)$.

Max-Miner [Bayardo, Jr. 98] and Pincer Search [Lin 98]

① We consider the collection of all frequent sets ($\mathcal{T}_F$) with its corresponding parameters: $\mathcal{L}_F$ (all itemsets), r (a binary database) and q_F (the selection predicate "is σ-frequent in r"), defined in Example 8 (Section 3.1.1).
② The use of Theorem 2 is proposed to condense this theory. From that theorem, we know that if $X \in \mathcal{T}_F$ and $Y \subset X$, then $Y \in \mathcal{T}_F$. Therefore, outputting any proper subset of a frequent set turns out to procuce a redundant statement[2].
③ It is clear that the non-redundant frequent sets are the ones with no frequent proper superset. Thus, only the maximal frequent sets w.r.t. itemset inclusion are non-redundant. If these maximal frequent sets are adopted as the condensed representation, then it will consist of a collection of itemsets, therefore $\mathcal{L}_{\mathcal{C}F} = \mathcal{L}_F$.
The result of $InferentialClosure_F$ applied to some collection of patterns $\mathcal{C}$ simply produces all subsets of the itemsets in $\mathcal{C}$. Formally,

$$InferentialClosure_F(\mathcal{C}) = \{Y \in \mathcal{L}_F : \bigvee_{X \in \mathcal{C}} Y \subseteq X\}.$$

④ The condensed representation selection predicate q_{cF} is thus $q_{cF}(r, X) \equiv Sup(r, X) \geq \sigma \wedge \bigwedge_{Y \supset X} Sup(r, Y) < \sigma$. It is easy to prove that $\mathcal{T}_F = InferentialClosure_F(\mathcal{C}_F)$, where $\mathcal{C}_F = Th(\mathcal{L}_{\mathcal{C}F}, r, q_{cF})$.
The reconstruction algorithm to generate $\mathcal{T}_F$ from $\mathcal{C}_F$ is trivial.

CLOSE [Pasquier 99c]

① We consider $\mathcal{T}_{FS}$, the collection of all frequent sets along with their supports, redefined in the context of pattern theories as Example 9 (Section 3.1.1). The parameters corresponding to $\mathcal{T}_{FS}$ are: $\mathcal{L}_{FS}$ (all pairs itemset-support), r (a binary database) and q_{FS} (detailed in Example 9).
② In case of long frequent sets, very often we can observe lot of itemsets with the same value of support. In the case where two itemsets X, Y are included one in the other, say $Y \subset X$, the same support of both is an evidence of an association rule with confidence equal to 1 (note that $Conf(r, Y \Rightarrow X \setminus Y) = Sup(r, X)/Sup(r, Y) = 1$). Such rules often hold in data.
Thus an itemset may have the support equal to the support of one of its supersets. All itemsets with a proper superset having the same support can be considered redundant.

[2]Remark that in this context we search for frequent sets only (without the corresponding supports).

③ Non-redundant patterns (w.r.t. the redundancy defined above) are the pairs itemset-support such that no proper superset of the itemset has the same support. Thus, the pair $<X, S_X>$ is non-redundant if among itemsets having the same support S_X, X is maximal w.r.t. itemset inclusion. So-defined pairs are further limited to frequent ones to represent frequent sets only.
The resulting condensed representation will consist of a collection of pairs itemset-support, therefore $\mathcal{L}_{\mathcal{C}FS1} = \mathcal{L}_{FS}$.
The $InferentialClosure_{FS1}$ function is defined as:

$$InferentialClosure_{FS1}(\mathcal{C}) = \{<Y, S_Y> \in \mathcal{L}_{FS} : \bigvee_{<X, S_X> \in \mathcal{C}} \Big(Y \subseteq X \wedge S_Y = S_X \wedge \neg \bigvee_{<Z, S_Z> \in \mathcal{C}} Y \subseteq Z \wedge Z \subset X\Big)\}.$$

For each pair $<X, S_X>$, the function outputs every subset Y of X with S_X as support, as long as there is no pair $<Z, S_Z>$ in $\mathcal{C}$ with $Z \subset X$, such that Y is a subset of Z. Without this second condition, supports of different supersets of an itemset Y would have been propagated on Y, creating possible conflicts.
④ The condensed representation selection predicate is thus
$q_{cFS1}(r, <X, S_X>) \equiv S_X = Sup(r, X) \wedge S_X \geq \sigma \wedge \bigwedge_{Y \supset X} Sup(r, Y) < S_X$.
All and only frequent closed sets (defined in Section 2.2 together with the algorithm **CLOSE**) are selected along with their supports.
An efficient recontruction of $\mathcal{T}_{FS}$ from $\mathcal{C}_{FS1}$, where $\mathcal{C}_{FS1} = Th(\mathcal{L}_{\mathcal{C}FS1}, r, q_{cFS1})$ was also described in Section 2.2. The corresponding algorithm (**Freq-Closed-To-Freq**) could be interpreted as an implementation of $InferentialClosure_{FS1}(\mathcal{C}_{FS1})$.

3.2 Original Condensed Representation Based on Frequent δ-Free Sets

In this section, we consider an original condensed representation based on frequent δ-free sets. We describe it first since it has been proposed earlier [Boulicaut 00a, Boulicaut 00c, Boulicaut 03]. In the following, we describe a small modification of this representation, in order to make both original condensed representations of the thesis more uniform. This representation is called *strong-rule-bordered condensation.*
After informal and formal parts defining this representation, we will abstract it into a framework for representations with loss, namely ϵ-adequate representations. ϵ-adequate representation based on strong-rule-bordered condensation is more condensed than the ϵ-adequate representation based on frequent sets (see Table 2.5).
We show that a restricted variant of this condensed represenation may be abstracted into the original framework for exact represenations proposed in Section 3.1.

3.2.1 Informal Introduction

The strong-rule-bordered condensation is based on itemsets called frequent δ-free sets. It offers two possibilities: a full equivalence to frequent sets or a possible small loss of the information, corresponding respectively to $\delta = 0$ and $\delta > 0$. In the former case, frequent closed sets and frequent δ-free sets (with $\delta = 0$) present similar benefits. Frequent δ-free sets with $\delta > 0$ lead to a much smaller representation and to a more efficient extraction, but at the cost of some uncertainty on the supports of regenerated itemsets.

The key intuition of the strong-rule-bordered condensation (SRBC for short) is illustrated on the following example. Let A, B, C, D represent binary attributes in a relational table r depicted in Table 3.1 and suppose that we are interested in the support of $\{A, B, C\}$ in r. If we know that the rule $\{A, B\} \Rightarrow \{C\}$ nearly holds in R (i.e., when A and B are set to true in a row then, excepted in a few cases, C is also true) then we can approximate the support of itemset $\{A, B, C\}$ using the support of $\{A, B\}$. In Table 3.1, the rule $\{A, B\} \Rightarrow \{C\}$ has only one exception. So, we can use the support of $\{A, B\}$ as an approximated value for the support of $\{A, B, C\}$. Moreover, we can approximate the support of any itemset X such that $\{A, B, C\} \subseteq X$ by the support of $X \setminus \{C\}$ because the rule $X \setminus \{C\} \Rightarrow \{C\}$ also holds with a few exceptions. For instance, the support of $X = \{A, B, C, D\}$ can be approximated by the support of $\{A, B, D\}$ since the rule $\{A, B, D\} \Rightarrow \{C\}$ can not have more exceptions than $\{A, B\} \Rightarrow \{C\}$. Furthermore, the support of $\{A, B, D\}$ does not need to be known exactly, but can also be itself approximated. For example, the rule $\{A, D\} \Rightarrow \{B\}$ holds in Table 3.1 with one exception, so the support of $\{A, D\}$ can be used as an approximation of the support of $\{A, B, D\}$ and then also of the support of $\{A, B, C, D\}$.

We call δ-free set an itemset Y such that the items in Y can not be used to form a nearly exact rule. If we consider only rules having at most one exception, then the only 1-free sets in Table 3.1 are $\{\emptyset, \{A\}, \{B\}, \{C\}, \{D\}, \{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}, \{C, D\}\}$. The δ-freeness of itemsets is anti-monotonic in the sense that if a set is not a δ-free set then none of its supersets can be a δ-free set. The algorithm we disgned to extract the δ-free sets takes advantage of this property. It considers first itemsets of size 1 (i.e., the itemsets containing 1 item), then in the next iteration sets of size 2, and so on. When it determines that a set X is not δ-free then it prunes the search space since there is no need to consider any of the supersets of X. For example, if the algorithm is executed on Table 3.1 and takes into account rules having at most one exception, then it will never consider the set $\{A, B, C, D\}$ because several sets among its subsets are not 1-free (e.g., $\{A, B, C\}$).

Experiments (presented in Section 3.2.6) show that *frequent* δ-free sets can be extracted efficiently, even on highly-correlated data sets. Frequent δ-free sets are not sufficient to

A	B	C	D
x	x		
x			x
	x	x	x
x		x	x
x	x		x
x	x	x	
x			
x			
	x		
	x		
		x	
		x	
			x
			x
x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x

Table 3.1: A relational table over a set of four items ("x" denotes the presence of the item in the row).

reconstruct frequent sets. The negative border of frequent δ-free sets[3] is essential to define the alternate representation. The SRBC is made up of itemsets that are either frequent δ-free sets or in the negative border of frequent δ-free sets.

3.2.2 Formalizing SRBC

We introduce the concept of δ-free set in a concise way.

Definition 24. (δ-strong rule) A *δ-strong rule*[4] over Z in a binary database r is an association rule $X \Rightarrow Y$ over Z such that $Sup(r, X) - Sup(r, X \cup Y) \leq \delta$, i.e., the rule is violated in no more than δ rows.

In this definition, δ is supposed to have a small value, so a δ-strong rule is intended to be a rule with very few exceptions. Also important is that an association rule has a nonempty right-hand-side, so in this definition $Y \neq \emptyset$.

Definition 25. (δ-free set) Let r be a binary database over R, $X \subseteq R$ is a *δ-free set* w.r.t. r if and only if there is no δ-strong rule over X in r. The set of all δ-free sets w.r.t. r is noted $DeltaFree(r, \delta)$.

[3]We instantiate the notion of negative border recalled in Definition 4.

[4]Stemming from the notion of *strong rule* of [Piatetsky-Shapiro 91].

Since δ is supposed to be rather small, informally, a δ-free set is a set of items such that its subsets (seen as conjunctions of properties) are not related by any very strong positive correlation.

Lemma 6. *Let r be a binary database over R, X be an itemset such that $X \subseteq R$, A be a single positive item in $R \setminus X$ and $\overline{A}$ its corresponding negative item.*
Then the following two propositions are equivalent:
(1) $Sup(r, X) - Sup(r, X \cup \{A\}) \leq \delta$,
(2) $|\mathcal{M}(r, X \cup \{\overline{A}\})| = Sup(r, X \cup \{\overline{A}\}) \leq \delta$.

Proof. Observe that $\mathcal{M}(r, X) = \{t : t \in \mathcal{M}(r, X \cup \{A\}) \vee t \in \mathcal{M}(r, X \cup \{\overline{A}\})\}$ and that the multisets $\mathcal{M}(r, X \cup \{A\})$ and $\mathcal{M}(r, X \cup \{\overline{A}\})$ are mutually exclusive. Therefore $Sup(r, X) = Sup(r, X \cup \{A\}) + Sup(r, X \cup \{\overline{A}\})$. Consequently, *(1)* is equivalent to $Sup(r, X \cup \{\overline{A}\}) \leq \delta$. □

Lemma 7. *Let r be a binary database over R, and X, Y be itemsets such that $Y \subseteq X \subseteq R$. Let $A \in Y$. If $Y \setminus \{A\} \Rightarrow \{A\}$ is a δ-strong rule in r then $X \setminus \{A\} \Rightarrow \{A\}$ is also a δ-strong rule in r.*

Proof. Suppose that $Y \setminus \{A\} \Rightarrow \{A\}$ is a δ-strong rule in r. By Definition 24, we have $Sup(r, Y \setminus \{A\}) - Sup(r, Y) \leq \delta$. By Lemma 6, $Sup(r, (Y \setminus \{A\}) \cup \{\overline{A}\}) \leq \delta$. Then, since X is a superset of Y, by Lemma 2, $\delta \geq Sup(r, (Y \setminus \{A\}) \cup \{\overline{A}\}) \geq Sup(r, (X \setminus \{A\}) \cup \{\overline{A}\})$. Using again Lemma 6, we conclude that $X \setminus \{A\} \Rightarrow \{A\}$ is also a δ-strong rule. □

Lemma 8. *Given r a binary database over R, X is a δ-free set w.r.t. r iff* $\bigwedge_{A \in X} Sup(r, X \setminus \{A\}) - Sup(r, X) > \delta$.

Proof. Let us prove the equivalence of negations.

First, suppose that there exists $A \in X$ such that $Sup(r, X \setminus \{A\}) - Sup(r, X) \leq \delta$. By Definition 24, we deduce that $X \setminus \{A\} \Rightarrow \{A\}$ is a δ-strong rule, which proves that X is not a δ-free set w.r.t. r.

For the equivalence, suppose that X is not δ-free w.r.t. r, i.e., $\bigvee_{Y \subseteq X} \bigvee_{Z \subset Y} Z \Rightarrow Y \setminus Z$ is a δ-strong rule in r. Let Y and Z be such itemsets as in the expression. By Definition 24, we have $Sup(r, Z) - Sup(r, Y) \leq \delta$.

Let us consider $A \in Y \setminus Z$. $Z \cup \{A\} \subseteq Y$ implies $Sup(r, Z \cup \{A\}) \geq Sup(r, Y)$. It means that $\delta \geq Sup(r, Z) - Sup(r, Y) \geq Sup(r, Z) - Sup(r, Z \cup \{A\})$, i.e., $Z \Rightarrow \{A\}$ is a δ-strong rule.

By Lemma 7, $X \setminus \{A\} \Rightarrow \{A\}$ is also a δ-strong rule, because $Z \subseteq X \setminus \{A\}$. Thus $Sup(r, X \setminus \{A\}) - Sup(r, X) \leq \delta$. □

One of the most interesting properties of *δ-freeness* is its *anti-monotonicity* w.r.t. itemset inclusion (see Definition 23).

The anti-monotonicity has been identified as a key property for efficient pattern mining [Mannila 97b, Ng 98], since it is the formal basis of a safe pruning criterion. Indeed, efficient frequent set mining algorithms like **APRIORI** and **DIC** (presented in the previous chapter) make use of the (anti-monotone) property *"is frequent"* for pruning.
The anti-monotonicity of δ-freeness follows directly from the definition of δ-free sets and is stated by the following theorem.

Theorem 3. *Let X be an itemset. For all $Y \subseteq X$ if $X \in DeltaFree(r, \delta)$ then $Y \in DeltaFree(r, \delta)$.*

We show now that δ-free sets can be used to answer support queries with a bounded error. The following lemma states that the support of any itemset can be approximated using the support of one of the δ-free sets.

Lemma 9. *Let r be a binary database over a set of items R, $X \subseteq R$ and $\delta \in [0, |r|]$, then there exists $Y \subseteq X$ such that $Y \in DeltaFree(r, \delta)$ and $Sup(r, Y) \geq Sup(r, X) \geq Sup(r, Y) - \delta|X|$.*

Proof. We show this using a recurrence on $|X|$. The statement is true for $|X| = 0$ if we take $Y = \emptyset$ (the empty set is trivially δ-free). Suppose the statement is true for $|Z| = i$. Let X be a subset of R such that $|X| = i + 1$. If $X \in DeltaFree(r, \delta)$ then we can simply choose $Y = X$. If $X \notin DeltaFree(r, \delta)$ then by definition of $DeltaFree(r, \delta)$ there exists a δ-strong rule $Z_1 \Rightarrow Z_2$ based on X. Let A be an item in Z_2 and $Z_3 = X \setminus \{A\}$. As $|Z_3| = |X| - 1$ using the recurrence hypothesis we know that there exists $Y \subseteq Z_3$ such that $Y \in DeltaFree(r, \delta)$ and $Sup(r, Z_3) \geq Sup(r, Y) - \delta|Z_3|$. Since $Z_1 \Rightarrow Z_2$ is a δ-strong rule, then $Sup(r, Z_1) - Sup(r, Z_1 \cup Z_2) \leq \delta$. $Sup(r, Z_1) - Sup(r, Z_1 \cup Z_2)$ is the number of rows not matched by Z_2 but matched by Z_1, thus $Sup(r, Z_1) - Sup(r, Z_1 \cup Z_2)$ is greater or equal to $Sup(r, Z_1 \cup Z_3) - Sup(r, Z_1 \cup Z_2 \cup Z_3)$ (i.e., the number of rows not matched by Z_2 but matched by Z_1 and Z_3). So we have $Sup(r, Z_1 \cup Z_3) - Sup(r, Z_1 \cup Z_2 \cup Z_3) \leq \delta$ which simplifies to $Sup(r, Z_3) - Sup(r, X) \leq \delta$. Since $Sup(r, Z_3) \geq Sup(r, Y) - \delta|Z_3|$ and $|Z_3| = |X| - 1$ we deduce $Sup(r, X) \geq Sup(r, Y) - \delta|X|$. The other inequality $Sup(r, Y) \geq Sup(r, X)$ is straightforward because $Y \subseteq Z_3 \subseteq X$. □

This lemma states that the support of an itemset X can be approximated using the support of one of the δ-free sets, but it does not determine which δ-free set to use. We now show that this can be done by simply choosing among the δ-free sets included in X any δ-free set with a minimal support value. This is stated more formally by the following theorem.

Theorem 4. *Let r be a binary database over a set of items R, $X \subseteq R$ and $\delta \in [0, |r|]$, then for any $Y \subseteq X$ such that $Y \in DeltaFree(r, \delta)$ and $Sup(r, Y) = min(\{Sup(r, Z) : Z \subseteq X$ and $Z \in DeltaFree(r, \delta)\})$ we have $Sup(r, Y) \geq Sup(r, X) \geq Sup(r, Y) - \delta|X|$.*

Proof. Let Y be a subset of X such that $Y \in DeltaFree(r, \delta)$ and satisfying $Sup(r, Y) = min(\{Sup(r, Z) : Z \subseteq X \text{ and } Z \in DeltaFree(r, \delta)\})$. Since $Y \subseteq X$ we have immediately that $Sup(r, Y) \geq Sup(r, X)$. By Lemma 9, there exists $Z \subseteq X$ such that $Z \in DeltaFree(r, \delta)$ and $Sup(r, Z) \geq Sup(r, X) \geq Sup(r, Z) - \delta|X|$. Since Y has the minimal support among all subsets of X in $DeltaFree(r, \delta)$, then $Sup(Z) \geq Sup(Y)$. Thus $Sup(Z) - \delta|X| \geq Sup(Y) - \delta|X|$. As $Sup(r, X) \geq Sup(r, Z) - \delta|X|$, we have $Sup(r, X) \geq Sup(Y) - \delta|X|$. □

In practice, computing the whole collection of δ-free sets is often intractable. We show now that such an exhaustive mining can be avoided since an alternative representation to answer support queries can be obtained if we extract only *frequent* δ-free sets together with a subset of the corresponding negative border.

Definition 26. (frequent δ-free set) Let r be a binary database over a set of items R, we denote $FreqDeltaFree(r, \sigma, \delta) = Freq(r, \sigma) \cap DeltaFree(r, \delta)$ the set of σ-frequent δ-free sets w.r.t. r.

With frequent δ-free sets alone, we cannot directly apply Lemma 9, because we do not know if an itemset X is not in the collection, because it is infrequent or because it is not δ-free. In order to distinquish between these situations, we consider additionally the negative border of frequent δ-free sets.

Definition 27. (negative border of frequent δ-free sets) Let r be a binary database over a set of items R, the negative border of $FreqDeltaFree(r, \sigma, \delta)$ is defined as $\mathcal{B}d^-(FreqDeltaFree(r, \sigma, \delta))$[5].

Informally, the negative border $\mathcal{B}d^-(FreqDeltaFree(r, \sigma, \delta))$ consists of the smallest itemsets (w.r.t. set inclusion) that are not σ-frequent δ-free. Our approximation technique only needs a subset of the negative border $\mathcal{B}d^-(FreqDeltaFree(r, \sigma, \delta))$. This subset, denoted $FreqDeltaFree\mathcal{B}d^-(r, \sigma, \delta)$, is the set of all frequent sets in $\mathcal{B}d^-(FreqDeltaFree(r, \sigma, \delta))$.

Definition 28. $FreqDeltaFree\mathcal{B}d^-(r, \sigma, \delta) = \mathcal{B}d^-(FreqDeltaFree(r, \sigma, \delta)) \cap Freq(r, \sigma)$.

We can now define the SRBC representation.

Definition 29. (strong-rule-bordered condensation) The strong-rule-bordered condensation is the collection of itemsets (and their supports) that are either frequent δ-free sets or in the negative border of frequent δ-free sets. Formally, $SRBC(r, \sigma, \delta) = <\{<X, Sup(r, X)> : X \in FreqDeltaFree(r, \sigma, \delta)\}, \{<X, Sup(r, X)> : X \in FreqDeltaFree\mathcal{B}d^-(r, \sigma, \delta)\}>$[6].

[5] See Definition 4 for $\mathcal{B}d^-$ operator.

[6] Note that this is a pair of collections of itemset-support pairs.

The next theorem states that the collection of itemsets of the strong-rule-bordered condensation is a subcollection of all frequent sets. The proof is trivial, following the definitions of $FreqDeltaFree(r, \sigma, \delta)$ and of $FreqDeltaFree\mathcal{B}d^-(r, \sigma, \delta)$.

Theorem 5. $FreqDeltaFree(r, \sigma, \delta) \cup FreqDeltaFree\mathcal{B}d^-(r, \sigma, \delta) \subseteq Freq(r, \sigma)$.

3.2.3 SRBC as an ϵ-adequate Representation

We can now define the ϵ-adequate representation w.r.t. the support queries based on the SRBC.

Let $\mathcal{DB}_R$ be the set of all possible binary databases over the set of items R. For a given set of items R, let $\mathcal{Q}_R$ be the set of all queries retrieving the relative support of an itemset (subset of R)[7]. If we denote Q_X the query in $\mathcal{Q}_R$ asking for the relative support of the itemset X then $\mathcal{Q}_R = \{Q_X : X \subseteq R\}$ and the value of Q_X on a database instance $r \in \mathcal{DB}_R$ is defined by $Q_X(r) = Sup(r, X)/|r|$.

Definition 30. (representation of binary databases based on SRBC) The representation of $S = \mathcal{DB}_R$ based on $SRBC(r, \sigma, \delta)$ w.r.t. a query class $\mathcal{Q}_R$, is defined by a class of structures C, a representation mapping rep and a query evaluation function m, where:

$$\bigwedge_{r \in \mathcal{DB}_R} rep(r) = SRBC(r, \sigma, \delta),$$

$$C = \{rep(r) : r \in \mathcal{DB}_R\},$$

$$m(Q_X, <\mathcal{F}, \mathcal{B}>) = \begin{cases} S_X/|r| & \text{if } <X, S_X> \in \mathcal{F} \\ min(\{m(Q_{X\setminus\{A\}}, <\mathcal{F}, \mathcal{B}>) : A \in X\}) & \text{if } \bigvee_{<Z, S_Z> \in \mathcal{B}} Z \subseteq X \\ 0 & \text{otherwise} \end{cases}$$

Using this representation, the support of an itemset X is approximated following the reasoning used in the proof of Lemma 9:

- If X is a frequent δ-free set we return its support directly (it is stored in the SRBC).
- Otherwise, X is a superset of an infrequent set or a superset of a non-δ-free set.
- If among subsets of X there is a frequent set in the negative border, we do not know immediately whether X has an infrequent subset, so we recursively compute

[7]Note that $\mathcal{DB}_R$ and $\mathcal{Q}_R$ are the same as defined in Table 2.5, in the example of ϵ-adequate representation.

the supports of its subsets and according to Theorem 4, we choose the one with the least support.

- In case X is not frequent δ-free set and does not have any frequent subset in the negative border, then it has an infrequent subset in the negative border, so X is infrequent and its support is approximated by 0.

We now establish that this representation is an ϵ-adequate representation for the following database class and query class.

Definition 31. (ϵ-adequate representation of binary databases based on SRBC) $\mathcal{DB}_{R,s} = \{r : r \in \mathcal{DB}_R \wedge |r| \geq s\}$, i.e., the set of all binary databases having at least s rows. $\mathcal{Q}_{R,n} = \{Q_X : X \subseteq R \wedge |X| \leq n\}$, i.e., the set of support queries on itemsets having no more than n items.

Theorem 6. *For $r \in \mathcal{DB}_{R,s}$, $SRBC(r, \sigma, \delta)$ is an ϵ-adequate representation w.r.t. $\mathcal{Q}_{R,n}$, where $\epsilon = max(\sigma, n\delta/s)$.*

Proof. Let Q_X be a query in $\mathcal{Q}_{R,n}$ and r a database in $\mathcal{DB}_{R,s}$. Let $c = SRBC(r, \sigma, \delta)$.
If X is a frequent δ-free set then it is in the SRBC and we know its support exactly. Therefore, we do not commit any error here.
If any of the recursive calls of $m(Q_X, c)$ terminates at the last case (returning 0), then X has an infrequent subset and X is not σ-frequent, i.e., $Q_X(r) < \sigma$. Since $m(Q_X, c) = 0$ we have $|Q_X(r) - m(Q_X, rep(r))| \leq \sigma$.
If all recursive calls terminate on a frequent δ-free set then this means that all itemsets included in X are either σ-frequent or not δ-free. Every recursive result retains the minimum value, therefore overall result will be the minimum of all considered support values. Moreover, for any known support, the supports of the subsets cannot be lower (the support is anti-monotone), thus $m(Q_X, rep(r))$ retains the minimum value of all frequent δ-free sets included in X. Whence, $m(Q_X, rep(r)) = min(\{Sup(r, Z) : Z \subseteq X \wedge Z \in DeltaFree(r, \delta)\})$. Thus, by Theorem 4, $m(Q_X, rep(r)) \geq Q_X(r) \geq m(Q_X, rep(r)) - \delta|X|/|r|$. So we have $|Q_X(r) - m(Q_X, rep(r))| \leq n\delta/s$. □

3.2.4 Abstracting SRBC into the Condensed Theories Framework

Now, we are going to instantiate the original framework for condensed theories for the SRBC. Since the framework is foreseen to correct representations, we set δ to 0 in this section.
The SRBC is a pair of collections (see Definition 29), but the patterns of condensed theories are simple collections. In order to be consistent we ought to merge two components of the SRBC.

① We consider a second condensed representation of $\mathcal{T}_{FS}$ (the first was considered for **CLOSE** in Section 3.1.5).
② Same observation as in case of **CLOSE**. But a different class of patterns will be considered as redundant.
An evidence of an exact rule (an association rule with confidence equal to 1) will be used to derive supports of redundant patterns. Note that if the association rule $X \Rightarrow Y$ is an exact rule w.r.t. the binary database, then for any $Z \subseteq R \setminus Y$ such that $Z \supset X$ the association rule $Z \Rightarrow Y$ is also exact. This exact rule permits also to deduce that $Sup(r, Z \cup Y) = Sup(r, Z)$.
Therefore in such a case, the proper supersets of $X \cup Y$ along with their corresponding supports will be considered as redundant patterns.
③ To represent all frequent sets we need all frequent 0-free sets, frequent sets of the negative border of frequent 0-free sets and the corresponding supports. Thus, we store exactly $SRBC(r, \sigma, 0)$.
The resulting condensed representation will consist of a collection of pairs itemset-support, therefore $\mathcal{L}_{\mathcal{C}_{FS2}} = \mathcal{L}_{FS}$.
The $InferentialClosure_{FS2}$ function is defined recursively as:

$$InferentialClosure_{FS2}(\mathcal{C}) = \mathcal{C} \cup \{<X, S_X> \in \mathcal{L}_{FS} : \bigvee_{<Z, S_Z> \in \mathcal{C}} \Big(Z \subset X \wedge \bigvee_{A \in Z} <Z \setminus \{A\}, S_Z> \in \mathcal{C} \wedge <X \setminus \{A\}, S_X> \in InferentialClosure_{FS2}(\mathcal{C})\Big)\}.$$

This recursive function may be described as following:

- all members of the condensed representation $\mathcal{C}$ belong to $InferentialClosure_{FS2}(\mathcal{C})$,
- if X has a proper subset Z (and its support) in $\mathcal{C}$ such that there is a 0-strong rule over Z (i.e., a proper subset of Z has the same support as Z), then by Lemma 7 we know which proper subset of X (member of $InferentialClosure_{FS2}(\mathcal{C})$) has the same support as X, so we use it and include X and its support in $InferentialClosure_{FS2}(\mathcal{C})$.

④ The condensed representation's selection predicate is
$q_{cFS2}(r, <X, S_X>) \equiv S_X = Sup(r, X) \wedge S_X \geq \sigma \wedge \bigwedge_{Y \subset X} \bigwedge_{Z \subset Y} Sup(r, Z) \neq Sup(r, Y)$.
An efficient algorithm mining $\mathcal{C}_{FS2} = Th(\mathcal{L}_{\mathcal{C}_{FS2}}, r, q_{cFS2})$ is given in the following section.
An efficient levelwise recontruction algorithm of $\mathcal{T}_{FS}$ from $\mathcal{C}_{FS2}$ is given in Chapter 4.

3.2.5 Discovering SRBC

In this section, we describe an algorithm, called **Min-Ex**, that generates the SRBC from a binary database. Implementation issues are presented in the following section.

The Algorithm - an Abstract Version

Min-Ex can be seen as an instance of the levelwise search algorithm presented in [Mannila 97b]. It explores the itemset lattice (w.r.t. set inclusion) levelwise, starting from the empty set and stopping at the level of the largest frequent δ-free set.
The *anti-monotonicity* of δ-freeness w.r.t. itemset inclusion (Theorem 3) is used for pruning. Actually, the combined property "frequent and δ-free" is used. It is also anti-monotone as stated by the following lemma.

Lemma 10. *Let r be a binary database over R, and X be an itemset, $X \subseteq R$. For all $Y \subseteq X$ if $X \in FreqDeltaFree(r, \sigma, \delta)$ then $Y \in FreqDeltaFree(r, \sigma, \delta)$.*

Proof. Suppose that $X \in FreqDeltaFree(r, \sigma, \delta)$. Let us consider any $Y \subseteq X$.
Since $X \in FreqDeltaFree(r, \sigma, \delta)$, $X \in Freq(r, \sigma)$ and $X \in DeltaFree(r, \delta)$. According to the anti-monotonicity of support (stated by Corollary of Lemma 1), $Y \in Freq(r, \sigma)$. Similarly, the anti-monotonicity of disjunction-freeness (stated by Theorem 3) implies that $Y \in DeltaFree(r, \delta)$.
Finally, $Y \in Freq(r, \sigma) \cap DeltaFree(r, \delta) = FreqDeltaFree(r, \sigma, \delta)$. □

Algorithm 11 shows the pseudo-code of **Min-Ex**. The collection of candidates is initialized with the singletons (line 1) and then the algorithm iterates on candidate evaluation and larger candidate generation. At each iteration of this loop, it scans the database (line 3) to find out which candidates of size i are frequent. Among them, δ-free sets are stored in line 4 and non δ-free sets in line 5. Then, **Min-Ex** generates candidates for the next iteration, taking every set of size $i + 1$ such that all subsets of size i are frequent δ-free sets (line 6). The algorithm finishes when there is no more candidate (line 2).
Found itemsets are reported in line 9. Note that the empty set is added without test — it is known to be frequent, and it is δ-free, because there cannot be any association rule based on $\emptyset$, in particular any valid δ-strong rule.
Using the correctness result of the levelwise search algorithm given in [Mannila 97b] the following theorem is straightforward.

Theorem 7 (Correctness of Min-Ex). *Given r a binary database over a set of items R, and σ, δ two absolute thresholds, the algorithm* **Min-Ex** *computes all frequent δ-free sets and all frequent sets of the negative border of frequent δ-free sets.*

Implementation Issues

We used techniques similar to the ones described in Section 2.2 for **APRIORI**. The candidate generation is made using a join-based function, and the itemset support counters are updated w.r.t. a row of the database using a prefix-tree data structure.
The key point that needs a new specific technique is the δ-freeness test in lines 4 and 5 of the algorithm. An efficient computation of this test can be done, based on the following

remark: Z is not a δ-free set if and only if there exist $A \in Z$ and $X = Z \setminus \{A\}$ such that X is not δ-free or X is δ-free and $X \Rightarrow \{A\}$ is a δ-strong rule. Furthermore, the step 6 of the algorithm guarantees that if Z is a candidate then X must be δ-free since X is a subset of Z of size $|Z| - 1$. Therefore, we might compare the support value of Z with the support values of its subsets of size $|Z| - 1$ and if they do not all differ from the support value of Z by more that δ, Z is not δ-free.
The test described above is quite efficient (we use only already counted support values) and may easily be incorporated in the algorithm.

3.2.6 Experiments

The running prototype is implemented in C++. We use a PC with 1 GB of memory and a 1900 MHz Pentium IV processor under Linux operating system.
For an experimental evaluation, we chose the **PUMSB*** data set, a PUMS census data set[8] preprocessed by researchers from IBM Almaden Research Center. The particularity of PUMS data sets is that they are very dense and make the mining of all frequent sets together with their supports intractable for low support thresholds, because of the combinatorial explosion of the number of frequent sets [Bayardo, Jr. 98].

SRBC vs. Frequent Sets

We compared the size of $Freq(r, \sigma)$, $FreqClosed(r, \sigma)$ and $SRBC(r, \sigma, \delta)$ for different support thresholds and different values of δ.

[8] http://www.almaden.ibm.com/cs/quest/data/long_patterns.bin.tar

Algorithm 11. (Min-Ex)

Input: *r a binary database over a set of items R, $\sigma \in [0, |r|]$ and $\delta \in [0, |r|]$ two thresholds.*
Output: $SRBC(r, \sigma, \delta)$.

1: **let** $\mathcal{CG}_1 := \{\{A\} : A \in R\}$, $i := 1$;
2: **while** $\mathcal{CG}_i \neq \emptyset$ **do**
3: *Scan r and compute supports of itemsets in $\mathcal{CG}_i$;*
4: **let** $\mathcal{FG}_i := \{X : X \in \mathcal{CG}_i \wedge X$ *is a σ-frequent δ-free set in r*$\}$;
5: **let** $\mathcal{FB}_i := \{X : X \in \mathcal{CG}_i \wedge X$ *is a σ-frequent and not δ-free set in r*$\}$;
6: **let** $\mathcal{CG}_{i+1} := \{X : X \subseteq R \wedge |X| = i + 1 \wedge \bigwedge_{Y \subset X} |Y| = i \Rightarrow Y \in \mathcal{FG}_i\}$;
7: **let** $i := i + 1$;
8: **od**
9: **output** $< \bigcup_{j<i}\{<X, support\ of\ X> : X \in \mathcal{FG}_j\} \cup \{\emptyset, |r|\}$,
$\bigcup_{j<i}\{<X, support\ of\ X> : X \in \mathcal{FB}_j\} >$;

The three representations (frequent sets, frequent closed sets and the SRBC) are composed of a collection of itemsets and their associated supports. We retain two measures that reflect the size of these representations

- total number of itemsets in the representation, and
- flat-storage space used.

The first measure is interesting in itself, but it does not correspond directly to the amount of data to be stored. The benefit of the second measure is to characterize this aspect more precisely.

We consider the storage in a binary file without any ad-hoc encoding or compression, and we make the following assumptions. The information relative to an itemset can be stored using one integer for its support and one integer per item, all used integers are ranging from 0 to 49 046 (for the data set used in experiments). We neglect the additional marker of the end of the sequence of items. And finally, we consider that the elementary storage unit is simply one 32-bit long integer in a fixed format. So, let $\mathcal{S}$ be the collection of itemsets in one of the three representations, the corresponding flat storage space (in bytes) is computed as $4 * \sum_{X \in \mathcal{S}}(|X| + 1)$.

Figure 3.1 emphasizes, using logarithmically scaled axes, the observed difference of sizes of the various representations. Lower values correspond to more condensed representations. It should be noticed that the support thresholds are given as relative support thresholds (i.e., 100 $\times$ absolute support threshold / total number of rows in the data set).

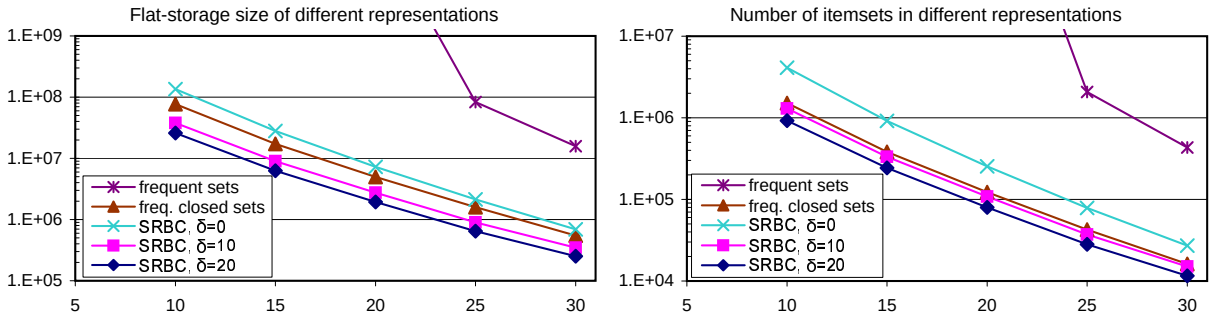

Figure 3.1: Sizes of different representations.

We observed, as expected, that increasing δ leads to higher condensations. Between $\delta = 0$ and $\delta = 20$ we observed around 4-fold reduction in size, for both used measures.

An interesting comparison is between the SRBC and frequent closed sets. The representation based on frequent closed sets is comparable in size with the SRBC for $\delta = 10$ using the number of itemsets as measure (the curves nearly overlap), or comparable with an SRBC with δ between 0 and 10 for the flat-storage measure.

For both measures, the condensation of the SRBC for $\delta = 20$ is higher than of frequent closed sets.

The comparison with frequent sets was less challenging — according to Theorem 5 we *a priori* know that the SRBC is always smaller or equal in size.

Expectedly, we observe higher condensation of the SRBC than of frequent sets. The difference is very important and ranges from one to more-than-five orders of magnitude. Not represented on the figures were the sizes of the collections of frequent sets for relative support thresholds of 10%, 15% and 20%. For these three support thresholds, we use lower-bound estimations of both measures for frequent sets. These lower-bounds are computed using $SRBC(r, \sigma, 0)$ extracted by **Min-Ex** to find the exact size of the largest frequent set. If this size is m then there are a least 2^m frequent sets, and $\frac{m}{2}2^m$ integers in the flat-storage. The observed that largest frequent sets have resp. 32 (20%), 35 (15%) and 39 items (10%).

Notice that given these values, $Freq(r, \sigma)$ is so large that it is clearly impossible to provide it on our platform, while the extraction of $SRBC(r, \sigma, \delta)$ remains tractable.

We observe a brutal change between the size of $Freq(r, 0.25 * |r|)$ and of $Freq(r, 0.20 * |r|)$: 1000 times more frequent sets than expected by extrapolating the trend given by $Freq(r, 0.25 * |r|)$ and $Freq(r, 0.30 * |r|)$. If we look at the trend of the number of frequent δ-free sets it seems to be unchanged. The reason for this, is that between 0.25 and 0.20 we reach a relative support threshold where the number of strong rules increases significantly and then leads to the explosion of the number of frequent sets, but not to the explosion of the number of frequent δ-free sets.

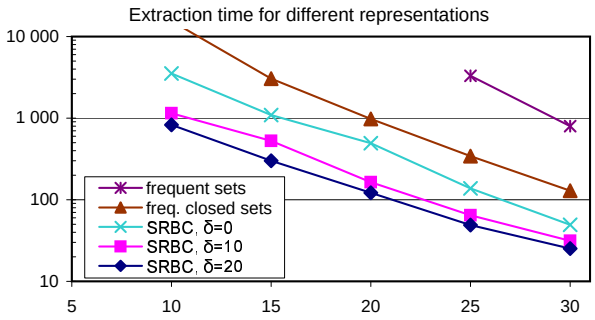

Figure 3.2: Extraction times of different representations.

Using also logarithmically scaled axes, Figure 3.2 shows that the extraction time for **Min-Ex** grows up exponentially when the support threshold is reduced. This is due to the combinatorial explosion of the number of frequent δ-free sets. **APRIORI**-based algorithms have a similar exponential evolution of the extraction time, due in this case to the combinatorial explosion of the number of frequent sets.

We observed that the extraction of frequent δ-free sets outperforms in all cases the extraction of both, frequent sets and frequent closed sets.

Scale-up Experiments

On Figure 3.3, we report the extraction time when changing the number of rows or the number of items in the data set. The same data set as in the previous experiments is used, but a fixed relative support threshold of 20% is used.

We observe an exponential complexity w.r.t. the number of items (Figure 3.3, right) and a linear complexity w.r.t. number of rows in the data set if the value of δ follows the number of tuples (e.g., if we double the number of rows then we double the value of δ). This is emphasized by a superimposed straight line on Figure 3.3 (left).

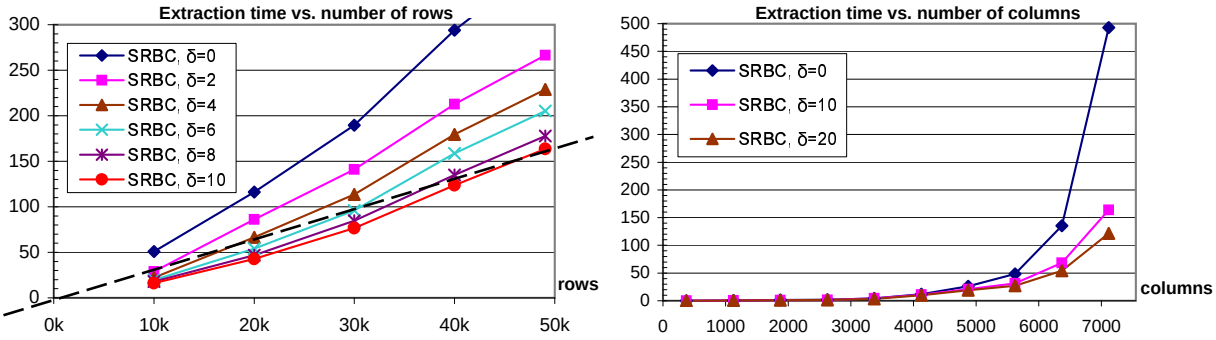

Figure 3.3: Behavior of **Min-Ex** w.r.t. the number of rows and the number of items of the binary database.

3.2.7 Summary of Results Related to SRBC

In this section, we have shown how δ-strong rules that are valid in a binary database can be put into use to define condensed representations (an ϵ-adequate representation of the binary database and a condensed theory of frequent sets). We also presented the algorithm **Min-Ex** computing the SRBC in a binary database, the component of the condensed representations.

We showed through experiments that on highly correlated data sets the SRBC is computed very efficiently, when compared to **APRIORI** (computing all frequent sets) and **CLOSE** (computing frequent closed sets).

The representation based on frequent closed sets (implicitly proposed in [Pasquier 99c] and formalized in Section 3.1.5) is strongly related to the notion of 0-free sets (δ-free sets with $\delta = 0$). In [Bastide 00b], an equivalent notion to frequent 0-free set, called *frequent key pattern*, has been independently proposed with the goal of mining faster frequent sets. Notice however that they have not been developped as a condensed representation. Mining 0-free sets or closed sets lead to similar gains, but mining δ-free sets with $\delta \neq 0$ offers additional search space reductions (at the cost of an uncertainty on supports). It should also be noticed that by definition exact rules are very sensitive to noise. If we process a noisy data set (a very common case in practice) a few exceptions to the exact rules can appear easily. Then the pruning methods based on exact rules will be less effective, while the mining of δ-free sets with $\delta \neq 0$ can still benefit of an important search space reduction.

The techniques mentioned in this section present important benefits on dense data sets, but if we consider very sparse data sets, we can hardly expect to have many exact or nearly exact rules that hold, and thus all these techniques are likely to be with little or

without benefits. Moreover, on very sparse data sets, these techniques may be a little bit slower than the extraction of frequent sets without additional pruning, since they cannot take advantage of important search space reductions, but have to pay for a small overhead due to the (unfruitful) tests performed to detect the δ-strong rules.

3.3 Original Condensed Representation Based on Frequent Disjunction-Free Sets

In this section, we describe frequent disjunction-free sets and define a condensed representation based on them. This second original representation, called *disjunction-bordered condensation*, contains the same amount of information as the full collection of frequent sets.

The preliminary study about this representation has been presented in [Bykowski 01] and a mature research paper can be found in [Bykowski 03].

Like, strong-rule-bordered condensation, disjunction-bordered condenstation (DBC for short) consists of a subcollection of the frequent sets and can be extracted more efficiently than frequent sets. This collection is lossless. Therefore, we abstract it only into the framework for exact represenations proposed in Section 3.1.

The DBC can be used to greatly reduce the running time and the storage space requirement of the data mining processes involving frequent sets and frequent closed sets.

3.3.1 Informal Introduction

The condensation of the collection of frequent sets by the DBC is due to a property that binds supports of some itemsets by equations. This property is based on expressions called simple disjunctive rules. The general form of such rules is $A_1 \wedge A_2 \wedge \ldots \wedge A_{n-2} \Rightarrow A_{n-1} \vee A_n$, where A_i represent different items (with an exception on the last 2 items, i.e., A_{n-1} and A_n, which are allowed to be the same). This rule states that if $A_1, \ldots, A_{n-2}$ are set to *true* within a row, then A_{n-1} or A_n is set to *true* within the same row. The rule may hold in a row or not. The latter case arises when A_{n-1} and A_n are set to $false$ in spite of all $A_1, A_2, \ldots, A_{n-2}$ being set to *true*.

Consider the table r' depicted in Figure 3.4. The rule $B \wedge C \Rightarrow A \vee D$ holds for example in the second and third row, but not in the first one. The size gain of the DBC is based on rules that hold in all rows of a table, as for instance $A \Rightarrow C \vee D$ in r'.

Observe the rows with A *true* in Figure 3.4. Since $A \Rightarrow C \vee D$ holds in all rows, there is no row with A *true* and both C and D $false$. Thus, the support of A is simply equal to the sum of the supports of $\{A, C\}$ and $\{A, D\}$ minus the support of $\{A, C, D\}$ (because the support of $\{A, C, D\}$ has been counted in the support of both $\{A, C\}$ and $\{A, D\}$). So if we know the supports of $\{A\}$, $\{A, C\}$ and $\{A, D\}$, then we can avoid the extraction

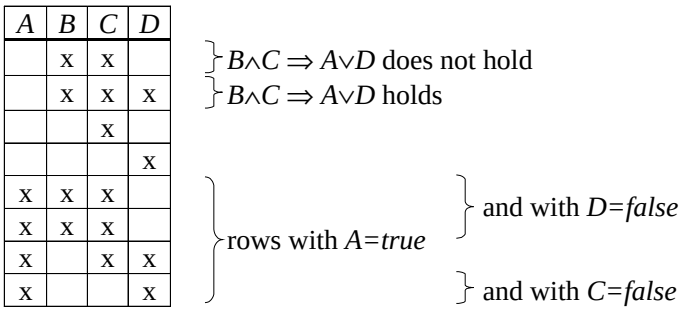

Figure 3.4: Illustration of simple disjunctive rules.

and the support counting of $\{A, C, D\}$ by simply using the equation $Sup(r', \{A, C, D\}) = Sup(r', \{A, C\}) + Sup(r', \{A, D\}) - Sup(r', \{A\})$.
Moreover, the figure shows that we also have $Sup(r', \{A, B, C, D\}) = Sup(r', \{A, B, C\}) + Sup(r', \{A, B, D\}) - Sup(r', \{A, B\})$ as a consequence of the previous equation. If there is more attributes than A, B, C and D, then similar relations will hold for all supersets of $\{A, C, D\}$. Therefore, we may skip the support counting of all proper supersets of $\{A, C, D\}$, while still being able to compute their supports using these relations between the itemset supports.
An itemset containing items that can be used to form a simple disjunctive rule holding in all rows is called a *non-disjunction-free set*. The remaining itemsets are called *disjunction-free sets*. In the previous example $\{A, C, D\}$ and $\{A, B, C, D\}$ are non-disjunction-free, because of the rule $A \Rightarrow C \vee D$.
The DBC is simply the collection of all frequent disjunction-free sets and of all frequent and minimal (w.r.t. set inclusion) non-disjunction-free sets, along with the corresponding support information.
The most interesting property of the DBC is that this collection of itemsets can be extracted very efficiently and is sufficient to compute all frequent sets and their supports.

3.3.2 Formal Definitions

The notion of disjunction-free set is based on *simple disjunctive rules*, so we define them first.

Definition 32. (simple disjunctive rule) Let X be a set of (positive) items, a *simple disjunctive rule* based on X is an expression of the form $Y \Rightarrow A \vee B$, where $Y \subset X$ and $A, B \in X \setminus Y$. Notice that A and B are single items and also that a rule of the form $Y \Rightarrow A \vee A$ is a particular case of simple disjunctive rule. Let r be a binary database over R, where $X \subseteq R$. The simple disjunctive rule $Y \Rightarrow A \vee B$ is *valid* in r if and only if $\mathcal{M}(r, Y) = \{t \in r : t \in \mathcal{M}(r, Y \cup \{A\}) \vee t \in \mathcal{M}(r, Y \cup \{B\})\}$.

Now, we state three lemmas. They are fundamentals needed for the further development of the representation.

Lemma 11. *Let r be a binary database over R, and Y be a set of items such that $Y \subseteq R$. Let A and B be single items in R, and $\overline{A}, \overline{B}$ be their corresponding negative items. Then the following three propositions are equivalent:*
(1) $Y \Rightarrow A \vee B$ is a valid simple disjunctive rule in r,
(2) $\mathcal{M}(r, Y \cup \{\overline{A}, \overline{B}\}) = \emptyset$,
(3) $Sup(r, Y \cup \{\overline{A}, \overline{B}\}) = 0$.

Proof. From Definition 12 *(2)* is trivially equivalent to *(3)*. Now, we show that *(3)* implies *(1)* and later that *(1)* implies *(2)*.

Suppose $Sup(r, Y \cup \{\overline{A}, \overline{B}\}) = 0$. By Definition 12, there is no $t \in r$ such that $Y \subseteq t$ and $\{A, B\} \cap t = \emptyset$. Thus, $\bigwedge_{t \in r} Y \subseteq t \Rightarrow A \in t \vee B \in t$. So, $\mathcal{M}(r, Y) = \{t \in r : t \in \mathcal{M}(r, Y \cup \{A\}) \vee t \in \mathcal{M}(r, Y \cup \{B\})\}$ and finally, by Definition 32, $Y \Rightarrow A \vee B$ is valid.

Suppose now that $Y \Rightarrow A \vee B$ is valid. By Definition 32, $\mathcal{M}(r, Y) = \{t \in r : t \in \mathcal{M}(r, Y \cup \{A\}) \vee t \in \mathcal{M}(r, Y \cup \{B\})\}$. So, there is no row $t \in r$ such that $Y \subseteq t$ and $A \notin t$ and $B \notin t$. Thus, the multiset $\{t \in r : Y \subseteq t \wedge \{A, B\} \cap t = \emptyset\} = \{t \in r : Pos(Y \cup \{\overline{A}, \overline{B}\}) \subseteq t \wedge Neg(Y \cup \{\overline{A}, \overline{B}\}) \cap t = \emptyset\}$ is empty. And then, by Definition 12, $\mathcal{M}(r, Y \cup \{\overline{A}, \overline{B}\}) = \emptyset$. □

Lemma 12. *Let r be a binary database over R, and X, Y be itemsets such that $Y \subseteq X \subseteq R$. Let $A, B \in Y$. If $Y \setminus \{A, B\} \Rightarrow A \vee B$ is valid in r then $X \setminus \{A, B\} \Rightarrow A \vee B$ is also valid in r.*

Proof. Suppose that $Y \setminus \{A, B\} \Rightarrow A \vee B$ is valid in r. By Lemma 11, $\mathcal{M}(r, (Y \setminus \{A, B\}) \cup \{\overline{A}, \overline{B}\}) = \emptyset$. Then, since X is a superset of Y from Definition 12 and Lemma 2 we deduce that $\mathcal{M}(r, (X \setminus \{A, B\}) \cup \{\overline{A}, \overline{B}\}) = \emptyset$. Whence, using again Lemma 11, we know that $X \setminus \{A, B\} \Rightarrow A \vee B$ is valid. □

Lemma 13. *Let r be a binary database over R, $X \subseteq R$ be an itemset, and A, B be items in X. Then there exists $Y \subset X$ such that $Y \Rightarrow A \vee B$ is a valid simple disjunctive rule based on X, if and only if $Sup(r, X) = Sup(r, X \setminus \{A\}) + Sup(r, X \setminus \{B\}) - Sup(r, X \setminus \{A, B\})$.*

Proof. Consider $Y \Rightarrow A \vee B$ the valid simple disjunctive rule based on X and $Z = X \setminus \{A, B\}$. By Definition 32, $Y \subseteq Z$ and then from Lemma 12, we deduce that $Z \Rightarrow A \vee B$ is valid.
We can partition the multiset of rows matched by Z into three multisets $\mathcal{M}(r, Z \cup \{A\})$, $\mathcal{M}(r, Z \cup \{\overline{A}, \overline{B}\})$ and $\mathcal{M}(r, Z \cup \{\overline{A}, B\})$. As $Z \Rightarrow A \vee B$ is valid, using Lemma 11 we have $\mathcal{M}(r, Z \cup \{\overline{A}, \overline{B}\}) = \emptyset$. Thus, $Sup(r, Z) = Sup(r, Z \cup \{A\}) + Sup(r, Z \cup \{\overline{A}, B\})$. Because $\mathcal{M}(r, Z \cup \{\overline{A}, B\}) = \{t \in r : t \in \mathcal{M}(r, Z \cup \{B\}) \wedge t \notin \mathcal{M}(r, Z \cup \{A, B\})\}$ and $\mathcal{M}(r, Z \cup \{B\}) \supseteq \mathcal{M}(r, Z \cup \{A, B\})$, we have $Sup(r, Z \cup \{\overline{A}, B\}) = Sup(r, Z \cup \{B\}) - Sup(r, Z \cup \{A, B\})$, and finally $Sup(r, X \setminus \{A, B\}) = Sup(r, X \setminus \{B\}) + Sup(r, X \setminus \{A\}) - Sup(r, X)$.

Suppose now that $Sup(r, X) = Sup(r, X \setminus \{A\}) + Sup(r, X \setminus \{B\}) - Sup(r, X \setminus \{A, B\})$. Let $Z = X \setminus \{A, B\}$. $Sup(r, Z \cup \{\overline{A}, \overline{B}\}) = Sup(r, Z \cup \{\overline{A}\}) - Sup(r, Z \cup \{\overline{A}, B\}) = (Sup(r, Z) - Sup(r, Z \cup \{A\})) - (Sup(r, Z \cup \{B\}) - Sup(r, Z \cup \{A, B\})) = Sup(r, X \setminus \{A, B\}) - Sup(r, X \setminus \{B\}) - Sup(r, X \setminus \{A\}) + Sup(r, X) = 0$. Therefore, from the Lemma 11, we deduce that $Z \Rightarrow A \vee B$ is valid, which proves the existence of the required rule. □

Now, we define the notion of disjunction-free sets.

Definition 33. (disjunction-free set) Let r be a binary database over R, $X \subseteq R$ is a *disjunction-free set* w.r.t. r if and only if no simple disjunctive rule based on X is valid in r. The set of all disjunction-free sets w.r.t. r is noted $DisjFree(r)$.

The anti-monotonicity of disjunction-freeness follows directly from the definition of disjunction-free sets and is stated by the following lemma.

Lemma 14. *Let r be a binary database over R, and X be an itemset, $X \subseteq R$. For all $Y \subseteq X$ if $X \in DisjFree(r)$ then $Y \in DisjFree(r)$.*

Definition 34. (frequent disjunction-free set) Let r be a binary database over a set of items R, $FreqDisjFree(r, \sigma) = Freq(r, \sigma) \cap DisjFree(r)$ denotes the set of all σ-frequent disjunction-free sets w.r.t. r.

Once more, we instantiate the concept of negative border and define it for σ-frequent disjunction-free sets.

Definition 35. (negative border of frequent disjunction-free sets) Let r be a binary database over a set of items R, the negative border of $FreqDisjFree(r, \sigma)$ is defined as $\mathcal{B}d^-(FreqDisjFree(r, \sigma))$.

Out of all itemsets of the negative border, we need only the frequent ones. The definition follows.

Definition 36. (frequent sets of negative border) Let r be a binary database over a set of items R, the collection of all frequent sets of negative border of $FreqDisjFree(r, \sigma)$ is defined as $FreqDisjFree\mathcal{B}d^-(r, \sigma) = \mathcal{B}d^-(FreqDisjFree\mathcal{B}d^-(r, \sigma)) \cap Freq(r, \sigma)$.

Definition 37. (disjunction-bordered condensation) The disjunction-bordered condensation is the collection of frequent sets that are either disjunction-free sets or in the negative border of frequent disjunction-free sets. Formally, $DBC(r, \sigma) = <\{<X, Sup(r, X)> : X \in FreqDisjFree(r, \sigma, \delta)\}, \{<X, Sup(r, X)> : X \in FreqDisjFree\mathcal{B}d^-(r, \sigma, \delta)\}>$.

3.3.3 Properties of Disjunction-Bordered Condensation

The following theorem supports an earlier claim, which stated that the disjunction-bordered condensation is a subcollection of the collection of all frequent sets. The proof is trivial, following the definitions of $FreqDisjFree(r,\sigma)$ and of $FreqDisjFree\mathcal{B}d^{-}(r,\sigma)$.

Theorem 8. $FreqDisjFree(r,\sigma) \cup FreqDisjFree\mathcal{B}d^{-}(r,\sigma) \subseteq Freq(r,\sigma)$.

We can now state the *correctness* of the DBC, i.e., the DBC is sufficient to determine the support of any frequent set.

Theorem 9. *Let r be a binary database over a set of items R, X be an itemset such that $X \subseteq R$, and σ be an absolute support threshold. Using the itemsets in $FreqDisjFree(r,\sigma)$ and in $FreqDisjFree\mathcal{B}d^{-}(r,\sigma)$, together with their supports, we can determine if X is σ-frequent, and when X is σ-frequent we can also determine $Sup(r,X)$.*

Proof. Let X be any itemset. If X is in $DBC(r,\sigma)$ (in $FreqDisjFree(r,\sigma)$ or in $FreqDisjFree\mathcal{B}d^{-}(r,\sigma)$), we know its support, so we can state that X is σ-frequent, and we can also give $Sup(r,X)$. This situation will be referred to as *trivial case* in this proof. The proof is made by induction on $|X|$.

First, let us consider the case where $X = \emptyset$. Given that $\sigma \in (0,|r|]$ and that $Sup(r,\emptyset) = |r|$, $X \in Freq(r,\sigma)$. Moreover, there cannot be simple disjunctive rule based on $\emptyset$ (see Definition 32), so X must be frequent disjunction-free set and we have the *trivial case.*

In the following, we suppose that $X \neq \emptyset$.

Hypothesis. Suppose that for every itemset $W \subset X$, we can determine if W is σ-frequent, and when W is σ-frequent we can also determine $Sup(r,W)$.

If $X \in FreqDisjFree(r,\sigma)$ then we have again the *trivial case.*

If $X \notin FreqDisjFree(r,\sigma)$ and $\bigwedge_{Y \subseteq X} Y \notin FreqDisjFree\mathcal{B}d^{-}(r,\sigma)$ then we show that X is not σ-frequent. First, observe that if $X \notin FreqDisjFree(r,\sigma)$ then, by Definition 4, $\bigvee_{W \subseteq X} W \in \mathcal{B}d^{-}(FreqDisjFree(r,\sigma))$. Let W be such a set. W is not σ-frequent since $\bigwedge_{Y \subseteq X} Y \notin FreqDisjFree\mathcal{B}d^{-}(r,\sigma)$. By the anti-monotonicity of support, we know that X is not σ-frequent, because $W \subseteq X$.

If $X \notin FreqDisjFree(r,\sigma)$ and $X \in FreqDisjFree\mathcal{B}d^{-}(r,\sigma)$ then, we have once more the *trivial case.*

We consider now the case where $X \notin FreqDisjFree(r,\sigma)$ and $\bigvee_{Y \subset X} Y \in FreqDisjFree\mathcal{B}d^{-}(r,\sigma)$. Let Y be such an itemset.

$Y \in FreqDisjFree\mathcal{B}d^{-}(r,\sigma)$ implies that Y is not a disjunction-free set. In this case, we now construct a valid simple disjunctive rule based on Y, and then we use this rule to decide if X is σ-frequent and to compute its support. By Definition 33, Y is not a disjunction-free set implies that there exists $Z \subset Y$ and $A, B \in Y \setminus Z$ such that $Z \Rightarrow A \vee B$

is a valid simple disjunctive rule in r. By Lemma 13, we have $Sup(r, Z \cup \{A, B\}) = Sup(r, Z \cup \{B\}) + Sup(r, Z \cup \{A\}) - Sup(r, Z)$.
Since Y is σ-frequent, by Lemma 1 all its subsets are σ-frequent. Because $|Y| < |X|$ and Y is σ-frequent, by the induction hypothesis we can determine the supports of all subsets of Y. So we can find among all subsets of Y four itemsets $Z, Z \cup \{A, B\}, Z \cup \{B\}, Z \cup \{A\}$ satisfying the relation $Sup(r, Z \cup \{A, B\}) = Sup(r, Z \cup \{B\}) + Sup(r, Z \cup \{A\}) - Sup(r, Z)$, which corresponds to a valid rule $Z \Rightarrow A \vee B$.
By the induction hypothesis, we can determine for the itemsets $X \setminus \{A, B\}, X \setminus \{A\}$ and $X \setminus \{B\}$ (the items A and B are the same as above) if they are all σ-frequent and if so we can also determine their supports. If these three sets are σ-frequent, using their supports we can determine the support of X as $Sup(r, X) = Sup(r, X \setminus \{A\}) + Sup(r, X \setminus \{B\}) - Sup(r, X \setminus \{A, B\})$ (according to Lemma 13), because $Z \Rightarrow A \vee B$ is a valid disjunctive rule based on X. If at least one of these itemsets is not σ-frequent so neither is X (by Lemma 1), which completes the proof. □

It should be noticed that the proof of Theorem 9 is constructive and that it can be used as a naïve recursive algorithm to determine $Sup(r, X)$.

3.3.4 Abstracting DBC into the Condensed Theories Framework

In this section, we instantiate the original framework for condensed theories on the DBC. As in case of the SRBC, the DBC is a pair of collections, but the condensed theories framework allows a single pattern collection. Therefore, we will refer to the DBC as a single collection containing both frequent disjunction-free sets and frequent sets of the negative bodrer.

① We consider a third condensed representation of $\mathcal{T}_{FS}$.
② As redundant, we consider a frequent set X (and its corresponding support) such that a disjunctive rule based on any proper subset of X is valid in r. If a valid disjunctive rule can be based on X, but not on its proper subsets, we do not consider X as redundant (it will be necessary to infer other patterns).
③ Therefore, the itemsets from $FreqDisjFree(r, \sigma) \cup FreqDisjFree\mathcal{B}d^{-}(r, \sigma)$ (and their corresponding supports) are non-redundant. The condensed representation will consist of a collection of pairs itemset-support ($\mathcal{L}_{\mathcal{C}FS3} = \mathcal{L}_{FS}$).
The $InferentialClosure_{FS3}$ function is recursively defined as[9]:

$$InferentialClosure_{FS3}(\mathcal{C}) = \mathcal{C} \cup \{<X, S_X> \in \mathcal{L}_{FS} : \bigvee_{<Z, S_Z> \in \mathcal{C}} \Big(Z \subset X \wedge \bigvee_{\substack{A, B \in Z \\ S_{ZA}, S_{ZB}, S_{ZAB} \in \mathbb{N} \\ S_{XA}, S_{XB}, S_{XAB} \in \mathbb{N}}} <Z \setminus \{A\}, S_{ZA}> \in \mathcal{C} \wedge <Z \setminus \{B\}, S_{ZB}> \in \mathcal{C} \wedge <Z \setminus \{A, B\}, S_{ZAB}> \in \mathcal{C} \wedge S_Z + S_{ZAB} = S_{ZA} + S_{ZB} \wedge <X \setminus \{A\}, S_{XA}> \in InferentialClosure_{FS3}(\mathcal{C}) \wedge <X \setminus \{B\}, S_{XB}> \in InferentialClosure_{FS3}(\mathcal{C}) \wedge <X \setminus \{A, B\}, S_{XAB}> \in InferentialClosure_{FS3}(\mathcal{C}) \wedge S_X = S_{XA} + S_{XB} - S_{XAB} \wedge S_X \geq \sigma \} \Big) \}.$$

[9]Note that A appearing in the formula can be the same item as B.

This recursive function may be described as following:

- all members of the condensed representation $\mathcal{C}$ belong to $InferentialClosure_{FS3}(\mathcal{C})$,
- if X has a proper subset Z (and its support) in $\mathcal{C}$ such that there is a valid disjunctive rule based on Z (as asserted by the relationship on supports of the subsets Z, $Z \setminus \{A\}$, $Z \setminus \{B\}$, $Z \setminus \{A, B\}$), then by Lemma 12 we know from which subsets of X (members of $InferentialClosure_{FS3}(\mathcal{C})$) the support of X can be inferred, so we use them and include X and its support in $InferentialClosure_{FS3}(\mathcal{C})$ if the inferred support of X satisfies the support criterion.

④ The condensed representation's selection predicate is
$q_{cFS3}(r, <X, S_X>) \equiv S_X = Sup(r, X) \wedge S_X \geq \sigma \wedge \bigwedge_{\substack{Y \subset X \\ A, B \in Y}} Sup(r, Y) + Sup(r, Y \setminus \{A, B\}) \neq Sup(r, Y \setminus \{A\}) + Sup(r, Y \setminus \{B\})$.
The condensed representation $\mathcal{C}_{FS3} = Th(\mathcal{L}_{\mathcal{C}FS3}, r, q_{cFS3})$ of $\mathcal{T}_{FS}$ is the disjunction-bordered condensation.
An efficient algorithm mining the representation is given in Section 3.3.6. An efficient levelwise recontruction algorithm of $\mathcal{T}_{FS}$ from $\mathcal{C}_{FS3}$ is given in Chapter 4. Additionally in Chapter 4, efficient representation change algorithms are also given, changing $\mathcal{C}_{FS3}$ either into $\mathcal{C}_{FS1}$ (defined in Section 3.1.5) or into $\mathcal{C}_{FS2}$ (defined in Section 3.2.4).

3.3.5 Comparing the Representation Size

In this section, we report a comparison between the sizes of the different representations. For different data sets and support thresholds, we consider the collection of all frequent sets, the collection of all frequent closed set[10] and the DBC.
We measured the size of the three representations (frequent sets, frequent closed sets and the DBC) in the same way as in the experiments with the SRBC (in Section 3.2.6).
We report representation sizes on three different commonly used data sets: **Mushroom** (characteristics of some mushroom species), **Connect-4** (collection of game-related state information), and **PUMSB** (PUMS census data). All these data sets have been preprocessed by researchers from IBM Almaden Research Center[11]. The particularity of the selected data sets is that they are very dense and the combinatorial explosion of the number of frequent sets makes the mining of all frequent sets together with their supports intractable for low support thresholds [Bayardo, Jr. 98].
The representation sizes for several support thresholds are given in Figure 3.5 (note that some axes are logarithmically scaled). Lower values correspond to more condensed representations. It should be noticed that the support thresholds are given as relative support thresholds (i.e., 100 × absolute support threshold / total number of rows in the data set).

[10] As defined in Section 3.1.5, frequent closed sets allow regenerating all frequent sets with their supports.
[11] http://www.almaden.ibm.com/cs/quest/data/long_patterns.bin.tar

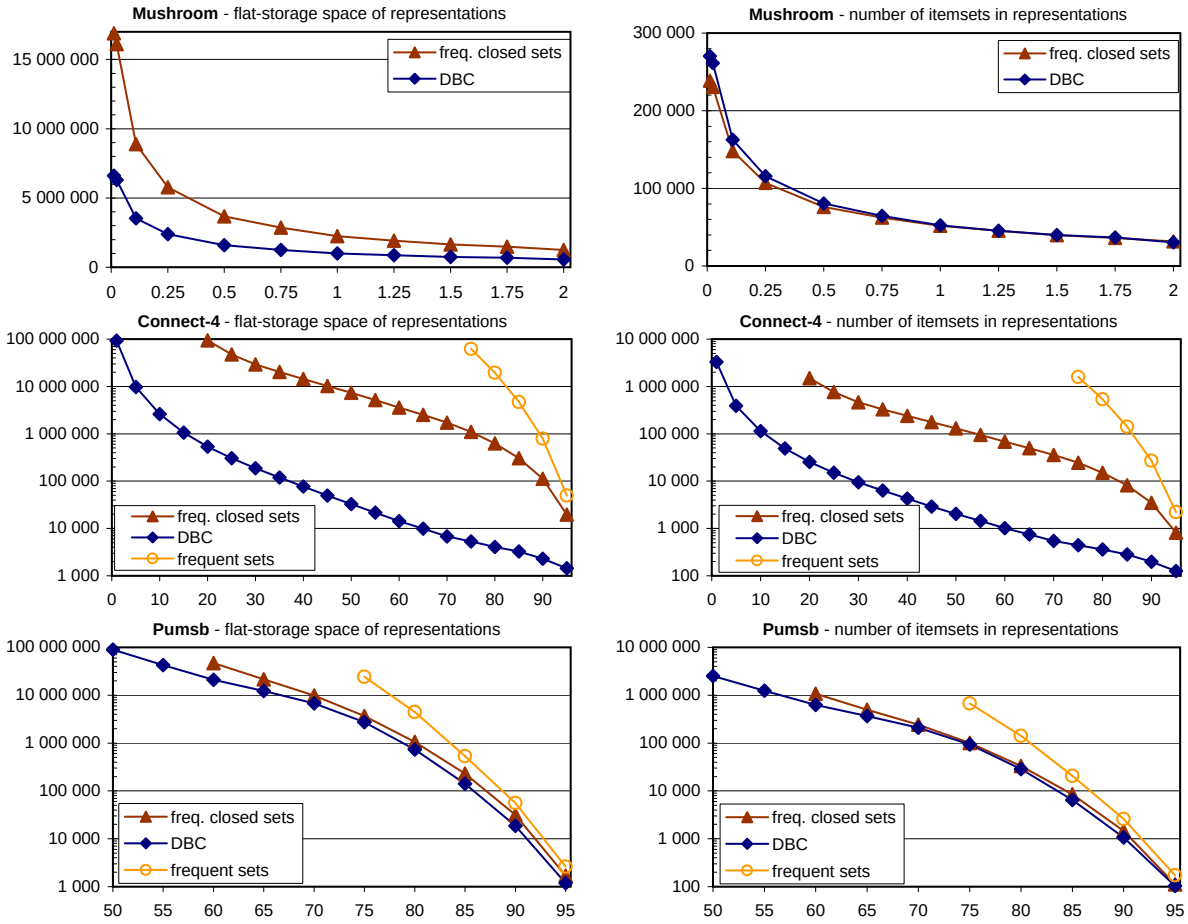

Figure 3.5: Different representation sizes on the three data sets.

The **Mushroom** data set is based on 119 items only and contains 8124 rows — a relatively small number in data mining. Nonetheless, at the thresholds used in the experiments (Figure 3.5, upper graphics) the collection of all frequent sets was huge (more than 22 millions for a relative support threshold of 2%) and we could not extract frequent sets completely for thresholds lower than 2%, even with a non-naïve implementation of the **APRIORI** algorithm.

The **Connect-4** data set contains 67 557 rows, but a relatively small number of items (129). Its difficulty lies in a very high correlation. A huge number of frequent sets was observed even for as high support thresholds as 75% (Figure 3.5, middle graphics).

The last data set, **Pumsb**, contains 49 046 rows and is very challenging because of the high number of items (7117). For this reason, when decreasing the support threshold, the sizes quickly grow for all representations (see Figure 3.5, lower graphics).

On Figure 3.5 in the left graphics, we can see that the DBC representation is smaller than the frequent closed sets representation in all experiments when we consider their flat-storage spaces. The same holds with few exceptions if we count the number of itemsets required by each representation (Figure 3.5, right graphics).

For the comparison of the size of the DBC with the size of the collection of all frequent sets, according to Theorem 8 we know that the DBC is always smaller or equal in size. In the experiments presented in Figure 3.5, the difference is very important and ranges

from one to four orders of magnitude.

3.3.6 Discovering DBC

In this section we describe two algorithms to mine the DBC. Two main strategies have been proposed to explore the search space during frequent set mining: breadth-first (e.g., **APRIORI**) and depth-first (e.g., **FP-Growth**, **Tree Projection**). Each of them has its pros and cons, and even if we consider only the extraction time criterion there is no always-winning strategy as we will show in Section 3.3.7. So, in this section we consider both strategies and we describe the corresponding algorithms. For each, we give an abstract version to find $FreqDisjFree(r, \sigma)$ and $FreqDisjFree\mathcal{B}d^{-}(r, \sigma)$, and then we describe the key implementation issues.
The *anti-monotonicity* of disjunction-freeness w.r.t. itemset inclusion is important for an efficient mining (Lemma 14). Actually, the combined property "frequent and disjunction-free" is used. It is also anti-monotone as stated by the following lemma.

Lemma 15. *Let r be a binary database over R, and X be an itemset, $X \subseteq R$. For all $Y \subseteq X$ if $X \in FreqDisjFree(r, \sigma)$ then $Y \in FreqDisjFree(r, \sigma)$.*

Proof. Suppose that $X \in FreqDisjFree(r, \sigma)$. Let us consider any $Y \subseteq X$.
Since $X \in FreqDisjFree(r, \sigma)$, $X \in Freq(r, \sigma)$ and $X \in DisjFree(r)$. According to the anti-monotonicity of support (stated by Corollary of Lemma 1), $Y \in Freq(r, \sigma)$. Similarly, the anti-monotonicity of disjunction-freeness (stated by Lemma 14) implies that $Y \in DisjFree(r)$.
Finally, $Y \in Freq(r, \sigma) \cap DisjFree(r) = FreqDisjFree(r, \sigma)$. □

During the extraction of the DBC, according to the contrapositive of this property, when an itemset is not frequent disjunction-free then there is no need to consider any of its proper supersets. This pruning criterion will be applied in breadth-first and depth-first strategies.

Breadth-First Extraction - Abstract Algorithm

Algorithm 12 presents the pseudo-code of **HLin-Ex** — an algorithm mining the DBC in a levelwise manner. It is formulated as an instance of the generic levelwise-search algorithm presented in [Mannila 97b]. It explores iteratively the itemset lattice (w.r.t. set inclusion) levelwise, starting from the singleton itemsets (line 1) and stopping at the level of the largest itemset from $DBC(r, \sigma)$, or, less commonly, one level further. At each iteration, it scans the database (line 3) to find which itemsets of the current level are frequent disjunction-free sets. Then, it generates candidates for the next iteration considering only the itemsets of the next level for which all proper subsets are frequent disjunction-free sets (line 6).

Note that the empty set is added without test (line 9) — it is known to be frequent, and it is disjunction-free, because there cannot be any disjunctive rule rule based on $\emptyset$ (see Definition 32), in particular any valid disjunctive rule.
Using the anti-monotonicity of the combined property stated by Lemma 15 and the correctness result of the levelwise search algorithm of [Mannila 97b], the following theorem is straightforward.

Theorem 10 (Correctness of HLin-Ex). *Given r a binary database over a set of items R, and σ an absolute support threshold, Algorithm 12 computes all frequent disjunction-free sets, and all frequent sets of the negative border of frequent disjunction-free sets.*

Breadth-First Extraction - Implementation Issues

Techniques similar to the ones presented in Section 2.2 for **APRIORI**. The candidate generation (line 6) is made using a join-based **APRIORI-Gen** function, the data set is linearly scanned to count supports of candidates (line 3) and the itemset support counters are updated w.r.t. a row of the data set using a prefix-tree data structure.
We implemented line 6 as a call to **APRIORI-Gen** function. That function uses only σ-frequent disjunction-free sets of size i to generate candidates of size $i+1$. Therefore, we partition $\mathcal{FDF}$ into $\mathcal{FDF}_i$ according to itemset size, and use only the portion corresponding to itemsets of size i for the call.
A specific aspect is the implementation of the disjunction-freeness test (lines 4 and 5). When we need to test this property for an itemset X, we already know the supports of all its subsets. So we check if there exist A and B, items of X, such that $Sup(r,X) = Sup(r,X\setminus\{A\}) + Sup(r,X\setminus\{B\}) - Sup(r,X\setminus\{A,B\})$. By Lemma 13 and Definition 33,

Algorithm 12. (HLin-Ex)

Input: *r a binary database over a set of items R, $\sigma \in [0,|r|]$ an absolute support threshold.*
Output: $DBC(r,\sigma)$.

1: **let** $\mathcal{C} := \{\{A\} : A \in R\}$, $i := 1, \mathcal{FDF} := \emptyset$, $\mathcal{FB} := \emptyset$;
2: **while** $\mathcal{C} \neq \emptyset$ **do**
3: *Scan r and compute supports of itemsets in $\mathcal{C}$;*
4: $\mathcal{FDF} := \mathcal{FDF} \cup \{X : X \in \mathcal{C} \wedge X$ *is σ-frequent disjunction-free in r*$\}$;
5: $\mathcal{FB} := \mathcal{FB} \cup \{X : X \in \mathcal{C} \wedge X$ *is σ-frequent and not disjunction-free in r*$\}$;
6: $\mathcal{C} := \{X : X \subseteq R$ *and* $|X| = i+1$ *and* $\bigwedge_{Y \subset X} |Y| = i \Rightarrow Y \in \mathcal{FDF}\}$;
7: **let** $i := i + 1$;
8: **od**
9: **output** $<$ $\{<X, support\ of\ X> : X \in \mathcal{FDF}\} \cup \{\emptyset, |r|\}$,
$\{<X, support\ of\ X> : X \in \mathcal{FB}\}$ $>$;

X is disjunction-free if and only if no such a pair of items exists.

Algorithm 13 shows the pseudo-code of the function testing the disjunction-freeness of an itemset Z. The function searches for any pair of items A, B such that the above support relationship is valid (which corresponds to a valid simple disjunctive rule basen on Z). The meaning of its result if it contains two different items A, B is that $A, B \in Z$ and $Z \setminus \{A, B\} \Rightarrow A \vee B$ is valid. When the result contains only one item A, it means that $A \in Z$ and $Z \setminus \{A\} \Rightarrow A \vee A$ is valid. Finally, when the result is empty, it corresponds to the case where Z is a disjunction-free set. If there are more than one valid rule based on Z, the function is supposed to report any of them.

FindDisjRule uses the supports of its proper subsets of size $|X| - 1$ and $|X| - 2$. Since all these proper subsets of X are already in $\mathcal{FDF}_{|X|-1}$ and $\mathcal{FDF}_{|X|-2}$ when the test is to be performed for X in lines 4 and 5, we can simply call **FindDisjRule**$(X, Sup(r, X), \mathcal{FDF}_{|X|-1} \cup \mathcal{FDF}_{|X|-2})$.

The test described above is quite efficient (we use only already counted support values) and may be integrated easily in an implementation of **HLin-Ex**.

Depth-First Extraction - Abstract Algorithm

Algorithm 14 shows the pseudo-code of the **VLin-Ex**, an algorithm computing the DBC in a depth-first manner. Techniques similar to the ones presented in Section 2.2 for **FP-Growth** and **CLOSET** are used.

The algorithm is given by means of a recursive function $FindFDFS$ which explores the itemset lattice in a depth-first manner. A call $FindFDFS(X, r, \sigma, Tail)$ finds in r the σ-frequent disjunction-free sets in a well-defined partition of the search space. The algorithm

Algorithm 13. (FindDisjRule)

Input: *Itemset Z, S support of Z, collection of itemsets $\mathcal{C}$ including all proper subsets of Z of size $|Z| - 1$ and of size $|Z| - 2$, the supports of itemsets in $\mathcal{C}$ are also supposed to be accessible.*

Output: *Itemset $P = \{A, B\}$ corresponding to a valid rule $Z \setminus \{A, B\} \Rightarrow A \vee B$ (see description in text).*

1: *Find any $A, B \in Z$* **such that**
  support of $Z \setminus \{A\}$ + support of $Z \setminus \{B\} = S$ + support of $Z \setminus \{A, B\}$;
2: **if** *such A, B exist* **then** *// in this case $Z \setminus \{A, B\} \Rightarrow A \vee B$ is valid*
3: **let** $P := \{A, B\}$;
4: **else**
5: **let** $P := \emptyset$;
6: **fi**
7: **output** P;

uses the divide-and-conquer strategy as follows. A call to the function will consider only the partition of the search space corresponding to proper supersets of X. Moreover, this subspace is further reduced to subsets of $X \cup Tail$, i.e., proper supersets of X that have not been examined earlier (see the explanation accompanying **FP-Growth** for details about the search-space traversal).
To compute the DBC we call $FindFDFS(\emptyset, r, \sigma, R)$, which returns $<\mathcal{FDF}, \mathcal{FB}>$ and we have:

$$\{<X, Sup(r, X)> : X \in FreqDisjFree(r, \sigma, \delta)\} = \mathcal{FDF} \cup \{<\emptyset, |r|>\}$$

$$\{<X, Sup(r, X)> : X \in FreqDisjFree\mathcal{B}d^{-}(r, \sigma, \delta)\} \subseteq \mathcal{FB}$$

As in case of **HLin-Ex**, in **VLin-Ex** the empty set can also be added without test — it is known to be frequent, and it is disjunction-free, because there cannot be any disjunctive rule rule based on $\emptyset$ (see Definition 32), in particular any valid disjunctive rule.
The computation of the exact collection of frequent sets of the negative border can be performed in a straightforward post-processing step using $\mathcal{FB}$ — non-minimal itemsets (w.r.t. set inclusion) of this collection must simply be removed to obtain the collection of frequent sets of the negative border.
Considering only supersets of X by the call $FindFDFS(X, r, \sigma, Tail)$ enables an efficient determination of support and of disjunction-freeness, because, only a restricted part of the database is required. Moreover, this restricted part of the database is further restricted in a recursive manner for recursive calls. Thus, in case of most calls to it, $FindFDFS$ works on a relatively small fraction of the database. Precautions should be however taken at this stage.
To test the disjunction-freeness, we wish to call the function **FindDisjRule**. We cannot use it directly however, because in depth-first approaches when X is considered we do not necessarily have the information about all subsets of X (to check $Sup(r, X) = Sup(r, X \setminus \{A\}) + Sup(r, X \setminus \{B\}) - Sup(r, X \setminus \{A, B\})$). For testing disjunction-freeness of the itemset X, **FindDisjRule** needs the supports of all subsets of X with up to 2 items missing. Therefore, for the abstract version of the algorithm, we consider the following function, denoted $\mathcal{A}ug\mathcal{M}(r, X)$, which selects the rows of a database so that the supports of subsets of X of size at least $|X| - 2$ remain preserved.

Definition 38. Given r a binary database over R and an itemset X, the augmented set of rows matching X is defined as $\mathcal{A}ug\mathcal{M}(r, X) = \{t \in r : |X \setminus t| \leq 2\}$.

Lemma 16 (anti-monotone $\mathcal{A}ug\mathcal{M}$). *Let r be a binary database over R, and X, Y be sets of items such that $Y \subseteq X \subseteq R$. Then $\mathcal{A}ug\mathcal{M}(r, X) \subseteq \mathcal{A}ug\mathcal{M}(r, Y)$.*

Proof. Let t be any row in r that belongs to $\mathcal{A}ug\mathcal{M}(r, X)$. From Definition 38, a row t belongs to $\mathcal{M}(r, X)$ iff $|X \setminus t| \leq 2$. $Y \subseteq X$ implies that $Y \setminus t \subseteq X \setminus t$. It follows that

$|Y \backslash t| \leq |X \backslash t| \leq 2$ and further that $t \in \mathcal{A}ug\mathcal{M}(r, Y)$. Thus, $\mathcal{A}ug\mathcal{M}(r, X) \subseteq \mathcal{A}ug\mathcal{M}(r, Y)$. □

Using this lemma, we can apply the incremental computation of the fraction of the binary database, when recursively calling $FindFDFS$ and still have the necessary information to test the disjunction-freeness.
The correctness of Algorithm 14 is stated by the following theorem.

Theorem 11 (Correctness of VLin-Ex). *Given r a binary database over a set of items R, and σ an absolute support threshold, the call $FindFDFS(\emptyset, r, \sigma, R)$ returns $<\mathcal{FDF}, \mathcal{FB}>$, and we have $\{<X, Sup(r, X)> : X \in FreqDisjFree(r, \sigma, \delta)\} = \mathcal{FDF} \cup \{<\emptyset, |r|>\}$ and $\{<X, Sup(r, X)> : X \in FreqDisjFree\mathcal{B}d^{-}(r, \sigma, \delta)\} \subseteq \mathcal{FB}$.*

Proof. First of all, we recall that the list $ordered\_list(X)$ is the permutation of all items in X that is sorted in ascending order according to the linear order $\prec$ for items. $X[i]$ designates i^{th} element of $ordered\_list(X)$.

Let us now consider $<\mathcal{FDF}, \mathcal{FB}>$ returned by $FindFDFS(\emptyset, r, \sigma, R)$.

Algorithm 14. (VLin-Ex)

Function $FindFDFS(X, r, \sigma, Tail)$
Input: *X the itemset considered as the current starting point in the search space, r a binary database over a set of items R, σ an absolute support threshold, and $Tail$ a set of items such that the itemsets X and $X \cup Tail$ delimit the partition of the search-space to be considered by this call to the function.*
Output: *All pairs $<Y, Sup(r', Y)>$ such that Y verifies $X \subset Y \subseteq X \cup Tail$ and it is frequent disjunction-free set in r'.*

1: **let** $\mathcal{C} := \{X \cup \{A\} : A \in Tail\}$;
2: **let** $\mathcal{FDF} := \emptyset$, $\mathcal{FB} := \emptyset$;
3: *Scan r and compute supports of itemsets in $\mathcal{C}$;*
4: **for all** $A \in Tail$ *in the ascending order given by* $\prec$ **do**
5: **let** $Tail := Tail \setminus \{A\}$;
6: **let** $Y := X \cup \{A\}$; // $Y \in \mathcal{C}$
7: **if** Y *is a σ-frequent disjunction-free set in r* **then**
8: **let** $<\mathcal{FDF}', \mathcal{FB}'> := Find(Y, \mathcal{A}ug\mathcal{M}(r, Y), \sigma, Tail)$;
9: **let** $\mathcal{FDF} := \mathcal{FDF} \cup \mathcal{FDF}' \cup \{<Y, support\ of\ Y>\}$;
10: **let** $\mathcal{FB} := \mathcal{FB} \cup \mathcal{FB}'$;
11: **elsif** Y *is a σ-frequent in r* **then**
12: **let** $\mathcal{FB} := \mathcal{FB} \cup \{<Y, support\ of\ Y>\}$;
13: **fi**
14: **od**
15: **output** $<\mathcal{FDF}, \mathcal{FB}>$;

Below, we show that $\bigwedge_{Y \in FreqDisjFree(r,\sigma) \setminus \{\emptyset\}} <Y, Sup(r,Y)> \in \mathcal{FDF}$ using an induction on $|Y|$.

Let $Y \in FreqDisjFree(r,\sigma)$ and $|Y| = 1$ (i.e., Y is a singleton itemset). Note that every singleton itemset is considered directly by $FindFDFS(\emptyset, r, \sigma, R)$ (line 6) and therefore if Y is σ-frequent disjunction-free (line 7) we have $Y \in \mathcal{FDF}$ (line 9).

Induction hypothesis. Let Z be an itemset such that $|Z| \geq 2$. Suppose that the property holds for every itemset $X \subset Z$ such that $ordered\_list(X)$ is a prefix of $ordered\_list(Z)$.

Given $Y \in FreqDisjFree(r,\sigma)$, such that $|Y| = n+1 \geq 2$, let x be an n-element prefix of $ordered\_list(Y)$ and X be the set of items occurring in x. Note that $Y[n+1]$ is the item marking off the difference between Y and X.

By Lemma 15, $X \in FreqDisjFree(r,\sigma)$, and then, by the induction hypothesis, $<X, Sup(r,X)> \in \mathcal{FDF}$. Adding an itemset to $\mathcal{FDF}$ for the first time in line 9 is preceded by a call to $FindFDFS$ (line 8) with that itemset as the first parameter. It follows that $FindFDFS$ has been called at least once using $FindFDFS(X, r_X, \sigma, Tail_X)$, where r_X and $Tail_X$ are the corresponding instances of parameters r and $Tail$, at the time $FindFDFS$ is called.

Similarly, for every $k = \{1, \ldots, n\}$, $Find(\{X[1], X[2], \ldots, X[k]\}, \ldots)$ has been called in line 8. At any of these events, for every item B in the corresponding instance of $Tail$ such that $X[k] \prec B$, B had not yet been processed (due to the ascending order in which items are processed in line 4). This implies that B is not yet removed from the corresponding $Tail$ in line 5.

Note that for all $k = \{1, \ldots, n\}$ the property $X[k] \prec Y[n+1]$ holds. Otherwise, the list $ordered\_list(Y)$ would not have followed the ascending order of items, which would have contradicted with how $ordered\_list(Y)$ is defined.

Combining the last two properties proves that $Y[n+1]$ is in $Tail_X$.

Let us focus on the call $FindFDFS(X, r_X, \sigma, Tail_X)$. This call starts by the following candidate generation step $\mathcal{C} := \{X \cup \{A\} : A \in Tail_X\}$. We know that $Y[n+1] \in Tail_X$. Therefore, Y is generated as candidate (in line 1) and tested for sufficient support and disjunction-freeness (in line 7). Since $Y \in FreqDisjFree(r,\sigma)$, Y is collected in $\mathcal{FDF}$.

The soundness of the algorithm (i.e., $\bigwedge_{Y \in \mathcal{FDF}} Y \in FreqDisjFree(r,\sigma)$) is immediate (lines 7 and 9).

Finally, we consider again the call $FindFDFS(\emptyset, r, \sigma, R)$, which returns $<\mathcal{FDF}, \mathcal{FB}>$. Let Y be an element of $FreqDisjFree\mathcal{B}d^-(r,\sigma)$ such that $|Y| = n+1$, and x be the n-element prefix of $ordered\_list(Y)$. Let X be the set of items occurring in x.

By Definition 36, $X \in FreqDisjFree(r,\sigma)$, and since $\bigwedge_{Z \in FreqDisjFree(r,\sigma) \setminus \{\emptyset\}} Z \in \mathcal{FDF}$, we have $X \in \mathcal{FDF}$ unless $X = \emptyset$. Using the same reasoning as above we know that Y is generated as candidate in line 1, thus tested in lines 7 and 11, and stored in line 12. Thus $\{<Y, Sup(r,Y)> : Y \in FreqDisjFree\mathcal{B}d^-(r,\sigma,\delta)\} \subseteq \mathcal{FB}$. □

Depth-First Extraction - Implementation Issues

We combined best features from state-of-the-art algorithms implementing a depth-first strategy for frequent set mining. In particular, we use a support counting technique similar to the one presented in [Agarwal 01], and a compact storage of a collection of rows (instances of the parameter r of $FindFDFS$) in a prefix-tree structure as described for **FP-Growth** in Section 2.2.

We wished to adopt as many optimisations described for **FP-Growth** as possible. Most of them could have been adopted directly. The use of $\mathcal{A}ug\mathcal{M}$ instead of $\mathcal{M}$ allows selecting only a fraction of rows for the recursive calls, leading to considerably smaller binary databases by deeper recursive calls to $FindFDFS$.

The loop of line 4 enumerates the items $A \in Tail$ following the ascending order of supports of the corresponding itemsets $X \cup \{A\}$ in r. This order of exploration has been inspired from **FP-Growth** to reduce the number of rows in partitions ($\mathcal{A}ug\mathcal{M}(r, Y)$) with higher number of items (long $Tail_Y$), when calling $FindFDFS(Y, \mathcal{A}ug\mathcal{M}(r', Y), \sigma, Tail_Y)$.

The challenging optimisation was the one preserving only the items contained in $Tail_Y$ for the construction of the prefix-tree for $\mathcal{M}(r', Y)$, when calling $FindFS(Y, \mathcal{M}(r', Y), \sigma, Tail_Y)$. This can be done for **FP-Growth** because the presence of all items Y is already checked by computing $\mathcal{M}(r', Y)$ and the exploration of the corresponding partition of the set-enumeration-tree neither contains items other than $Y \cup Tail_Y$ nor requires supports involving them.

In case of **VLin-Ex**, $\mathcal{A}ug\mathcal{M}$ does not guarantee the presence/absence of any single item, and thus if based only on that property the optimisation would not had been applicable — we would had had to preserve all items of $Y \cup Tail_Y$.

An alternative might be found by dropping the use of **FindDisjRule**. An equivalent test to the validity of the relationship $Sup(r, X) = Sup(r, X \setminus \{A\}) + Sup(r, X \setminus \{B\}) - Sup(r, X \setminus \{A, B\})$ is to check if $Sup(r, (Y \cup \{\overline{A}, \overline{B}\}) \setminus \{A, B\})$ is zero (see Lemma 11). The presence/absence status of all items of Y in the collection of rows matching $(Y \cup \{\overline{A}, \overline{B}\}) \setminus \{A, B\}$ is known (A and B are absent in all rows, others are present also in all rows). Therefore, we may split the collection of rows $\mathcal{A}ug\mathcal{M}(r', Y)$ into such collections for combinations of present/absent items of Y. Note that the rows in $\mathcal{A}ug\mathcal{M}(r', Y)$ are those and only those matching at most two negative items of Y. This observation not only permits to limit the numbmer of collections to be considered for an itemset Y, but also suggests the way to compute recursively $\mathcal{A}ug\mathcal{M}(r', Y)$ as follows.

Suppose that X is the parent of $Y = X \cup \{D\}$ in the set-enumeration-tree corresponding to the search-space explored by **VLin-Ex**. Consider the collection of rows $\mathcal{M}(r, G_X)$ matching a full clause G_X over X (see Definition 10) and preserving only the items from $Tail_X$. When calling recursively $FindFDFS(Y, \mathcal{A}ug\mathcal{M}(r, Y), \sigma, Tail_Y)$, we should incrementally select the rows of $\mathcal{A}ug\mathcal{M}(r, Y)$, i.e., (among others) $\mathcal{A}ug\mathcal{M}(\mathcal{M}(r, G_X), \{D\})$. If G_X contains 0 or 1 negative item, we preserve all rows, because they all match Y with at

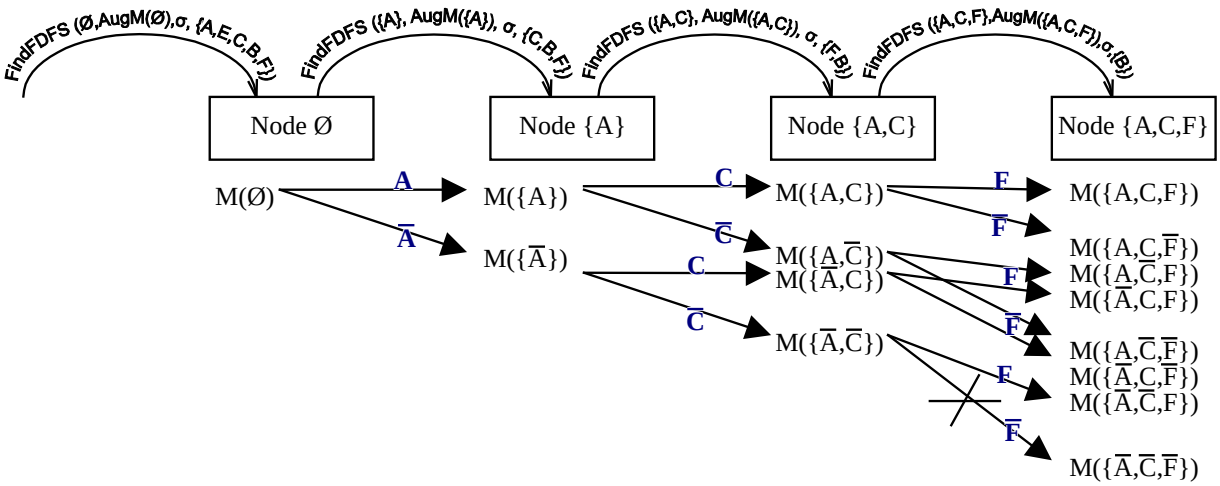

Figure 3.6: Illustration of the construction of $\mathcal{A}ug\mathcal{M}(r, \{A, C, F\})$ the node $\{A, C\}$ and its child $\{A, C, F\}$. Previous constructions are also shown. The call-stack is represented by arcs. r' parameter is omitted for $\mathcal{M}$.

most two items missing. If G_X contains 2 negative items, we select only the rows matching D (since the rows missing two items of X and not containing D would miss 3 items of Y). We partition the resulting collection of rows according to the missing items, in order to detect empty collection of rows $\mathcal{M}(r, (Y \cup \{\overline{A}, \overline{B}\}) \setminus \{A, B\})$, where A, B represent two items in Y (detection of non-disjunction-free sets).

Figure 3.6 represents schematically the construction of $\mathcal{A}ug\mathcal{M}(r, \{A, C, F\})$ in an imaginary context. Recursive steps for the parents of $\{A, C, F\}$ are also represented.

3.3.7 Extracting DBC in Practice

In this section, we consider the extraction times of the DBC and the condensed representation based on frequent closed sets.

To mine efficiently the frequent closed sets we use the algorithms **CLOSE** and **CLOSET**, both described in Section 2.2. Additionally, we report the extraction times of the **APRIORI** algorithm, which extracts all frequent sets. For more details on the used implementations, see Section 2.2.9.

We have run the following experiments on a PC with 256 MB of memory and an 800 MHz Pentium III processor under Linux operating system.

We compared the extraction times of frequent closed sets and of the DBC on the data sets that have been described in Section 3.3.5. We varied data sets and support thresholds in order to get a meaningful overview. The running times, in seconds, are given in Figure 3.7 (note that some axes are logarithmically scaled and that we use relative support thresholds, as in Section 3.3.5). The results are given for **HLin-Ex**, **VLin-Ex**, for the implementations of **CLOSE** and **CLOSET**, and additionally for the implementation of **APRIORI**.

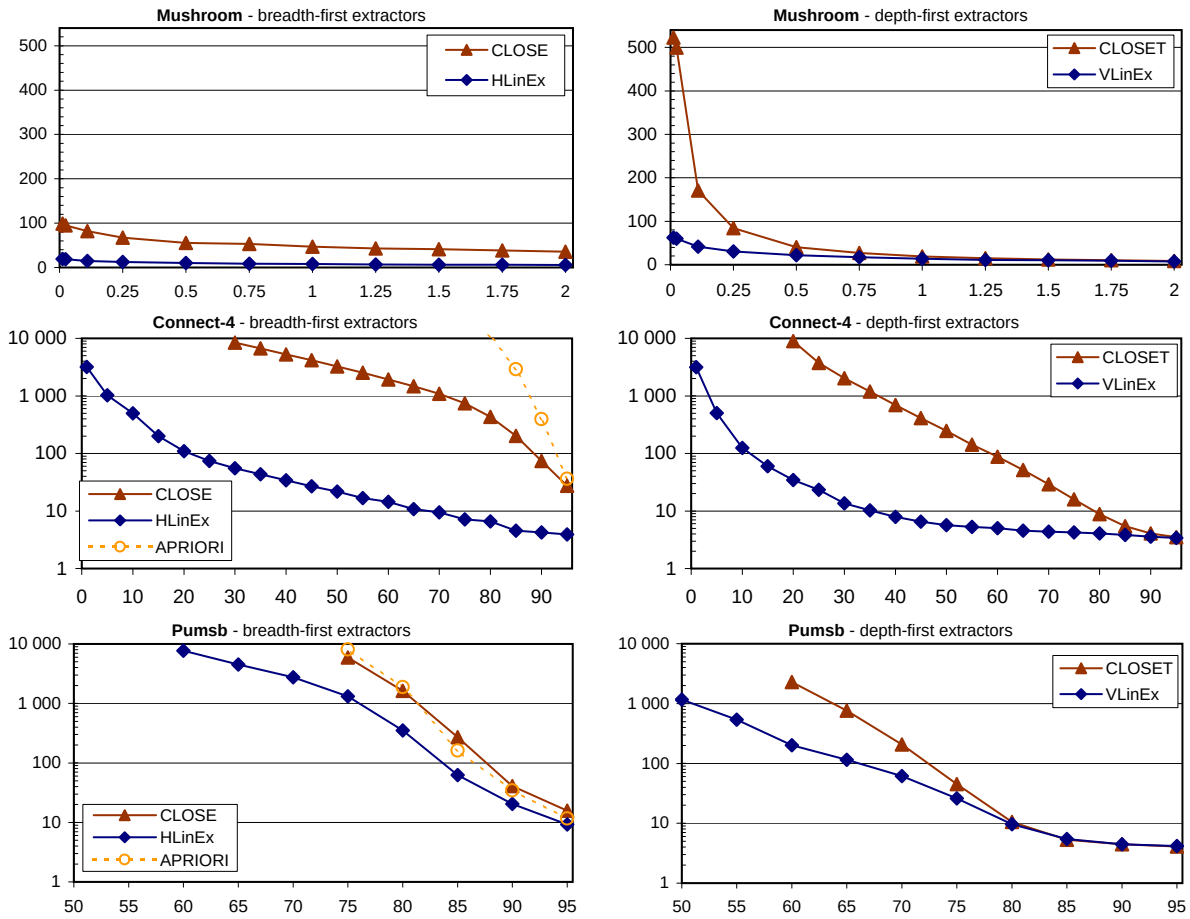

Figure 3.7: Experiments on the three data sets with breadth-first (left) and depth-first (right) algorithms.

On the **Mushroom** data set, we observed that extracting the DBC is advantageous (see Figure 3.7, upper graphics) especially using **HLin-Ex**. Observed speed-up was up to 5 times for breadth-first extractors and up to 8.5 times for depth-first extractors. With the extraction time above 1 hour, the curve corresponding to **APRIORI** is exceptionally not reported on the figure for this data set. Thus, we could magnify the difference between extraction times for frequent closed sets and the DBC.

The **Connect-4** data set is very difficult for mining frequent sets (see Figure 3.7, middle graphics), and mining directly frequent closed sets using **CLOSE** and **CLOSET** allows a significant improvement (compare the results for **CLOSE** with the ones for **APRIORI**). Further significant improvements (over **CLOSE** and **CLOSET**) can be achieved by using **HLin-Ex** and **VLin-Ex**. At lowest support thresholds for which we were able to extract frequent closed sets, we observed over 150 times faster extractions for **HLin-Ex** vs. **CLOSE** and up to 260 times faster extractions for **VLin-Ex** vs. **CLOSET**.

In the case of the last data set, i.e., of **PUMSB**, extracting frequent patterns was very difficult for all extractors, and **CLOSE** was not significantly advantageous over **APRIORI**. However, **HLin-Ex** and **VLin-Ex** offer an evident benefit at lower support thresholds (see Figure 3.7, lower graphics).

In nearly all experiments on the three data sets, the extraction of the DBC is significantly

more efficient than the extraction of the frequent closed sets. For the very difficult cases, the DBC can be extracted at lower support thresholds, using the same resources.

3.3.8 Summary of Results Related to DBC

In this section, we showed how disjunctive rules that are valid in a binary database can be put into use to define a condensed representation of frequent sets. We also presented two algorithms, **HLin-Ex** and **VLin-Ex**, computing the DBC. Experimental study of these algorithms and of the proposed representation showed that mining highly correlated data sets using the DBC outperforms **APRIORI** (computing all frequent sets) and **CLOSE** (computing frequent closed sets) in terms of condensation and extraction time.
As in case of the SRBC, the DBC may present very little or no benefits if we consider very sparse data sets. Their extraction could be actually a little slower.

Chapter 4

Applications

Obtaining frequent sets from the SRBC and the DBC (the representations described in Chapter 3) is the most basic application for these condensed representations. Since these condensed representations often represent intractably large collections of frequent sets (see Section 3.2.6 and 3.3.5), obtaining all frequent sets is not always possible. In that case, it does not mean we cannot proceed with mining tasks. Frequent set mining is often followed by some data mining tasks making use of frequent sets, and it is a common practice to use some well-defined subcollections of frequent sets and to discard remaining ones. In such cases, the results described in Section 4.1 could be refined to regenerate such subcollections, circumventing often voluminous collections of all frequent sets. For example, the regeneration procedures of all frequent sets (**Regen-Freq**, **Regen-Approx-Freq**) can be easily modified to extract frequent sets under constraints (extraction of frequent sets under constraints has been widely investigated, see e.g., [Srikant 97, Ng 98, Boulicaut 00d]).

Downstream applications include all applications of frequent sets, such as association rule derivation (with the equivalent tractability prerequisite as for frequent sets), derivation of association rules under constraints [Srikant 97, Ng 98, Boulicaut 00d], computing association rule covers (discussed shortly in Section 4.4), extracting association rules with negations or their covers (Section 4.5), generalized formulae support derivation (see [Bykowski 02b]), clustering techniques [Das 98, Han 98] and so on.

We do not discuss them all. We focus on providing the fundamentals that may benefit them.

Computing all association rules from the proposed condensed representations is straightforward by regenerating all frequent sets and then using Algorithm 1, unless we make use of lossy representations. For this particular choice, we need pay attention to errors on the evaluation functions (e.g., support, confidence) propagated from overestimates of the frequent set supports provided by an SRBC with $\delta > 0$. We discuss and experiment this aspect for support and confidence measures in Section 4.2.

Another issue is that computing association rule covers can be done without computing

all frequent sets. In [Bastide 00a] and in [Zaki 00] resp. Bastide *et al.* and Zaki have described methods to compute association rule covers[1]. Both require frequent closed sets and frequent 0-free sets for actual computation of the cover (the use of frequent closed sets is explicitely stated in both papers, the need of frequent 0-free sets is derived from the properties on other itemsets involved in the process of generating covers). Consequently, we give a special importance to show how to convert the lossless disjunction-bordered condensation into frequent closed sets and frequent 0-free sets. This can be done even when the collection of all frequent sets is intractably large; the procedure and experiments are detailed in Section 4.3. In order to circumvent regenerating all frequent sets this procedure uses quite dense theoretical contents.
The lossless variant of strong-rule-bordered condensation already contains 0-free sets and sole regeneration of frequent closed sets need to be addressed in order to derive rule covers. This procedure is enclosed in the procedure converting the disjunction-bordered condensation into frequent closed sets and frequent 0-free sets, and therefore is not specifically detailed nor experimented.
Section 4.4 briefly recalls the results from the literature related to the association rule covers and shows the relevance of the results of Section 4.3 w.r.t. this goal.
Specific applications, taking better advantage of smaller size of the one of the condensed representations have been carried out or are still under development (see [Bykowski 02a, Bykowski 00, Bykowski 02b]).

4.1 Frequent Set Regeneration from Condensed Representations

In Chapter 3, we showed that regeneration of all frequent sets along with the value of the evaluation function of these patterns (i.e., support) from the condensed representations developed in this thesis is possible. We can regenerate that information without accessing the data set. It could lead to an extremely fast regeneration process.
In this section, we devise an efficient regeneration process. We provide theoretical properties, algorithms and experimental studies. The quasi-constant time complexity per regenerated frequent sets can be achieved.
The regeneration from the DBC has been described in [Bykowski 03], while the regeneration from the SRBC has partly been reported in [Boulicaut 03].
We start with lossless representations, because of a lesser number of issues to be addressed.

[1] By cover we mean here a non-redundant subcollection of all association rules permitting regenerate all association rules if necessary. For a further discussion see Section 4.4.

4.1.1 Regeneration from DBC

In this section, we present a levelwise algorithm, called **Regen-Freq**, that regenerates all frequent sets from the DBC. Here, we assume that we regenerate $Freq(r, \sigma)$ and the corresponding supports given $DBC(r, \sigma)$. Nonetheless, the support thresholds may be different — we might regenerate $Freq(r, \sigma')$, where $\sigma' \geq \sigma$, by skipping from the $DBC(r, \sigma)$ the itemsets whose supports are below σ' and applying the same algorithm to the reduced input (and the increased support threshold). It is quite important, because it means that we do not regenerate unnecessary frequent sets only to discard them when the regeneration finishes — we eliminate them at the early stage and regenerate only the required ones.
We give **Regen-Freq** in a detailed form. The actual implementation integrates few straightforward optimizations. Most important hints will be mentioned throughout this section.

The data structure corresponding to one frequent set will be called a node. Here is its structure:

itemset	: set of items
support	: integer
prunedby	: set of items

Let $\mathcal{FS}_i$ denote the collection of nodes corresponding to itemsets of size i. A collection of such nodes is stored in an itemset-prefix-tree, a structure allowing efficient access to the node given the value of *itemset* (see [Agrawal 94], or Section 2.2.1).
For a node N, the access to these three fields is denoted respectively $N.itemset$, $N.support$ and $N.prunedby$. $N.itemset$ corresponds to a frequent set[2] and $N.support$ is its support. $N.prunedby$ is a set of at most two items. Its meaning if it contains two different items A, B is that $A, B \in N.itemset$ and $N.itemset \backslash \{A, B\} \Rightarrow A \vee B$ is valid. When $N.prunedby$ contains only one item A, it means that $A \in N.prunedby$ and $N.itemset \setminus \{A\} \Rightarrow A \vee A$ is a valid disjunctive rule. Finally, when $N.prunedby$ is empty, it corresponds to the case where $N.itemset$ is a disjunction-free set. If there are more than one valid rule based on $N.itemset$, $N.prunedby$ is supposed to report any of them.
Note the similarity of the above specification to the specification of the function **Find-DisjRule** presented as Algorithm 13 in Section 3.3.6. When necessary, we will call that function to fill the field $N.prunedby$[3].
Now, we can give the algorithm **Regen-Freq**, which regenerates all frequent sets and their supports from the DBC. It is detailed as Algorithm 15.

[2]The field *itemset* is represented in the algorithm for clarity, but using an *itemset*-based prefix-tree structure provides a much more compact storage of the items in this itemset.

[3]Our implemented extractors store in the materialization of the DBC the values of *prunedby* corresponding to the frequent sets of the negative border.

Algorithm 15. (Regen-Freq)

Input: σ, $DBC(r,\sigma)$ *of the form* $<\mathcal{F},\mathcal{B}>$.
Output: $Freq(r,\sigma)$ *and their supports.*

1: **for all** $<X,S_X> \in \mathcal{F}$ **do** // *Disjunction-free input sets*
2: *Create new node* N_X *in* $\mathcal{FS}_{|X|}$ *with* $N_X.support := S_X$, $N_X.itemset := X$, $N_X.prunedby := \emptyset$;
3: **od**
4: **for all** $<X,S_X> \in \mathcal{B}$ **do** // *Non-disjunction-free input sets*
5: *Create new node* N_X *in* $\mathcal{FS}_{|X|}$ *with* $N_X.support := S_X$, $N_X.itemset := X$;
6: **if** $|X| = 1$ **then let** $N_X.prunedby := X$;
7: **else let** $N_X.prunedby := \mathbf{FindDisjRule}(X, S_X, \mathcal{FS}_{|X|-1} \cup \mathcal{FS}_{|X|-2})$;
8: **fi**
9: **od**

10: **let** $i := 1$;
11: **while** $\mathcal{FS}_i \neq \emptyset$ **do**
12: **let** $\mathcal{C}_{i+1} := \mathbf{APRIORI-Gen}(\mathcal{FS}_i)$;
13: **for all** $X \in \mathcal{C}_{i+1}$ **do**
14: *Find node* N_Y *in* $\mathcal{FS}_i$ **such that** $N_Y.itemset \subset X$ **and** $N_Y.prunedby \neq \emptyset$;
15: **if** *such* N_Y *exists* **then**
16: **case** $|N_Y.prunedby|$ **of** :
17: *2:*
18: **let** A, B *be items* **such that** $N_Y.prunedby = \{A, B\}$;
19: **let** N_A, N_B *be the nodes in* $\mathcal{FS}_i$ *and* N_{AB} *be the node in* $\mathcal{FS}_{i-1}$ **such that** $N_A.itemset = X \setminus \{A\}$ **and** $N_B.itemset = X \setminus \{B\}$ **and** $N_{AB}.itemset = X \setminus \{A, B\}$;
20: **let** $S := N_A.support + N_B.support - N_{AB}.support$;
21: *1:*
22: *Find node* N *in* $\mathcal{FS}_i$ **such that** $N.itemset = X \setminus N_Y.prunedby$;
23: **let** $S := N.support$;
24: **end case**
25: **if** $S \geq \sigma$ **then**
26: *Create new node* N_X *in* $\mathcal{FS}_{|X|}$ *with* $N_X.support := S$, $N_X.itemset := X$, $N_X.prunedby := N_Y.prunedby$;
27: **fi**
28: **fi**
29: **od**
30: **let** $i := i + 1$;
31: **od** ;
32: **output** $\{<N.itemset, N.support> : N \in \bigcup_{j<i} \mathcal{FS}_j\}$;

In lines 1–9, Algorithm **Regen-Freq** loads the input and converts it to a collection of nodes, one node per frequent set. The nodes corresponding to frequent sets of the negative border are marked by a nonempty content of the field *prunedby*.
In line 11, the algorithm enters a loop corresponding to increasing sizes of regenerated frequent sets. Within the i^{th} iteration of the loop, the algorithm uses $\mathcal{FS}_i$ and $\mathcal{FS}_{i-1}$ (the collections of the nodes corresponding to frequent sets of size i and $i-1$) to produce $\mathcal{FS}_{i+1}$ (i.e., frequent sets of size $i+1$). Candidate itemsets, i.e., itemsets of size $i+1$ having all their proper subsets frequent, are computed in line 12 and stored in $\mathcal{C}_{i+1}$. An additional pruning is performed in lines 14–15. The condition ensures that at least one proper subset of the current candidate itemset is not disjunction-free[4]. Otherwise (i.e., when all proper subsets are disjunction-free), the candidate should be skipped. Two cases fit skipped candidates: the candidate is not frequent, then we do not want to regenerate it anyway, or the candidate itemset is frequent. A frequent sets of size $i+1$, for which all subsets of size i are frequent and disjunction-free, is either in $FreqDisjFree(r,\sigma)$ (when it is disjunction-free) or in $FreqDisjFree\mathcal{B}d^{-}(r,\sigma)$ (non-disjunction-free set). In both cases, it is in the DBC, and thus the corresponding node has already been created in lines 1–9.
In lines 16–24, for each retained candidate itemset X, **Regen-Freq** searches the information necessary to restore its support. According to Lemma 14, X is not disjunction-free, because it is a proper superset of an itemset that is not disjunction-free. According to Lemmas 12 and 13, **Regen-Freq** finds a valid simple disjunctive rule leading to an equation that can be used to derive the support of X from the support of some of its subsets. Then, if $Sup(r,X) \geq \sigma$, **Regen-Freq** creates a node N_X and fills it with the corresponding information (line 26). As previously, the value stored in $N_X.prunedby$ corresponds to any of the valid rules based on X, if there are more than one such rule.

Theorem 12 (Correctness of Regen-Freq). *The algorithm* **Regen-Freq** *outputs all and only σ-frequent sets along with their supports.*

Proof. By Lemma 1, if X is σ-frequent all its subsets are σ-frequent. Therefore, the proof is made by induction on $|X|$.
Hypothesis. Suppose that for every frequent set $X \neq \emptyset$, the algorithm **Regen-Freq** correctly constructs the nodes corresponding to all proper subsets of X, notably that they are all present in their respective collections $\mathcal{FS}_j$, and that their fields *itemset*, *support* and *prunedby* are correctly filled.
First, we consider a frequent set X. We are going to show that it is in the output.

Let us consider the case where X belongs to $DBC(r,\sigma)$. Then, X is σ-frequent and it is in the input. Therefore the corresponding node N_X is created in line 2 or in lines 5–8 and

[4]Since **APRIORI-Gen** already accesses all nearest proper subsets of each candidate, then in the implementation of the algorithm the condition used in lines 14–15 is checked within the modified **APRIORI-Gen**.

is never removed. Therefore, it will be output in line 32.

Suppose now that X is frequent, but does not belong to the DBC (is neither in $FreqDisjFree(r,\sigma)$ nor in $FreqDisjFree\mathcal{B}d^{-}(r,\sigma)$). Therefore X is not disjunction-free. Moreover, since X is σ-frequent, but not in $FreqDisjFree\mathcal{B}d^{-}(r,\sigma)$, there exists $Y \subset X$, such that $|Y| = |X| - 1$ and Y is not disjunction-free. The empty itemset is disjunction-free, thus $|Y| \geq 1$ and $|X| \geq 2$. Since the nodes corresponding to all subsets of X are correctly created (by *Hypothesis*), there exists a node N_Y corresponding to Y (line 15), and it has a nonempty, correct value of $N_Y.prunedby$.
By the same hypothesis, the nodes corresponding to all proper subsets of X are present in their respective collections $\mathcal{FS}_j$. Since $|X| \geq 2$, X is produced as candidate itemset by **APRIORI-Gen** in line 12 and it is considered in lines 16–27. The corresponding node is actually created in line 26, because X is σ-frequent. It will be in the output, because a created node is never removed.

For the equivalence, suppose that the output contains X along with its support. We are going to show that X is σ-frequent.
Observe that nodes are created in lines 2, 5–8 and 26. If the node N_X with $N_X.itemset = X$ is created in line 2 or in lines 5–8, then X belongs to the DBC, and thus is σ-frequent. If the node N_X is created in line 26, X is σ-frequent, because the corresponding value S is checked against σ in line 25. S is the correct support of X according to the fact that X is not disjunction-free, to the valid simple disjunctive rule $Y \setminus \{A, B\} \Rightarrow A \vee B$ corresponding to the value of $N_Y.prunedby$, to Lemma 13 and to *Hypothesis* ($N_Y.prunedby$ is correct and the support inference equation is applied on correct values of subset supports).
Therefore, if the itemset X is in the output, it is σ-frequent. □

4.1.2 Regeneration from SRBC

As shown in Section 3.2.3, we can regenerate all frequent sets from the SRBC with a bounded error on their supports. If that error remains relatively low, we could devise a functional approximate frequent set derivation procedure from the SRBC.
So, first we study the error in practice and show that it remains very low as compared to the theoretical bounds given previously. Later, we propose an efficient method to derive frequent sets in an approximate way, according to the previous theoretical results.

Approximation Error in Practice

In this section we report the practical error made on frequent set supports when using the ϵ-adequate representation based on the SRBC. The experiments are made on the **PUMSB*** data set used in the experiments on the extraction of the SRBC (Section 3.2.6).

itemset size	1	2	3	4	5	6	7	8
average abs. sup. error	0	0.36	1.17	2.14	3.24	4.33	5.31	6.12
average rel. sup. error	0	0.002%	0.007%	0.012%	0.019%	0.026%	0.032%	0.038%
maximal abs. sup. error	0	18	20	37	37	39	39	45
maximal rel. sup. error	0	0.11%	0.13%	0.18%	0.22%	0.24%	0.24%	0.29%

itemset size	9	10	11	12	13	14	15	16
average abs. sup. error	6.80	7.40	7.92	8.39	8.82	9.22	9.58	9.86
average rel. sup. error	0.042%	0.046%	0.050%	0.054%	0.057%	0.060%	0.062%	0.064%
maximal abs. sup. error	45	45	45	44	38	31	24	15
maximal rel. sup. error	0.29%	0.29%	0.29%	0.28%	0.26%	0.19%	0.15%	0.10%

Table 4.1: Error observed on frequent set supports by itemset size (**PUMSB***).

itemset support (%)	[30,40]	(40,50]	(50,60]	(60,70]	(70,80]	(80,90]	(90,100]
average abs. sup. error	6.20	3.29	0.07	0	0	0	0
average rel. sup. error	0.039%	0.016%	2.8×10^{-6}	0	0	0	0
maximal abs. sup. error	45	38	9	0	0	0	0
maximal rel. sup. error	0.29%	0.19%	0.03%	0	0	0	0

Table 4.2: Error observed on frequent set supports by interval of support (**PUMSB***).

Additionally, we report the results of experiments on a **PUMS** data set of Kansas in a less favorable context.

In the **PUMSB*** data set (denoted r_1 in this section), for the support threshold $\sigma = 0.3 * |r_1|$, there are 432 699 frequent sets and the largest has $n = 16$ items. We computed the condensed representation $SRBC(r_1, 0.3 * |r_1|, 20)$, which contains 11 579 elements.

Theoretical error bounds for the frequent set support approximation can be determined using Theorem 4 as follows. In this experiment, the maximal absolute support error is $\delta * n = 20 * 16 = 320$ rows. The maximal relative support error can be obtained assuming that the maximal theoretical absolute error occurs on the frequent set of minimal support (i.e., σ). The **PUMSB*** data set contains $|r_1| = 49\,046$ rows. So, the maximal relative support error is $\delta * n/\sigma \approx 2.17\%$.

The support of each of the 432 699 frequent sets is approximated using the collection $SRBC(r_1, 0.3 * |r_1|, 20)$ and Theorem 4 and then compared to the exact support. We *observed* the following errors. The maximal absolute support error is 45 rows, and the maximal relative support error is 0.29%. The average absolute support error is 6.01 rows and the average relative support error is 0.037%. Tables 4.1 and 4.2 show that these errors remain very low even for for low supports and largest frequent sets.

In the above experiment the value of δ is small w.r.t. the minimal support required. The ratio is $20/(0.3 * 49\,046) \approx 0.136\%$. We now report another experiment where the value of δ represents more than 1% of the minimal support required, and thus is likely to greatly increase the value of the error.

itemset size	1	2	3	4	5	6	7	8
average abs. sup. error	0	0.24	0.65	1.10	1.53	1.92	2.31	2.75
average rel. sup. error	0	0.03%	0.07%	0.13%	0.18%	0.24%	0.31%	0.38%
maximal abs. sup. error	0	6	10	12	14	18	18	18
maximal rel. sup. error	0	1.1%	1.3%	2.1%	2.7%	3.1%	3.1%	3.1%

itemset size	9	10	11	12	13
average abs. sup. error	3.28	3.90	4.58	5.20	5.50
average rel. sup. error	0.47%	0.58%	0.71%	0.83%	0.88%
maximal abs. sup. error	18	18	18	15	11
maximal rel. sup. error	3.1%	2.9%	2.9%	2.9%	2.0%

Table 4.3: Error observed on frequent set supports by itemset size (**PUMS** of Kansas).

itemset support (%)	[5,10]	(10,20]	(20,30]	(30,40]	(40,50]	(50,60]	(60,70]	(70,80]	(80,90]	(90,100]
average abs. sup. error	2.16	2.03	2.22	2.03	1.25	1.70	0.66	0	0	0
average rel. sup. error	0.337%	0.159%	0.089%	0.063%	0.027%	0.031%	0.010%	0	0	0
maximal abs. sup. error	18	14	10	10	5	6	6	0	0	0
maximal rel. sup. error	3.11%	1.17%	0.47%	0.33%	0.12%	0.10%	0.10%	0	0	0

Table 4.4: Error observed on frequent set supports by interval of support (**PUMS** of Kansas).

The data set used in this experiment is a **PUMS** data set of Kansas state[5] (denoted r_2 in the following). We use a version of this data set that has been preprocessed at the University of Clermont-Ferrand (France) in Prof. L. Lakhal's research group. We have reduced this data set to 10 000 rows and 317 items to be able to extract all σ-frequent itemsets at a low support threshold. For $\sigma = 0.05$ (500 rows), there are 90 755 frequent sets and the largest has $n = 13$ items. We computed $SRBC(r_2, 0.05 * |r_2|, 6)$, which contains 4771 elements.

In this experiment, the maximal absolute support error is $\delta * n = 6 * 13 = 78$ rows. The maximal relative support error is $\delta * n/\sigma = 15.6\%$ ($|r_2| = 10\,000$ rows in the experiment). The supports of the frequent sets are approximated using $SRBC(r_2, 0.05 * |r_2|, 6)$ and compared to the exact supports. We *observed* the following errors. The maximal absolute support error is 18 rows, and the maximal relative support error is 3.1%. The average absolute support error is 2.12 rows and the average relative support error is 0.28%. A more detailed distribution of the errors is given in Tables 4.3 and 4.4. These results show that the errors remain low in practice even when the value of δ is high w.r.t. the minimal support.

Regeneration Algorithm from SRBC

Above, we have seen that the SRBC is an interesting representation of frequent sets, even if we authorize it to be lossy (i.e., $\delta > 0$). In this section, we present a levelwise

[5] ftp://ftp2.cc.ukans.edu/pub/ippbr/census/pums/pums90ks.zip

algorithm, called **Regen-Approx-Freq**, that regenerates approximately frequent sets from the SRBC. It should be made clear that unless $\delta = 0$ given the overestimated nature of regenerated supports, some infrequent sets with supports very close to the threshold σ may also be output by the algorithm. Involved itemsets are the ones for which the support approximation procedure described in Section 3.2.3 gives the answer greater or equal than σ.

Regen-Approx-Freq is similar to the algorithm **Regen-Freq** presented in Section 4.1.1. As in the latter, it is possible to regenerate frequent sets with a different (higher) support threshold σ' by skipping from the input collection $SRBC(r, \sigma, \delta)$ the itemsets (and their supports) that turned to infrequent due to the new support threshold. In that case, the regeneration of the unnecessary frequent sets is eliminated very early.

The data structure corresponding to one frequent set will be called a node. Its structure is the following:

$itemset$	: set of items
$support$	: integer
$deltafree$	: Boolean

The signification of first two fields is the same as for **Regen-Freq**. The third field, $N.deltafree$, states if the itemset represented by the node is δ-free.

Now, we can give the algorithm **Regen-Approx-Freq**, which regenerates approximately frequent sets and their supports from the SRBC. It is detailed as Algorithm 16.

In lines 1–6, Algorithm **Regen-Approx-Freq** loads the input and converts it to a collection of nodes, one node per frequent set. The nodes corresponding to frequent sets of the negative border are marked by the field $deltafree$ set to $false$.

In line 8, the algorithm enters a loop corresponding to increasing sizes of regenerated frequent sets. Within the i^{th} iteration of the loop, the algorithm uses $\mathcal{FS}_i$ (the collections of the nodes corresponding to frequent sets of size i) to produce $\mathcal{FS}_{i+1}$ (i.e., frequent sets of size $i+1$). Candidate itemsets, i.e., itemsets of size $i+1$ having all their proper subsets frequent, are computed in line 9 and stored in $\mathcal{C}_{i+1}$. An additional pruning is performed in lines 11–12. The condition ensures that at least one proper subset of the current candidate itemset is not δ-free[6]. Otherwise (i.e., when all proper subsets are δ-free), the candidate should be skipped. Two cases fit skipped candidates: the candidate is not frequent, then we do not want to regenerate it anyway, or the candidate itemset is frequent. A frequent sets of size $i+1$, for which all subsets of size i are frequent and δ-free, is either in $FreqDeltaFree(r, \sigma, \delta)$ (when it is δ-free) or in $FreqDeltaFree\mathcal{B}d^{-}(r, \sigma, \delta)$ (non-δ-free set). In both cases, it is in the SRBC, and thus the corresponding node has already been created in lines 1–6.

[6]Since **APRIORI-Gen** already accesses all nearest proper subsets of each candidate, in the implementation of the algorithm the pruning of lines 11–12 as well as the computation shown as line 13 are performed within the modified **APRIORI-Gen**.

In line 13, for each retained candidate itemset X, **Regen-Approx-Freq** searches the information necessary to restore its support. According to Definition 30, to produce an overestimate of the support of X, we should consider only frequent δ-free subsets of X at this step. If we compute the minimum of supports over a larger collection of subsets of X, we will still produce an overestimate (due to anti-monotonicity of support), but it might be smaller than using frequent δ-free sets only. A smaller overestimate is simply a more precise estimate, so by deriving the supports from both, frequent δ-free sets and their negative border, we improve the result.
The regenerated itemsets obviously are not δ-free, so the field $deltafree$ is set to $false$ in line 14.

Algorithm 16. (Regen-Approx-Freq)

Input: σ, δ *and* $SRBC(r, \sigma, \delta)$ *of the form* $<\mathcal{F}, \mathcal{B}>$.
Output: $Freq(r, \sigma)$ *and their supports.*

1: **for all** $<X, S_X> \in \mathcal{F}$ **do** // *δ-free input sets*
2: *Create new node* N_X *in* $\mathcal{FS}_{|X|}$ *with* $N_X.support := S_X$, $N_X.itemset := X$, $N_X.deltafree :=$ **true** ;
3: **od**
4: **for all** $<X, S_X> \in \mathcal{B}$ **do** // *Non-δ-free input sets*
5: *Create new node* N_X *in* $\mathcal{FS}_{|X|}$ *with* $N_X.support := S_X, N_X.itemset := X$, $N_X.deltafree :=$ **false** ;
6: **od**

7: **let** $i := 1$;
8: **while** $\mathcal{FS}_i \neq \emptyset$ **do**
9: **let** $\mathcal{C}_{i+1} :=$ **APRIORI** $-$ **Gen**$(\mathcal{FS}_i)$;
10: **for all** $X \in \mathcal{C}_{i+1}$ **do**
11: *Find node* N_Y *in* $\mathcal{FS}_i$ **such that** $N_Y.itemset \subset X$ **and** $N_Y.deltafree =$ **false** ;
12: **if** *such* N_Y *exists* **then**
13: **let** $S := \min_{N \in \mathcal{FS}_i}(\{N.support : N.itemset \subset X\})$;
14: *Create new node* N_X *in* $\mathcal{FS}_{|X|}$ *with* $N_X.support := S$, $N_X.itemset := X$, $N_X.deltafree :=$ **false** ;
15: **fi**
16: **od**
17: **let** $i := i + 1$;
18: **od** ;
19: **output** $\{<N.itemset, N.support> : N \in \bigcup_{j<i} \mathcal{FS}_j\}$;

4.2 Deriving Association Rules from SRBC

A popular application of the extraction of frequent sets is the discovery of association rules. The notion of association rules has been recalled in Section 2.1.1. Computing all association rules from the proposed condensed representations is straightforward by regenerating all frequent sets and then using Algorithm 1, unless we make use of lossy representations.
In this section, we study the impact of approximation of frequent set supports on association rule supports and confidences when these rules are derived from the SRBC instead of frequent sets. We give bounds for the error made on support and confidence of association rules.

The error on support of association rules is the same as the error on support of itemsets. For a frequent rule $X \Rightarrow Y$, if we use $SRBC(r, \sigma, \delta)$ to determine its support, by Theorem 4 we always have an overestimate of its support with an error of at most $\delta|X \cup Y|$. In practice, we have the same approximation errors as those presented in Section 4.1.2.

The error bounds for confidence is less trivial to obtain from the previous results. Let $X \Rightarrow Y$ be a frequent rule in r. Suppose we use the SRBC to approximate $Sup(r, X \cup Y)$ and $Sup(r, X)$. These approximations are denoted respectively by $\overline{Sup}(r, X \cup Y)$ and $\overline{Sup}(r, X)$. Now, we can approximate $Conf(r, X \Rightarrow Y)$ by $\overline{Conf}(r, X \Rightarrow Y) = \overline{Sup}(r, X \cup Y)/\overline{Sup}(r, X)$. By Theorem 4, and since we have overestimated the supports, $Sup(r, X \cup Y)/(Sup(r, X) + \delta|X|) \leq \overline{Conf}(r, X \Rightarrow Y) \leq (Sup(r, X \cup Y) + \delta|X \cup Y|)/Sup(r, X)$.
Thus a bound for the absolute error made on the confidence when we use $\overline{Conf}(r, X \Rightarrow Y)$ instead of $Conf(r, X \Rightarrow Y)$ is $max(Sup(r, X \cup Y)/(Sup(r, X) + \delta|X|) - Conf(r, X \Rightarrow Y), (Sup(r, X \cup Y) + \delta|X \cup Y|)/Sup(r, X) - Conf(r, X \Rightarrow Y))$.
Now, we derive values of this bound in practice, using the experiments reported in Section 4.1.2. We consider the **PUMS** data set of Kansas state, which is less favorable than the other data set (**PUMSB***) since the error on the support was larger.
Let $ar(s, c)$ be the set of all association rules in this data set with support s and confidence c. For a given pair $<s, c>$, we bound the error made on confidence for all rules in $ar(s, c)$ as follows. The support of the left hand side of any of these rules is $s' = s/c$. Using the experimental results of Section 4.1.2, we can find the maximal observed relative support error made on s and s', denoted respectively by rse and rse'. Then we bound the absolute error made on the confidence by $max(s/(s' + s' * rse') - c, (s + s * rse)/s' - c)$.
We consider support $s \in \{0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ and confidence $c \in \{0.99, 0.95, 0.9, 0.85\}$. For each pair $<s, c>$, we used the maximal observed relative error on support (given in Table 4.4) to bound the error made on confidence for the set of rules $ar(s, c)$. The corresponding values are presented in Table 4.5. For example, if we consider rules with confidence 0.99 and support 0.05, the maximal absolute error made on confidence is 0.0308. For higher rule supports the error decreases. This variation cor-

	support						
confidence	0.05	0.1	0.2	0.3	0.4	0.5	0.6
0.99	0.0308	0.0116	0.0047	0.0033	0.0012	0.0010	0.0010
0.95	0.0295	0.0111	0.0045	0.0031	0.0011	0.0010	0.0010
0.9	0.0280	0.0105	0.0042	0.0030	0.0011	0.0009	0.0009
0.85	0.0264	0.0099	0.0040	0.0028	0.0010	0.0008	0.0009

Table 4.5: Bounds for absolute error on rule confidence.

responds to the reduction of the maximal observed relative error for higher supports (see Table 4.4). For lower confidence values the error decreases, too. This is due to the fact that a lower confidence implies a higher support for the left hand side of the rule and thus a lower error on the left hand side support.

4.3 Regeneration of Frequent Closed Sets from Lossless Representations

The DBC is a condensed representation of frequent sets. The experiments presented in Sections 3.3.5 and 3.3.7 have shown that the DBC is much smaller than the collection of frequent sets and that it can be extracted efficiently, even in difficult cases. Thus, in practice, the DBC can be advantageously extracted and stored instead of frequent sets to derive important data mining patterns (most commonly association rules).
We show now that the DBC is also an interesting condensed representation of the frequent closed sets and thus also offers the same practical benefits as the closed sets themselves. Among them we can mention the ability to generate directly rule covers such as of the class described in [Bastide 00a] (see Section 4.4 for details) in order to present to the expert a compact summary of (typically) huge collections of associations rules.
In Section 3.3.5, we have already described the interest of the DBC w.r.t. closed sets in terms of representation size. In this section, we give an algorithm to convert the DBC into the collection of frequent closed sets. Then, we present experiments showing that in the most difficult cases it is more efficient to extract the DBC and to convert it into closed sets than to extract the closed sets directly.
See Section 2.2.6 for basic properties concerning closed sets. In Section 4.3.1, we give some additional preliminaries.

4.3.1 Concepts Used in the Conversion Algorithm

In this section, we give some preliminary properties needed in Section 4.3.2 to present the conversion algorithm. In particular, we reuse the concept of *0-free set*, introduced in the context of strong-rule-bordered condensation in Section 3.2.

The following lemmas are needed in Section 4.3.2 to demonstrate the correctness of the conversion algorithm.

Lemma 17. *Let r be a binary database over R, X be an itemset such that $X \subseteq R$, A be a single positive item in $R \setminus X$ and $\overline{A}$ its corresponding negative item.*
Then the following three propositions are equivalent:
(1) $Sup(r, X \cup \{A\}) = Sup(r, X)$,
(2) $X \Rightarrow \{A\}$ *is a 0-strong rule in* r,
(3) $\mathcal{M}(r, X \cup \{\overline{A}\}) = \emptyset$.

Proof. Observe that $\mathcal{M}(r, X) = \{t : t \in \mathcal{M}(r, X \cup \{A\}) \vee t \in \mathcal{M}(r, X \cup \{\overline{A}\})\}$ and that the multisets $\mathcal{M}(r, X \cup \{A\})$ and $\mathcal{M}(r, X \cup \{\overline{A}\})$ are mutually exclusive. Therefore $Sup(r, X) = Sup(r, X \cup \{A\}) + Sup(r, X \cup \{\overline{A}\})$. Consequently, *(1)* is equivalent to $Sup(r, X \cup \{\overline{A}\}) = 0$, and the latter to *(3)*. Finally, $Sup(r, X) - Sup(r, X \cup \{A\}) \leq 0$, which satisfies the definition of $X \Rightarrow \{A\}$ being a 0-strong rule in r (see Definition 24), i.e., *(2)*. □

Lemma 18. *Given r a binary database over R, X is a 0-free set w.r.t. r iff* $\bigwedge_{Y \subset X} Sup(r, Y) > Sup(r, X)$.

Proof. This lemma is a special case of Lemma 8, proved earlier. □

The computation of closures of frequent 0-free sets in the conversion algorithm will be based on the following theorem.

Theorem 13. *Let r be a binary database over a set of items R, σ be a support threshold and Z be a σ-frequent 0-free set. Then, $\mathcal{I} \circ \mathcal{M}(r, Z) = Z_1 \cup Z_2 \cup Z_3 \cup Z_4$, where $Z_1, \ldots, Z_4$ are:*

$$Z_1 = \{A \in R : Z \cup \{A\} \in FreqDeltaFree\mathcal{B}d^-(r, \sigma, 0) \wedge Sup(r, Z) = Sup(r, Z \cup \{A\})\}$$

$$Z_2 = \{A \in R : \bigvee_{X \in FreqDeltaFree(r, \sigma, 0)} |X| < |Z| \wedge Sup(r, X) = Sup(r, Z) \wedge Z \cup \{A\} \subseteq \mathcal{I} \circ \mathcal{M}(r, X)\}$$

$$Z_3 = \{A \in R : A \in Z \vee \bigvee_{\substack{X \subset Z \\ X \in FreqDeltaFree(r, \sigma, 0)}} A \in \mathcal{I} \circ \mathcal{M}(r, X))\}$$

$$Z_4 = \{A \in R : A \notin Z \wedge \bigvee_{B \in Z} (Z \cup \{A\}) \setminus \{B\} \in FreqDeltaFree\mathcal{B}d^-(r, \sigma, 0) \wedge Sup(r, (Z \cup \{A\}) \setminus \{B\}) = Sup(r, Z) \wedge \bigvee_{X \subset (Z \cup \{A\}) \setminus \{B\}} B \in \mathcal{I} \circ \mathcal{M}(r, X)\}$$

This theorem states that an item A is an element of the closure of a σ-frequent 0-free set Z if and only if it satisfies at least one of the following properties:

- A is such that the rule $Z \Rightarrow \{A\}$ is a 0-strong rule,
- $Z \cup \{A\}$ is included in the closure of a frequent 0-free set X having the same support as Z but a size strictly smaller than the size of Z,

- A is in Z or in the closure of a proper subset of Z,
- A is not in Z, but is in a 0-free set W such that firstly Z and W have the same size and support, secondly they contain the same items except one, and finally $Z \setminus W$ is included in the closure of a proper subset of W.

The proof of this theorem requires the following two lemmas.

Lemma 19. *Let r be a binary database over a set of items R, σ be a support threshold, Z be a frequent 0-free set and $A \in \mathcal{I} \circ \mathcal{M}(r, Z)$. If $A \notin Z_1$ then* $\bigvee_{Y \subset Z \cup \{A\}} \bigvee_{B \in Y} Y \setminus \{B\} \in FreqDeltaFree(r, \sigma, 0) \wedge B \in \mathcal{I} \circ \mathcal{M}(r, Y \setminus \{B\})$.

Proof. Let $A \in \mathcal{I} \circ \mathcal{M}(r, Z) \setminus Z_1$. $A \notin Z_1$ implies that $Z \cup \{A\} \notin FreqDeltaFree\mathcal{B}d^-(r, \sigma, 0) \vee Sup(r, Z) \neq Sup(r, Z \cup \{A\})$. But $A \in \mathcal{I} \circ \mathcal{M}(r, Z)$ implies that $Sup(r, Z) = Sup(r, Z \cup \{A\})$ (Lemma 4). Thus, the disjunct $Z \cup \{A\} \notin FreqDeltaFree\mathcal{B}d^-(r, \sigma, 0)$ must be true.
Because $Z \Rightarrow \{A\}$ is a 0-strong rule, $Z \cup \{A\}$ is a frequent set, but not a 0-free set.
There must exist $Y \subset Z \cup \{A\}$ such that $Y \in \mathcal{B}d^-(FreqDeltaFree(r, \sigma, 0))$. Y is σ-frequent (Lemma 1), and thus $Y \notin DeltaFree(r, \sigma, 0)$. Therefore, there must exist $B \in Y$ such that $X \Rightarrow \{B\}$ is a 0-strong rule and that $X \subseteq Y \setminus \{B\}$. Since $Y \in FreqDeltaFree\mathcal{B}d^-(r, \sigma, 0)$, all its subsets are 0-free and thus X must be equal to $Y \setminus \{B\}$ (otherwise, $X \cup \{B\} \subset Y$ would not be a 0-free set).
Finally, the fact that $Y \setminus \{B\} \Rightarrow \{B\}$ is a 0-strong rule implies that $B \in \mathcal{I} \circ \mathcal{M}(r, Y \setminus \{B\})$ (Lemmas 17 and 5). □

Lemma 20. *Let r be a binary database over a set of items R, σ be a support threshold, Z be a frequent 0-free set and $A \in \mathcal{I} \circ \mathcal{M}(r, Z)$. If $A \notin Z_2$ then* $\bigwedge_{X \subset Z \cup \{A\}} |X| < |Z| \Rightarrow Sup(r, X) > Sup(r, Z)$.

Proof. We prove the contrapositive.
Suppose that $A \in \mathcal{I} \circ \mathcal{M}(r, Z)$ and $\bigvee_{X \subset Z \cup \{A\}} |X| < |Z| \wedge Sup(r, X) \leq Sup(r, Z)$. Let X be an itemset as in the above expression. If there are more than one such itemsets, let X be any minimal of them (minimal w.r.t. set inclusion).
$A \in \mathcal{I} \circ \mathcal{M}(r, Z)$ implies $Sup(r, Z) = Sup(r, Z \cup \{A\})$ (Lemma 4) and further that $Sup(r, X) \leq Sup(r, Z \cup \{A\})$. By Lemma 1, $Sup(r, X) \geq Sup(r, Z \cup \{A\})$, thus $Sup(r, X) = Sup(r, Z) = Sup(r, Z \cup \{A\})$. By Lemma 5, $X \subset Z \cup \{A\} \wedge Sup(r, X) = Sup(r, Z \cup \{A\})$ implies that $Z \cup \{A\} \subseteq \mathcal{I} \circ \mathcal{M}(r, X)$.
By definition of X, none $X' \subset X$ satisfies $|X'| < |Z| \wedge Sup(r, X') \leq Sup(r, Z)$. Since every X', subset of X, has necessarily a smaller size than X and thus smaller than Z, the property must not hold due to the second conjunct, i.e., $Sup(r, X') > Sup(r, Z)$. Here $Sup(r, Z)$ is equal to $Sup(r, X)$. Since the inequality holds for every proper subset X' of X, from Lemma 18, we know that X is 0-free.

We have $X \subset Z \cup \{A\} \wedge |X| < |Z| \wedge X \in FreqDeltaFree(r, \sigma, 0) \wedge Sup(r, X) = Sup(r, Z) \wedge Z \cup \{A\} \subseteq \mathcal{I} \circ \mathcal{M}(r, X)$, i.e., $A \in Z_2$. □

Using these two lemmas, we finally can prove Theorem 13.

Proof. [of Theorem 13] We prove that every $A \in \mathcal{I} \circ \mathcal{M}(r, Z)$ that does not belong either to Z_1 or to Z_2, belongs to Z_3 or to Z_4.

Let $A \in \mathcal{I} \circ \mathcal{M}(r, Z) \setminus (Z_1 \cup Z_2)$.

By Lemma 19, $A \notin Z_1$ implies that $\bigvee_{Y \subset Z \cup \{A\}} \bigvee_{B \in Y} Y \setminus \{B\} \in FreqDeltaFree(r, \sigma, 0) \wedge B \in \mathcal{I} \circ \mathcal{M}(r, Y \setminus \{B\})$. Let Y be such an itemset and B be such a member of Y.

By Lemma 20, $A \notin Z_2$ implies that $\bigwedge_{X \subset Z \cup \{A\}} |X| < |Z| \Rightarrow Sup(r, X) > Sup(r, Z)$, in particular $\bigwedge_{X \subset (Z \cup \{A\}) \setminus \{B\}} Sup(r, X) > Sup(r, Z)$.

Three cases are possible.

The first is when $A \neq B \wedge A \notin Z$. Then $B \in \mathcal{I} \circ \mathcal{M}(r, Y \setminus \{B\})$ implies (by Lemma 4) that $Sup(r, Y \setminus \{B\}) = Sup(r, Y)$ and then (by Lemma 17) that $\mathcal{M}(r, (Y \setminus \{B\}) \cup \{\overline{B}\}) = \emptyset$. Since $Y \setminus \{B\} \subseteq (Z \cup \{A\}) \setminus \{B\}$, Lemma 2 implies that $\mathcal{M}(r, ((Z \cup \{A\}) \setminus \{B\}) \cup \{\overline{B}\}) \subseteq \mathcal{M}(r, (Y \setminus \{B\}) \cup \{\overline{B}\})$ $(= \emptyset)$. Therefore, no rows belong to the multiset $\mathcal{M}(r, ((Z \cup \{A\}) \setminus \{B\}) \cup \{\overline{B}\})$. Thus, by Lemma 17, $Sup(r, (Z \cup \{A\}) \setminus \{B\}) = Sup(r, Z \cup \{A\}) = Sup(r, Z)$.

By Lemma 18, $\bigwedge_{X \subset (Z \cup \{A\}) \setminus \{B\}} Sup(r, X) > Sup(r, Z) = Sup(r, (Z \cup \{A\}) \setminus \{B\})$ implies that $(Z \cup \{A\}) \setminus \{B\}$ is a 0-free set. Therefore, A belongs to Z_4.

The second case is $A = B \wedge A \notin Z$. In this case, $B \in \mathcal{I} \circ \mathcal{M}(r, Y \setminus \{B\})$ is equivalent to $A \in \mathcal{I} \circ \mathcal{M}(r, Y \setminus \{A\})$, where $(Y \setminus \{A\}) \subset (Z \cup \{A\}) \setminus \{A\}$, i.e., $Y \setminus \{A\} \subset Z$. This case turns out to be a case, where $A \in Z_3$.

The last case is when $A \in Z$. Also here, by definition of Z_3, $A \in Z_3$. □

4.3.2 Condensed Representation Conversion Algorithm

In this section, we describe the algorithm **Regen-FreqCl** that changes the DBC into the condensed representation based on frequent closed sets. Although a more abstract presentation is possible, we give a detailed version to ease the implementation of an efficient converter.

As previously, using only $<X, S_X>$ pairs from $DBC(r, \sigma)$ verifying $S_X\sigma' > \sigma$ may be used to eliminate unnecessary computations and obtain directly $FreqClosed(r, \sigma')$.

Regen-FreqCl computes the set of all frequent 0-free sets and their corresponding closures. These closures are computed in 5 steps. The first is the initialization of the closure of Z to the value of Z, followed by computing the parts $Z_1, \ldots, Z_4$ of the closure following Theorem 13.

Even though we give **Regen-FreqCl** in a detailed form, the actual implementation integrates additional straightforward optimizations. Most important hints will be mentioned throughout this section.

The data structure corresponding to one itemset is called a node. A collection of such nodes is stored in an itemset-prefix-tree, as in Section 4.1.1, to allow an efficient access. The structure of node is the following:

$itemset$	: set of items
$support$	: integer
$prunedby$	: set of items
$closure$	: array 1..4 of set of items

For a node N corresponding to an itemset Z, $N.itemset$ is the set of items in Z and $N.support$ is the support of Z.

$N.prunedby$ is a set of at most two items. If it contains two different items A, B it corresponds to a set Z that is 0-free but not disjunction-free and means that $A, B \in Z$ and $Z \setminus \{A, B\} \Rightarrow A \vee B$ is valid. If $N.prunedby$ contains a single item A it means that Z is not 0-free and that $Z \setminus \{A\} \Rightarrow \{A\}$ is a 0-strong rule. And when $N.prunedby = \emptyset$, the node corresponds to an itemset Z that is disjunction-free (and thus also 0-free).

The frequent sets are grouped in different collections according to their size and to the fact whether they are 0-free sets or not (in this latter case they are frequent sets of the negative border of the 0-free sets, defined in Section 3.2.2 as Definition 28).

For the frequent sets of size i, we call $\mathcal{ZF}reeFr_i$ the collection of the 0-free sets and $\mathcal{F}req\mathcal{N}eg\mathcal{ZB}d_i$ the collection of the others. All these collections are stored in separate itemset-prefix-trees.

The four elements of the array $closure$ denoted $closure_1, \ldots, closure_4$ correspond respectively to the sets $Z_1, \ldots, Z_4$ specified in Theorem 13. The four procedures depicted as Algorithms 17–20 are used by the conversion algorithm to compute these four elements accordingly to the definitions of $Z_1, \ldots, Z_4$.

The decomposition of the closure of a 0-free set into an array of 4 elements $closure_1, \ldots, closure_4$ helps in understanding how the Theorem 13 is applied. Observe however that the only position where we need to access items from one specific element of that array is the line 2 of the algorithm UPDATE_CLOSURE_4.

We implemented the storage of closures without any use of array — we add items to the collection corresponding to the combined closure and when we add items that would belong to $closure_3$ but not to $itemset$ (the items enumerated in line 2 of UPDATE_CLOSURE_4), we run the lines 3–8 of UPDATE_CLOSURE_4.

The only position, where we add items that would belong to $closure_3$ but not to $itemset$ is line 4 of UPDATE_CLOSURE_3.

Due to this optimisation, the resulting procedures must not merge the items each time a complete closure is needed, unlike suggested in algorithms UPDATE_CLOSURE_2 (lines 2

and 3), UPDATE_CLOSURE_3 (line 4) and in the forthcoming algorithm **Regen-FreqCl** (line 39).
The conversion algorithm uses the procedure **FindDisjRule** described in Section 3.3.6 to compute *prunedby* for itemsets that are 0-free but not disjunction-free and also the procedure **FindStrongRule** to determine *prunedby* for non-0-free sets. **FindStrongRule** (shown as Algorithm 21) proposes how to determine $N.prunedby$ of a non-0-free set $N.itemset$ when the nodes corresponding to all its subsets of size $|N.itemset| - 1$ are available.

Now, we give the conversion algorithm itself. Its pseudo-code is depicted as Algorithm 22.

In lines 1–11, Algorithm **Regen-FreqCl** loads the input and converts it to a collection of nodes, one node per itemset. For an itemset X, if the call to **FindStrongRule** gives a nonempty result (in line 5), then X is not 0-free, but it is an element of the set $FreqDeltaFree\mathcal{B}d^-(r, \sigma, 0)$ (frequent sets of the negative border of frequent 0-free sets). Thus, it is stored in $\mathcal{F}req\mathcal{N}eg\mathcal{Z}\mathcal{B}d_{|X|}$ and has a field *prunedby* filled with a single item. The nodes corresponding to frequent 0-free sets of $FreqDisjFree\mathcal{B}d^-(r, \sigma)$ (the negative border of frequent disjunction-free sets) are marked distinct when loaded in line 7 by the content of the field *prunedby*, which contains two items.
Reading the input is followed by the computation of the closure of $\emptyset$ in lines 12 and 13, according to Theorem 13 applied to this particular itemset. All other frequent closed sets are obtained by computing the closures of nonempty frequent 0-free sets. This leads to two interleaved processes. The first is the creation of nodes corresponding to frequent 0-free sets or to frequent sets from the negative border of frequent 0-free sets. And the

Algorithm 17. (UPDATE_CLOSURE_1)

Input: *$\mathcal{Z}\mathcal{F}reeFr_i$ (fields itemset and support) nodes corresponding to all itemsets of size i from $FreqDeltaFree(r, \sigma, 0)$. $\mathcal{F}req\mathcal{N}eg\mathcal{Z}\mathcal{B}d_{i+1}$ (fields itemset and support) nodes corresponding to all frequent sets of size $i + 1$ from $FreqDeltaFree\mathcal{B}d^-(r, \sigma, 0)$.*
Prerequisite: *$closure_1$ initialized in $\mathcal{Z}\mathcal{F}reeFr_i$ to $\emptyset$.*
Output: *Updated $closure_1$ of all nodes from $\mathcal{Z}\mathcal{F}reeFr_i$.*

```
1: for all N ∈ FreqNegZBd_{i+1} do
2:    for all A ∈ N.itemset do
3:       let N_A be the node in ZFreeFr_i such that
           N_A.itemset = N.itemset \ {A};
4:       if N_A.support = N.support then
5:          let N_A.closure_1 := N_A.closure_1 ∪ {A};
6:       fi
7:    od
8: od
```

Algorithm 18. (UPDATE_CLOSURE_2)

Input: *$\mathcal{ZF}reeFr_i$ (fields itemset and support) nodes corresponding to all itemsets of size i from $FreqDeltaFree(r, \sigma, 0)$. $\bigcup_{j<i} \mathcal{ZF}reeFr_j$ (all fields) nodes corresponding to all itemsets of sizes strictly lower than i from $FreqDeltaFree(r, \sigma, 0)$.*
Prerequisite: *$closure_2$ initialized in $\mathcal{ZF}reeFr_i$ to $\emptyset$.*
Output: *Updated $closure_2$ of all nodes from $\mathcal{ZF}reeFr_i$.*

1: **for all** $N_C \in \bigcup_{j<i} \mathcal{ZF}reeFr_j$ **do**
2: **for all** $N \in \mathcal{ZF}reeFr_i$ **such that** $N.itemset \subset \bigcup_{k=1..4} N_C.closure_k$
3: **and** $N_C.support = N.support$ **do**
let $N.closure_2 := N.closure_2 \cup \bigcup_{k=1..4} N_C.closure_k$;
4: **od**
5: **od**

Algorithm 19. (UPDATE_CLOSURE_3)

Input: *$\mathcal{ZF}reeFr_i$ (fields itemset and support) nodes corresponding to all itemsets of size i from $FreqDeltaFree(r, \sigma, 0)$. $\mathcal{ZF}reeFr_{i-1}$ (all fields) nodes corresponding to all itemsets of size $i-1$ from $FreqDeltaFree(r, \sigma, 0)$.*
Prerequisite: *$closure_3$ of each node N in $\mathcal{ZF}reeFr_i$ initialized to $N.itemset$.*
Output: *Updated $closure_3$ of all nodes from $\mathcal{ZF}reeFr_i$.*

1: **for all** $N \in \mathcal{ZF}reeFr_i$ **do**
2: **for all** $A \in N.itemset$ **do**
3: **let** N_A *be the node in* $\mathcal{ZF}reeFr_{i-1}$ **such that**
$N_A.itemset = N.itemset \setminus \{A\}$;
4: **let** $N.closure_3 := N.closure_3 \cup \bigcup_{k=1..4} N_A.closure_k$;
5: **od**
6: **od**

Algorithm 20. (UPDATE_CLOSURE_4)

Input: *$\mathcal{Z}\mathcal{F}reeFr_i$ (fields itemset, $closure_3$ and support) nodes corresponding to all itemsets of size i from $FreqDeltaFree(r, \sigma, 0)$.*
Prerequisite: *$closure_4$ initialized in $\mathcal{Z}\mathcal{F}reeFr_i$ to $\emptyset$.*
Output: *Updated $closure_4$ of all nodes from $\mathcal{Z}\mathcal{F}reeFr_i$.*

1: **for all** $N_W \in \mathcal{Z}\mathcal{F}reeFr_i$ **do**
2: **for all** $B \in N_W.closure_3 \setminus N_W.itemset$ **do**
3: **for all** $A \in N_W.itemset$ **do** // *here* $A \neq B$
4: **let** N_Z *be the node in* $\mathcal{Z}\mathcal{F}reeFr_i$ **such that**
$N_Z.itemset = (N_W.itemset \cup \{B\}) \setminus \{A\}$;
5: **if** $N_Z.support = N_W.support$ **then**
6: **let** $N_Z.closure_4 := N_Z.closure_4 \cup \{A\}$;
7: **fi**
8: **od**
9: **od**
10: **od**

Algorithm 21. (FindStrongRule)

Input: *Itemset Z, S support of Z, collection of nodes $\mathcal{C}$ including nodes corresponding to all subsets of Z of size $|Z| - 1$.*
Output: *Itemset P containing 1 item corresponding to the right-hand-side of a 0-strong rule $Z \setminus \{A\} \Rightarrow \{A\}$ if it exists, $\emptyset$ otherwise.*

1: *Find any $A \in Z$ and N_A a node in $\mathcal{C}$* **such that**
$N_A.itemset = Z \setminus \{A\}$ **and** $S = N_A.support$;
2: **if** *such A exists* **then** // *in this case $Z \setminus \{A\} \Rightarrow \{A\}$ is a 0-strong rule*
3: **let** $P := \{A\}$;
4: **else**
5: **let** $P := \emptyset$;
6: **fi**
7: **output** P;

Algorithm 22. (Regen-FreqCl)

Input: σ, $DBC(r, \sigma)$ *of the form* $<\mathcal{F}, \mathcal{B}>$.
Prerequisite: $\mathcal{F} \cup \mathcal{B}$ *not empty.*
Output: $FreqClosed(r, \sigma)$ *and their supports.*

1: **for all** $<X, S_X> \in \mathcal{F}$ **do** *// Disjunction-free input sets*
2: *Create new node* N_X *in* $\mathcal{Z}\mathcal{F}reeFr_{|X|}$ *with* $N_X.support := S_X$, $N_X.itemset := X$, $N.closure := [\emptyset, \emptyset, X, \emptyset]$, $N_X.prunedby := \emptyset$;
3: **od**
4: **for all** $<X, S_X> \in \mathcal{B}$ **do** *// Non-disjunction-free input sets*
5: **let** $P :=$ **FindStrongRule**$(X, S_X, \mathcal{Z}\mathcal{F}reeFr_{|X|-1})$;
6: **if** $P = \emptyset$ **then**
7: *Create new node* N_X *in* $\mathcal{Z}\mathcal{F}reeFr_{|X|}$ *with* $N_X.support := S_X$, $N_X.itemset := X$, $N_X.closure := [\emptyset, \emptyset, X, \emptyset]$, $N_X.prunedby :=$ **FindDisjRule**$(X, S_X, \mathcal{Z}\mathcal{F}reeFr_{|X|-1} \cup \mathcal{Z}\mathcal{F}reeFr_{|X|-2})$;
8: **else**
9: *Create new node* N_X *in* $\mathcal{F}req\mathcal{N}eg\mathcal{Z}\mathcal{B}d_{|X|}$ *with* $N_X.support := S_X$, $N_X.itemset := X$, $N_X.closure := [\emptyset, \emptyset, \emptyset, \emptyset]$, $N_X.prunedby := P$;
10: **fi**
11: **od**

12: **let** N_E *be the node in* $\mathcal{Z}\mathcal{F}reeFr_0$ **such that** $N_E.itemset = \emptyset$;
13: **let** $N_E.closure_1 := \{N_Y.itemset : N_Y \in \mathcal{F}req\mathcal{N}eg\mathcal{Z}\mathcal{B}d_1\}$;
14: **let** $i := 1$;
15: **while** $\mathcal{Z}\mathcal{F}reeFr_{i-1} \neq \emptyset$ **do**
16: **let** $\mathcal{C}_{i+1} :=$ **APRIORI** $-$ **Gen**$(\mathcal{Z}\mathcal{F}reeFr_i)$;
17: **for all** $X \in \mathcal{C}_{i+1}$ **do**
18: *Find node* N_Y *in* $\mathcal{Z}\mathcal{F}reeFr_i$ **such that** $N_Y.itemset \subset X$ **and** $|N_Y.prunedby| = 2$;
19: **if** *such* N_Y *exists* **then**
20: **let** A, B *be items* **such that** $N_Y.prunedby = \{A, B\}$;
21: **let** N_A, N_B *be nodes from* $\mathcal{Z}\mathcal{F}reeFr_i$ *and* N_{AB} *be a node from* $\mathcal{Z}\mathcal{F}reeFr_{i-1}$ **such that** $N_A.itemset = X \setminus \{A\}$ **and** $N_B.itemset = X \setminus \{B\}$ **and** $N_{AB}.itemset = X \setminus \{A, B\}$;
22: **let** $S := N_A.support + N_B.support - N_{AB}.support$;
23: **if** $S \geq \sigma$ **then**
24: **let** $P :=$ **FindStrongRule**$(X, S, \mathcal{Z}\mathcal{F}reeFr_i)$;
25: **if** $P = \emptyset$ **then**
26: *Create new node* N_X *in* $\mathcal{Z}\mathcal{F}reeFr_{|X|}$ *with* $N_X.support := S$, $N_X.itemset := X$, $N_X.closure := [\emptyset, \emptyset, X, \emptyset]$, $N_X.prunedby := \{A, B\}$;
27: **else**
28: *Create new node* N_X *in* $\mathcal{F}req\mathcal{N}eg\mathcal{Z}\mathcal{B}d_{|X|}$ *with* $N_X.support := S$, $N_X.itemset := X$, $N_X.closure := [\emptyset, \emptyset, \emptyset, \emptyset]$, $N_X.prunedby := P$;
29: **fi**
30: **fi**
31: **fi**
32: **od**
33: UPDATE_CLOSURE_1 $(\mathcal{Z}\mathcal{F}reeFr_i, \mathcal{F}req\mathcal{N}eg\mathcal{Z}\mathcal{B}d_{i+1})$;
34: UPDATE_CLOSURE_2 $(\mathcal{Z}\mathcal{F}reeFr_i, \bigcup_{j<i} \mathcal{Z}\mathcal{F}reeFr_j)$;
35: UPDATE_CLOSURE_3 $(\mathcal{Z}\mathcal{F}reeFr_i, \mathcal{Z}\mathcal{F}reeFr_{i-1})$;
36: UPDATE_CLOSURE_4 $(\mathcal{Z}\mathcal{F}reeFr_i)$;
37: **let** $i := i + 1$;
38: **od**
39: **output** $\{<\bigcup_{k=1..4} N.closure_k, N.support> : N \in \bigcup_{j<i} \mathcal{Z}\mathcal{F}reeFr_j\}$;

second is the construction of the frequent closed sets by computing the closures of frequent 0-free sets.

These two processes are performed within the loop starting at line 15. The i^{th} iteration first generates the itemsets of size $i+1$ that are either frequent 0-free sets (stored in $\mathcal{ZF}reeFr_{i+1}$, line 26) or frequent sets from the negative border of frequent 0-free sets (stored in $\mathcal{F}req\mathcal{N}eg\mathcal{ZB}d_{i+1}$, line 28). Then it determines the closures of the 0-free sets of size i using the characterization of the closed sets stated in Theorem 13.

It should be noticed that during this i^{th} iteration the algorithm cannot compute the closures of the newly generated 0-free sets of size $i+1$ since the use of Theorem 13 requires in this case the frequent sets from the negative border of 0-free sets of size $i+2$ (i.e., $\mathcal{F}req\mathcal{N}eg\mathcal{ZB}d_{i+2}$).

In lines 16–19, the algorithm selects candidate sets that are potentially frequent and 0-free, but are not in the input because they are proper supersets of non-disjunction-free sets. Next, in lines 20–22, for each candidate set X, the algorithm searches the necessary information to restore the support of X. It takes Y, any non-disjunction-free proper subset of X, and considers A, B the items in the corresponding $N_Y.pruned by$. Since $Y \setminus \{A, B\} \Rightarrow A \vee B$ is a valid simple disjunctive rule then, using Lemma 13, the algorithm infers the support S of X.

If X is frequent (line 23) then the corresponding node N_X is created in lines 24–29. If the call to **FindStrongRule** on X returns an empty set then X is a 0-free set, therefore N_X is created in $\mathcal{ZF}reeFr_{i+1}$ and the value of *pruneby* for Y propagates to X. If not, the set X is not a 0-free set, but all its subsets are 0-free sets and thus X is in the negative border of 0-free sets. In this case N_X is created in $\mathcal{F}req\mathcal{N}eg\mathcal{ZB}d_{i+1}$, and $N_X.pruneby$ is filled with the result of **FindStrongRule**.

The computation of the closures of the 0-free sets of size i (stored in $\mathcal{ZF}reeFr_i$) is then performed in lines 33–36.

The algorithm exits the main loop when the closures of frequent 0-free sets of the maximal size are computed.

Theorem 14. *The algorithm* **Regen-FreqCl** *regenerates all and only frequent 0-free sets in* $\bigcup_{j<k} \mathcal{ZF}reeFr_j$*, where* k *is the value of the variable* i *at the end of the execution of* **Regen-FreqCl**.

Proof. Similar to the proof of Theorem 12. □

Lemma 21. *Given* r *a binary database over* R*,* σ *a support threshold and* X *a frequent closed set w.r.t.* r*,* $\bigvee_{\substack{Y \subseteq X \\ Y \in FreqDeltaFree(r,\sigma,0)}} X = \mathcal{I} \circ \mathcal{M}(r, Y)$.

Proof. Let r, σ and X be same as in the lemma. Let $Y \subseteq X$ be the smallest itemset w.r.t. itemset inclusion having the same support as the support of X. Below, we show that Y is σ-frequent 0-free set and that $X = \mathcal{I} \circ \mathcal{M}(r, Y)$.

By Lemma 5, $Sup(r, X) = Sup(r, Y)$ implies $X \subseteq \mathcal{I} \circ \mathcal{M}(r, Y)$. Since $X = \mathcal{I} \circ \mathcal{M}(r, X)$ (X is closed), $\mathcal{I} \circ \mathcal{M}(r, X) \subseteq \mathcal{I} \circ \mathcal{M}(r, Y)$. By Lemma 3, $\mathcal{I} \circ \mathcal{M}(r, X) \supseteq \mathcal{I} \circ \mathcal{M}(r, Y)$. Therefore, $X = \mathcal{I} \circ \mathcal{M}(r, X) = \mathcal{I} \circ \mathcal{M}(r, Y)$.
Since Y is the smallest itemset having the same support, $\bigwedge_{Z \subset Y} Sup(r, Z) \neq Sup(r, Y)$. Additionally, by the anti-monotonicity of support, we can refine that property into $\bigwedge_{Z \subset Y} Sup(r, Z) > Sup(r, Y)$, which according to Lemma 18, implies that Y is 0-free set. $Sup(r, X) = Sup(r, Y)$ and X σ-frequent implies that Y is σ-frequent. □

Theorem 15 (Correctness of Regen-FreqCl). *The algorithm* **Regen-FreqCl** *outputs all and only frequent closed sets along with their supports.*

Proof. Immediate from Theorem 14 and Lemma 21. □

Note on Converting DBC into SRBC and SRBC into Frequent Closed Sets

The SRBC with $\delta = 0$ is a lossless representation. That representation and two other exact condensed representations investigated in this thesis (the DBC and frequent closed sets) should be possible to obtain from each other.
As a matter of fact, a spin-off of the DBC-to-frequent closed sets conversion algorithm presented above is that we can without difficulty obtain the DBC-to-SRBC and SRBC-to-frequent closed sets conversion algorithms by trimming Algorithm 22. The resulting conversion times will be certainly lower than the combined conversion DBC-to-frequent closed sets. In the next section we report only the hardest of the three, the DBC-to-frequent closed sets conversion.

4.3.3 Results of Experiments

We have presented the advantage of the DBC w.r.t. frequent closed sets in terms of representation size in Section 3.3.5. In this section, we report experiments showing that the frequent closed sets can be derived very efficiently from the DBC representation using the algorithm **Regen-FreqCl**. These additional experiments lead to the conclusion that in practice the DBC turns out to be also an interesting condensed representation of the frequent closed sets.
These experiments are run in the same conditions as in Sections 3.3.5 and 3.3.7 (same platform and data sets, and same implementations of **CLOSE** and **CLOSET**). We compare the extraction times of direct computation of frequent closed sets using **CLOSE** and **CLOSET** with the extraction of the DBC (by means of **HLin-Ex** and **VLin-Ex**) followed by its conversion into frequent closed sets using an implementation of **Regen-FreqCl**. The extraction times of the DBC alone (without running **Regen-FreqCl**) have been given in Section 3.3.7. The running times (in seconds) for several support thresholds are given in Figure 4.1 (note that some axes are logarithmically scaled).

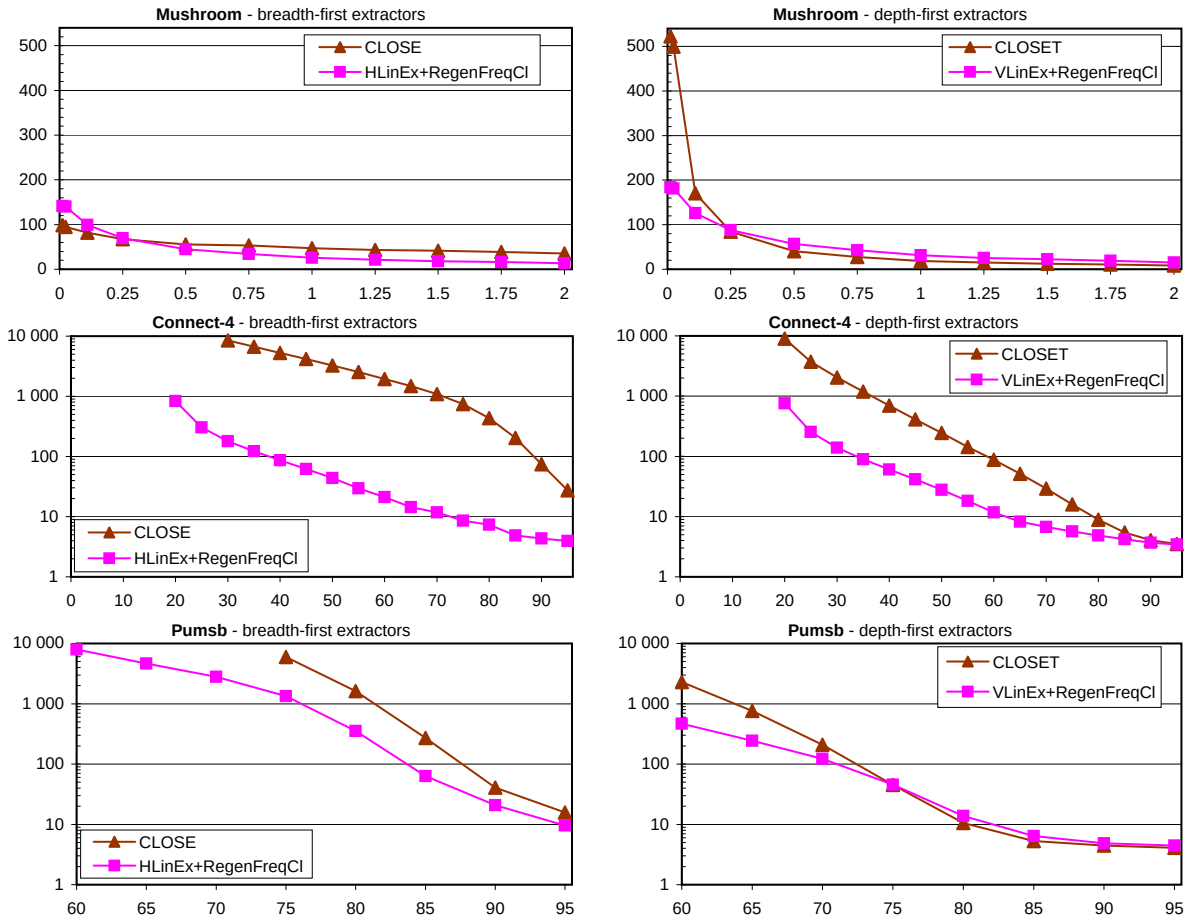

Figure 4.1: Experiments on the three data sets with breadth-first (left) and depth-first (right) algorithms, reporting the extraction time in seconds vs the relative support threshold in %.

The results on the **Mushroom** data set are the most ambiguous. All resulting times are very close, and for both strategies the curves related to the extraction of the DBC followed by **Regen-FreqCl** intersect the ones of the corresponding reference algorithms (see Figure 4.1, upper graphics). On the **Connect-4** data set, the extraction of the DBC followed by its conversion into closed sets is always faster (up to 15 times with depth-first extractors and up to 100 times with breadth-first ones). For the last data set, **PUMSB**, mining frequent closed sets was difficult for all techniques and has been stopped at a relative support threshold of 60%. In the most difficult cases (low support thresholds), the experiments show a real gain, up to fivefold speed-up, when using the DBC and its conversion versus the direct extraction of the closed sets.

In most of experiments, the extraction of the DBC followed by **Regen-FreqCl** produces the frequent closed sets much more efficiently than the direct extraction of the frequent closed sets. In the cases where the direct extraction is still faster, it should be noticed that the performances of both approaches are very close.

In the following series of experiments, we investigate the practical complexity of the frequent closed set computation using **Regen-FreqCl**. We report the average time spent by the methods to produce 1 byte of the result (using frequent closed set representation

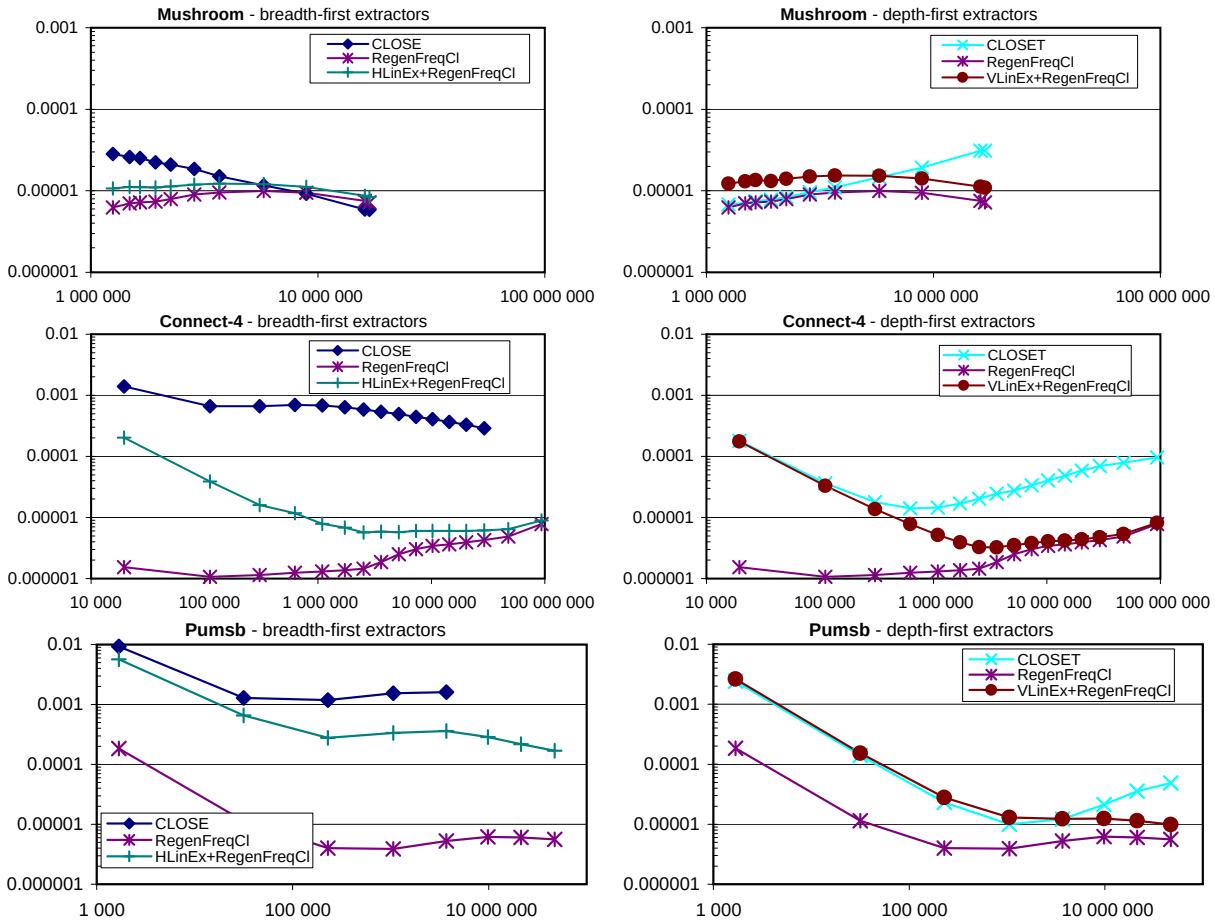

Figure 4.2: Experiments on the three data sets with breadth-first (left) and depth-first (right) algorithms, reporting the average computation time in seconds per flat-storage unit vs the flat-storage size.

flat-storage size, see Section 3.2.6). We provide results for the direct extraction (using **CLOSE** or **CLOSET**) and for extraction of the DBC followed by computation of frequent closed sets using **Regen-FreqCl**. We also measure the time per output byte for the regeneration alone, to focus on the practical complexity of the regeneration itself.

Figure 4.2 shows the total time for the computation of the frequent closed sets divided by the frequent closed sets size. The size of the representations varies over several orders of magnitude (note that all axes are logarithmically scaled). In spite of that, the running time of all investigated methods vary very little when divided by the output size, except for smaller representation sizes, below 100 000 bytes (**PUMSB** and **Connect-4**, the leftmost points of the curves), where size-independent initialisation time of programs cannot be distributed over otherwise large size of the result.

We noted that the computation time for representation sizes over 100 000 bytes vary very little for a single method, but the averages corresponding to different methods reside in different orders of magnitude. We also can observe that **CLOSET** shows a systematic growth (for all data sets) of the average time per output byte when output collection size increases.

However, the time of regeneration of frequent closed sets alone amounts to few μseconds

per output byte in all cases and does not show systematic growth of the average time. We conclude from above experiments that in practice the regeneration of the frequent closed sets from the DBC using **Regen-FreqCl** is quasi-linear w.r.t. **Regen-FreqCl** output size.

4.4 Computing Rule Covers from Condensed Representations of Frequent Sets

The idea of association rule covers is to provide an equivalent information about the data set as the full collection of rules, but in a compact, easier to interpret form.
This basic idea has been interpreted in several ways (see, e.g., [Toivonen 95, Aggarwal 98, Pasquier 99a, Bastide 00a, Zaki 00]). Some covers are not designed to infer values of support and confidence measures. Some may even not infer all association rules. But in some contexts, such covers may achieve high density of "worthy" rules.
In [Bastide 00a] and in [Zaki 00] resp. Bastide *et al.* and Zaki claim that a well-defined sub-collection of $RulesSuppConf(r, \sigma, \rho, R)$ is sufficient to derive $RulesSuppConf(r, \sigma, \rho, R)$[7]. In that sense, that subcollection is a cover of all association rules.
Below we briefly introduce only one of them, notably the cover presented in [Bastide 00a], although both covers are very similar. We adapt the vocabulary to the terms used in the thesis.

Definition 39. (Generic basis for exact association rules) Let r be a binary database over the set of items R. Let $\sigma \in (0, |r|]$ be a support threshold, $\rho \in [0, 1]$ a confidence threshold. Generic basis for exact association rules is the collection $\{X \Rightarrow Y \setminus X \ \ [Sup(r,Y), 1] : X \in FreqDeltaFree(r, \sigma, 0) \wedge X \subset Y \wedge Y = \mathcal{I} \circ \mathcal{M}(r, X)\}$.

Definition 40. (Transitive reduction of the informative basis for approximate association rules) Let r be a binary database over the set of items R. Let $\sigma \in (0, |r|]$ be a support threshold, $\rho \in [0, 1]$ a confidence threshold. Transitive reduction of the informative basis for approximate association rules is the collection $\{X \Rightarrow Y \setminus X \ \ [Sup(r,Y), Conf(r, X \Rightarrow Y \setminus X)] : X \in FreqDeltaFree(r, \sigma, 0) \wedge Y \in FreqClosed(r, \sigma) \wedge Conf(r, X \Rightarrow Y \setminus X) \geq \rho \wedge \mathcal{I} \circ \mathcal{M}(r, X) \subset Y \wedge \neg\Big(\bigvee_{Z \in FreqClosed(r,\sigma)} \mathcal{I} \circ \mathcal{M}(r, X) \subset Z \subset Y \Big)\}$.

The association rule cover is composed of both above bases.

The following examples illustrate this cover on an example data set. We reuse the binary database depicted as Table 2.1 (Section 2.1.1) and subsequent examples.

[7] $RulesSuppConf(r, \sigma, \rho, R)$ is the collection of all and only σ-frequent ρ-confident association rules along with their supports and confidences, see Definition 9.

Example 12. *If we consider the baskets represented in Table 2.1 and a support threshold value of* $2/8$*, then the collection of all frequent sets for the support threshold of 2/8 has been given as Example 1. All frequent 0-free sets and their corresponding supports are:*

$$\begin{array}{cccccccc} \emptyset & [8/8], & \{A\} & [4/8], & \{A,B\} & [2/8], & \{C\} & [5/8], \\ \{A,D\} & [2/8], & \{B\} & [4/8], & \{A,C\} & [2/8], & \{D\} & [4/8]. \end{array}$$

All frequent closed sets and their corresponding supports are:

$$\begin{array}{cccccccc} \emptyset & [8/8], & \{A\} & [4/8], & \{C\} & [5/8], & \{A,D\} & [2/8], \\ \{B,C\} & [4/8], & \{D\} & [4/8], & \{A,B,C\} & [2/8]. & & \end{array}$$

Example 13. *With the same example data set, we presented the collection of all association rules for the support and confidence thresholds respectively of 2/8 and 1/2 (17 rules) as Example 2.*
The generic basis for exact association rules is the following collection of association rules and their scores:

$$\{B,A\} \Rightarrow \{C\} \quad [2/8, 2/2], \quad \{C,A\} \Rightarrow \{B\} \quad [2/8, 2/2], \quad \{B\} \Rightarrow \{C\} \quad [4/8, 4/4].$$

The transitive reduction of the informative basis for approximate association rules is the collection:

$$\begin{array}{ccccc} \emptyset \Rightarrow \{A\} & [4/8, 4/8], & \{D\} \Rightarrow \{A\} & [2/8, 2/4], & \{A\} \Rightarrow \{B,C\} \quad [2/8, 2/4], \\ \{B\} \Rightarrow \{A,C\} & [2/8, 2/4], & \{C\} \Rightarrow \{B\} & [4/8, 4/5], & \emptyset \Rightarrow \{C\} \quad [5/8, 5/8], \\ \{A\} \Rightarrow \{D\} & [2/8, 2/4], & \emptyset \Rightarrow \{D\} & [4/8, 4/8]. & \end{array}$$

Clearly, the both bases of this cover may be efficiently computed from frequent 0-free sets and their respective closures directly from the above definitions. Therefore, the frequent closed set regeneration procedure detailed in Section 4.3.2 is very useful, because it outputs for each 0-free set X the closure and the support of X. By adding X itself to the already-output pairs closure-support, we may get directly all necessary components to compute easily the cover.
If we wished to apply an algorithm directly issued from the definitions of both bases when having either frequent closed sets or frequent 0-free sets alone, we would have to first produce the other component before computing the cover. Another option would be to use the frequent closed sets mining algorithms that output all frequent 0-free sets corresponding to a frequent closed set. However, this sharply narrows the choice of the extractor, because, e.g., **CLOSET** and the formal concept lattice methods (not reported in this thesis) are not designed to output all frequent 0-free sets along with frequent closed sets.

4.5 Towards Discovery of Association Rules with Negations

In this section, we consider association rules between generalized itemsets (see Definition 11 in Section 2.1.3). Some ideas presented here have been published in [Boulicaut 00b].
We want to find associations between positive items and negative ones. In the following, we use the term *generalized rules* for such rules. Mining generalized frequent rules turns to be intractable in practical cases, i.e., for support thresholds that enable the discovery of "standard" association rules. When mining association rules, the expensive step concerns the computation of the frequent sets from which the rules are derived. Therefore, we focus on this step.
We would like to identify possibilities for a tractable approach to the computation of generalized frequent sets (see Definition 12 to recall the notion of support for a generalized itemset) and their post-processing into generalized association rules.
First, we explain why a naïve approach (i.e., the straightforward encoding of both positive and negative items and the use of standard algorithms like **APRIORI**) cannot be used. Then we consider how it is possible to derive some generalized rules using only the information about "positive" itemsets.
Then, we consider mining generalized frequent sets (from which rules with negations can be derived) in the classical framework of frequent sets considering negative items as additional items in the binary database schema. Resulting explosion of the search-space size and (above all) tremendous increase in the number of frequent sets must be somehow counterbalanced. For that, we consider three approaches:

- constraining the search-space by asking the user to guide additionally the process towards the promising pattern classes. The user might, e.g., specify which items should, or should not appear in the extracted rules,
- using the condensed representations.

Roughly speaking, first, we identify constraints that can be "pushed" efficiently into the discovery algorithm. We consider relevant constraints for mining association rules with negations.
Then, we suggest making use of the condensed representations to alleviate heavy frequent set mining step. Finally, we propose a combination of both.

Notations We make use of notations introduced in Section 2.1.3. Additionally, we will make use of the *complemented* binary database r' that has been constructed by materializing negative items. Assuming the original binary database r over R, it means that for each item $A \in R$, one adds an item A' to the database schema (the resulting schema is

denoted R'). Respecting the semantics of negative items that have been given through Definition 12, for a given row, the value of A' is the Boolean negation of the value for A. Conforming to that, the results of $\mathcal{M}$ and Sup functions remain the same for itemsets based on materialized negative items or on non-materialized ones (see Definition 11). Nonetheless, unlike $\overline{A}$, the item A' is materialized, so we keep the notations different for formal reasons.

4.5.1 Problems with Negations... and Potential Solutions

A Naïve Tactic

Let us consider the use of **APRIORI** or other classical frequent set mining algorithms on the complemented binary database r' over R'.
The problem with this tactic is that for reasonable support thresholds, the number of frequent sets is intractable. For instance, assume you have 100 positive items with a maximum single-item relative support of 0.1 and a support threshold $\gamma = 0.05 * |r'|$. It turns out that every set of negative items up to size 9 is frequent. In that case, it leads to more than $\binom{100}{9} > 10^{12}$ frequent sets [Mannila 96b]. In fact, this encoding also introduces a high correlation in the data set: many association rules with high confidence hold in it. In practice, it means that we have to take higher support thresholds, possibly leading to uninteresting mining phases. Furthermore, even if the extraction of the frequent sets remains tractable, most of them will involve only negative items and, most of the derived rules (with high confidence) will concern only negative items as well. This is unfortunate for many application domains. Notice also that the positive part of the data can be already highly-correlated so that **APRIORI** and other classical frequent set mining algorithms can even not tackle the computation of $Freq(r, \gamma)$, i.e., subsets of R that are frequent in the original binary database.
This tactic is not viable for practical data sets.

"Approximation" of Generalized Itemset Support Using Inclusion-Exclusion Formula

Assume in this part that we compute $Freq(r, \gamma)$, i.e., the collection of γ-frequent sets for the original binary database (positive items only). We already know that this problem is easier to solve. Using only that (positive) information, it remains possible to mine some association rules with negations using the well-known inclusion-exclusion formula adapted to our notation as following.

Lemma 22. *The support of a generalized itemset T can be computed exactly using the collection $FreqSup(r, \gamma)$ if $Pos(T) \cup Neg(T) \in Freq(r, \gamma)$ and is equal to:*

$$Sup(r,T) = \sum_{Pos(T)\subseteq X\subseteq T} (-1)^{|Neg(X)|} Sup(r, Pos(X)\cup Neg(X)) \ .$$

Example 14. *Let us consider the data set depicted as Figure 2.1 (Section 2.1.1). $FreqSup(r,2)$ is given in Example 1.*
Since $Pos(\{A,\overline{B}\}) \cup Neg(\{A,\overline{B}\}) = \{A,B\} \in Freq(r,2)$, $Sup(r,\{A,\overline{B}\}) = Sup(r,\{A\}) - Sup(r,\{A,B\})$ is known exactly (= 2). $Pos(\{A,\overline{B},D\}) \cup Neg(\{A,\overline{B},D\}) = \{A,B,D\} \notin Freq(r,2)$, so $Sup(r,\{A,\overline{B},D\}) = Sup(r,\{A,D\}) - Sup(r,\{A,B,D\})$ can not be evaluated exactly.

The number of terms of the sum is growing exponentially with the number of negative items in the generalized itemset. However, for generalized itemsets for which it can be computed exactly following Lemma 22, it can not be higher than the number of frequent sets ($|Freq(r,\gamma)|$), which is the number of all possible known terms of the sum. Therefore, the computation of the support of a generalized itemset remains tractable when the computation of $FreqSup(r,\gamma)$ is tractable.

In a separate study [Bykowski 02b] with a wider problem context (which is outside of the scope of this thesis) we described an alternative solution to the problem of approximating the support of a generalized itemset based only on the support information of positive frequent sets. The solution is based on linear programming.

Theorem 16. *The support of all subsets of a generalized itemset T can be derived from the collection $FreqSup(r,\gamma)$ if $Pos(T) \cup Neg(T) \in Freq(r,\gamma)$.*

Proof. Let $S \subset T$. Then $Pos(S) \cup Neg(S) \subseteq Pos(T) \cup Neg(T)$. If $Pos(T) \cup Neg(T) \in Freq(r,\gamma)$, all subsets of $Pos(T) \cup Neg(T)$ belong to $Freq(r,\gamma)$ (by the anti-monotonicity of the support, Lemma 1) including $Pos(S) \cup Neg(S)$. From Lemma 22, the support of S may therefore be computed from the collection $FreqSup(r,\gamma)$.

"Approximation" of Association Rules with Negations

Consider now the generation of rules from a frequent set $X \in Freq(r,\gamma)$. Let T be a full clause over X (see Definition 10). There are $2^{|X|}$ such full clauses from which the generalized association rule derivation can be performed. For that, we can use the same approach as the one presented as Algorithm 1, but with generalized itemsets as input. Following Theorem 16, the supports of subsets of T, which are required by Algorithm 1, can be evaluated exactly.
This way, we can generate a lot of generalized rules, but the collection is not complete w.r.t. the support and confidence criteria: there is no guarantee that it computes all the frequent and confident generalized rules. The rules missed by this approach are the ones

derived from every frequent full clause T over X such that X is not frequent. Short assessment may show that typically the result of this approach will be far from complete. A special case of this method can be performed for free during the standard association rule discovery (i.e., when applying Algorithm 1 on frequent sets in the original data set). It concerns a restricted form of generalized rule: rules with only positive items at the left-hand side and a generalized item (positive or negative) at the right-hand side.
After the computation of $FreqSup(r, \gamma)$, for each frequent set-support pair $<X, S_X> \in FreqSup(r, \gamma)$ and for each $A \in X$, it is possible to test two following rules if they are γ-frequent and ρ-confident. These rules are $X \setminus \{A\} \Rightarrow \{A\}$ and $X \setminus \{A\} \Rightarrow \{\overline{A}\}$, the second is a generalized rule. Testing the confidence of the second rule needs for the evaluation of $1-(Sup(r,X)/Sup(r,X\setminus\{A\})) \geq \rho$, i.e., $Sup(r,X)/Sup(r,X\setminus\{A\}) \leq 1-\rho$. The term $Sup(r,X)/Sup(r,X \setminus \{A\})$ is actually the confidence of $X \setminus \{A\} \Rightarrow \{A\}$ and is evaluated anyway (for the first rule). The support of $X \setminus \{A\} \Rightarrow \{\overline{A}\}$ is simply $Sup(r, X \setminus \{A\}) - Sup(r, X)$, and can easily be evaluated and compared to γ.
However, even limited to this form, some frequent and confident rules might be missed by the method. To be complete, we need to derive rules from all generalized frequent sets with at most 1 negative item, which is not the case in the setting. Such generalized itemsets can be extracted, but generally not from "positive" frequent sets only. We will come back to the question and experiment the computation of all generalized frequent sets with at most 1 negative item in Section 4.5.2.

We have seen above that mining a complete collection of γ-frequent ρ-confident generalized rules is generally not possible with "positive" frequent sets only. However, it is possible to trade precision (soundness) against completeness.
Consider now the possibility to compute imprecise interestingness measures for generalized rules by using only positive information. The idea is to substitute to the unknown terms an interval that bounds the possible values for the support and the confidence. Thus, it gives rise to an incertitude to the values of these measures. Let us consider a second dataset in Fig. 4.3.

Example 15. *Consider the rule* $\{A, \overline{E}\} \Rightarrow \{B\}$. $Sup(r, \{A, \overline{E}\} \Rightarrow \{B\}) = Sup(r, \{A, B\}) - Sup(r, \{A, B, E\})$ *and* $Conf(r, \{A, \overline{E}\} \Rightarrow \{B\}) = Sup(r, \{A, \overline{E}\} \Rightarrow \{B\})/(Sup(r, \{A\}) - Sup(r, \{A, E\}))$. $Sup(r, \{A, B, E\})$ *is not available (because* $\{A, B, E\}$ *is not in* $Freq(r, 4)$ *neither in* $\mathcal{B}d^-(Freq(r, 4))$*) but it can be estimated within the interval* $[0, 3]$ *(because it is infrequent). Thus,* $Sup(r, \{A, \overline{E}\} \Rightarrow \{B\}) \in [1, 4]$ *and* $Conf(r, \{A, \overline{E}\} \Rightarrow \{B\}) \in [\frac{1}{4}, 1]$.

This method has been proposed in [Mannila 96b]. It gives rise to several problems. First, a rule can be interesting (w.r.t. interestingness criteria), uninteresting or *unresolved*. The last case arises when intervals for support and/or confidence cross the minimum support and/or confidence thresholds. It happens in Example 15 for $\sigma = 4$ and $\rho = 0.9$ (for

$r =$

	A	B	C	D	E
1	x	x	x		
2			x	x	x
3		x			x
4	x	x	x		x
5	x	x	x		
6	x	x		x	
7	x				x
8		x	x		
9	x			x	

Itemset	Support
$\emptyset$	9
$\{A\}$	6
$\{B\}$	6
$\{C\}$	5
$\{D\}$	3
$\{E\}$	4
$\{A, B\}$	4
$\{A, C\}$	3
$\{A, E\}$	2
$\{B, C\}$	4
$\{B, E\}$	2
$\{C, E\}$	2

Figure 4.3: A binary database r and the support for each itemset from $Freq(r, 4) \cup \mathcal{B}d^-(Freq(r, 4))$.

example). By substituting an interval of possible values to unknown terms in the support formula of a generalized itemset (as in Lemma 22), some itemsets can not be classified as frequent or infrequent. Hence, for a complete association rule generation in the worst case, one has to enumerate every generalized itemset that is known not to be infrequent (i.e., that is frequent or unresolved).

We already adressed the question of approximation of rule confidences based on approximate supports of frequent sets in Section 4.2, but here, we face "large" intervals of possible values, giving rise to a huge amount of unresolved sets. Lot of them might be infrequent, but it is not possible to identify which ones, unless we use an alternative method.

Example 16. *Continuing Example 15, from $Freq(r, 4) \cup \mathcal{B}d^-(Freq(r, 4))$, we infer using the inclusion-exclusion formula 12 generalized frequent sets, 24 unresolved ones, and 207 infrequent ones*[8]*. On this toy example the number of unresolved generalized itemsets is above the number of generalized frequent sets. This would be worse if we had $Freq(r, 4)$ only, because these numbers become respectively 10, 53 and 180.*

Problems related to incertitude on itemset support amplify with the computation of the confidence or other interestingness measures (observe the experiments presented in Section 4.2, where small error bounds on supports have been amplified when computing the confidence). If we had a small amount of unsound answers beside a complete collection, or their evaluation function values were close to thresholds, the trade-off could be acceptable. But in case of confidence measure, most of candidate rules cannot be firmly accepted or rejected.

[8]In the implementation of the experiment, we required all intermediate results (e.g., $Sup(r, \{A, \overline{B}, E\})$ when computing the support of $\{A, \overline{B}, \overline{E}\}$) to be in the $[0, |r|]$ range, which enhances the precision of the resulting intervals.

Using Constraints

A third direction of research is the effective use of various constraints during the itemset mining, which has been already subject of several studies, e.g., [Srikant 97, Ng 98, Boulicaut 00d].

For that, we suppose we work on the "complemented" binary database, on which we use frequent sets mining algorithms. These algorithms are moreover enhanced to mine frequent sets under constraints (see [Srikant 97, Ng 98, Boulicaut 00d] for details).

Let r' be a binary database over R'. We say that a generalized itemset $T \subseteq R'$ satisfies a constraint predicate $\mathcal{C}$ if $\mathcal{C}(r', T)$ evaluates to true. In the framework of pattern theories (Section 3.1.1), a combination of constraint predicates q is a selection predicate. We recall that the pattern theory associated with such a selection predicate is $\mathcal{T} = Th(\mathcal{L}, r', q)$. Notice that we need the support of every generalized itemset since our goal is to derive rules. Therefore, formally we should have $\mathcal{L} = \{<T, S_T> : T \subseteq R' \wedge S_T \in \mathbb{N}\}$. To alleviate the reasoning, we consider that the support of T is can be correctly obtained (in any manner, some have been shown for various algorithms in this thesis), and so we set $\mathcal{L} = \{T : T \subseteq R'\}$.

Most of the tractable constraints involve $\mathcal{C}_F$: given the support threshold γ, $\mathcal{C}_F(r', T) \equiv Sup(r', T) \geq \gamma$.

The effective use of constraints concerns the computation of $\mathcal{T}$ using q. q typically is a conjunction of atomic constraints that specify the interesting itemsets from which "interesting" rules are to be derived. We saw that **APRIORI** uses the constraint $\mathcal{C}_F$ to prune the search space. Deriving all σ-frequent ρ-confident association rules requires all itemsets whose support exceeds σ. Therefore, we may clearily restrict the problem to selection predicates being a conjunction of $\mathcal{C}_F$ and another (possibly complex) constraint. In theory, it is possible to have a "generate and test" approach for that selection predicate, which consists in computing all generalized frequent sets and then test other constraints on them, but this is clearly intractable in many cases. In the following, we are going to look for viable alternatives.

4.5.2 Constraint-Based Approach

In this section, we identify the simple promissing constraints that can be used during the generalized frequent set mining. We are interested in constraints that do not compromise the outcome of the task, i.e., preserving as much potentially interesting patterns as possible. We show the practical efficiency of some of them and propose how to overcome the efficiency problems, thus leading to tractable, serviceable generalized rule mining tasks.

We focus on finding promissing constraints, not on the contraint theory. Therefore, theoretical aspects will consequently be succinct. A systematic studies of different constraints on itemset include [Srikant 97, Ng 98, Boulicaut 00d].

Relevant Constraints

A simple observation is that the **APRIORI** trick is not only applicable to the support constraint, but also to all anti-monotone constraint predicates (Definition 23 recalls what an anti-monotone predicate is). In case of generalized itemsets, when we say that a constraint is anti-monotone, we mean that it is anti-monotone w.r.t. generalized itemset inclusion.

Example 17. *$T \subseteq \{A, B, C, D\}$ and $T \cap \{A', B', C\} = \emptyset$ are anti-monotone constraints. Indeed, $\mathcal{C}_F$ is anti-monotone (see Lemma 2). Another anti-monotone constraint is $\mathcal{C}_{amnn}(r', T) \equiv |T \setminus R| \leq n$, which states that the generalized itemset should contain at most n negative items.*

All above anti-monotone constraints are of interest for the rule extraction process, because the user is often interested in associations involving only items from a pre-defined set of items (first case) or not at all interested in rules involving items from another pre-defined set (second case). $\mathcal{C}_F$ is of interest for managing the tractability of the extraction and for focusing only on statistically significant patterns.
Interesting to notice is that some constraints require the binary database instance to be evaluated (e.g., $\mathcal{C}_F$), while the others examine only the pattern itself. The latter, we call *syntactic* constraints.
Now, we consider the restricted, but quite often wide enough class of constraints which are monotone constraints.

Definition 41. (monotone constraint) A monotone constraint $\mathcal{C}_m$ is the negation of an anti-monotone constraint. If $\mathcal{C}_m$ is monotone, $\neg\, \mathcal{C}_m$ is anti-monotone: $\mathcal{C}_m(r', T) \Rightarrow \bigwedge_{S \supset T} \mathcal{C}_m(r', S)$.

Example 18. *Continuing our running example, $T \supseteq \{A, B, C, D\}$ and $T \cap \{A', B', C\} \neq \emptyset$ are monotone constraints. An interesting case of a monotone constraint, denoted as $\mathcal{C}_{alnp}$, is $|T \cap R| \geq n$. This constraint states that the generalized itemset T should contain at least n positive items.*

The above mentioned monotone constraints are useful in the following, potentially practical situations. The user may be interested only in rules with the left-hand-side involving: all items from a pre-defined set of items (first case) or any item from a pre-defined set of items (second case). E.g., it is quite justifiable not to consider rules involving only negative terms. There is a huge amount of them satisfying the support and confidence criteria, but they probably will not let any significant insight in the data. The third example contraint formalizes this idea. A nuance is that the user may require more than 1 positive item, leading to rules involving at least n positive items. $n = 1$ or $n = 2$ are the quite unconstraining, $n = 3$ or above is already quite a restrictive requirement (to be avoided, if other constraints are efficient enough).

Experiments

Now, we propose a small experimental study showing the efficiency of selected constraints. We selected a benchmark data set, **Mushroom** whose exigency level in frequent set mining is well-known. The second criterion was a rather-low number of positive items (119). When "complemented", the binary database reached a typical number of items in the schema (238).

We materialized that "complemented" binary database. At that stage, we observed the following possible source of useless of computations. The binarization of an attribute taking only two distinct values will typically produce 2 binary items. They are logical complements of each other, and we do not need to duplicate them when considering negative items. Clearly, the outcome of mining process is preserved. We observed that in **Mushroom** there are 6 such attributes, and the thus we could reduce the number of generalized items involved in the process by 12, greately reducing the required computational effort. Removed items were the ones considered "negative".

Above, we proposed some constraints that promise tractable mining generalized rules. Now, we evaluate their potential. First, we experiment the combination of the well-studied $\mathcal{C}_F$ constraint with syntactic (monotone and anti-monotone) constraints. In order to avoid the inefficient "generate and test" approach mentioned earlier, we push the constraints into the levelwise algorithm based on **APRIORI**, as described in [Boulicaut 01b].

In order to reduce the number of extracted rules without loosing potentially interesting rules, we chose to sacrifice only rules with many and/or sole negative items. In the first series of runs, we mine only positive itemsets, which will serve as a reference and could be a base for approximating generalized rule evaluation functions as described in the previous section. In the second series of runs, we allow 1 negative item per generalized itemset, i.e., we use $\mathcal{C}_{am1n}$, leading to rules with at most one negative item per rule. Thus, we allow a relatively insignificant entrance into the generalized itemset space. However, this example does not fulfil the objectives of the study. Thus, we decided to make a bigger step and not to constrain the number of negative items. To counterbalance the absence of contraints on negative items, in the third series of runs we constrained positive items — we required at least 3 positive items per generalized itemset ($\mathcal{C}_{al3p}$). As said previously, $\mathcal{C}_{al3p}$ is still quite constraining, and we wish to have even weaker constraints, e.g., "at least 2 positive items".

The results presented as Figure 4.4 (platform: Pentium IV, 1900 MHz, 1 GB RAM, Linux operating system).

We observed a roughly-exponential increase when reducing the support threshold, typical for frequent pattern mining (note the logarithmically scaled axes). However, the effect of the entrance into the generalized itemset space was severe — allowing at most one negative item (additional to "positive itemsets") resulted in a consequent, about-26-fold increase in execution time due to a similar increase in the number of extracted patterns.

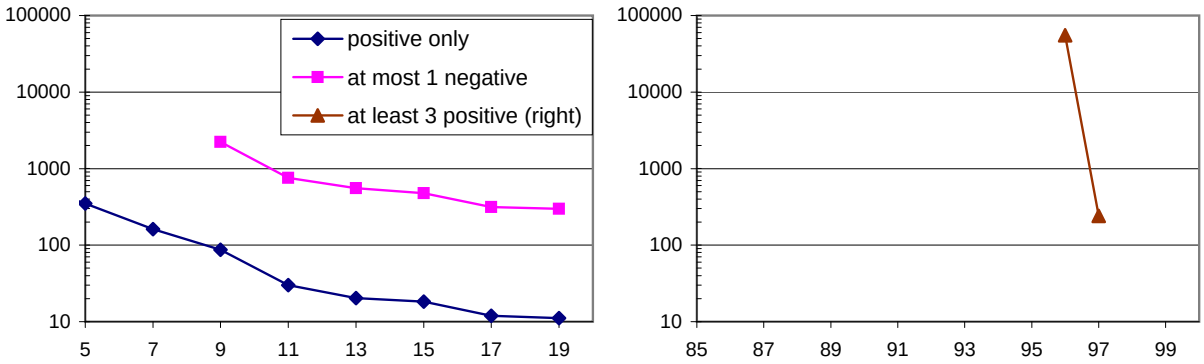

Figure 4.4: Effect of simple syntactic constraints on generalized itemset mining (extraction time in seconds vs relative support threshold).

This experiment shows the practical "weight" of negative items.
The third series of runs finished at an unexpectedly high support threshold of 96% facing a very steep explosion of running times. On Figure 4.4, the graphic with the relevant domain of relative supports is added on the right-hand-side. Thus, we gave up the experiment with weaker constraints, such as the constraint $\mathcal{C}_{al2p}$ (at least 2 positive items).

Then, we compared to the computation of the DBC without syntactic constraints on the complemented binary database. In order to make a fair comparison with the levelwise algorithm mining itemsets under constraints, we used the levelwise extractor of the DBC, i.e., **HLin-Ex**. We observed the running times reported on Figure 4.5.

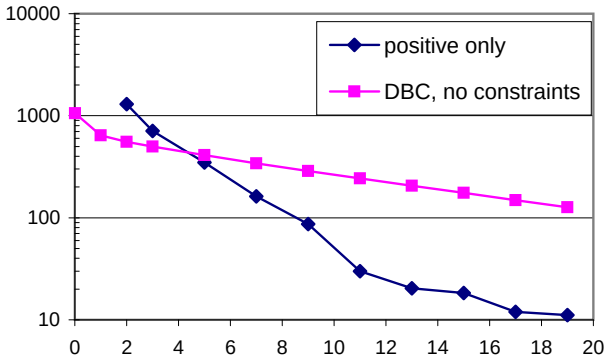

Figure 4.5: Effect of mining the condensed representation of all generalized frequent sets (extraction time in seconds vs relative support threshold).

We can observe relatively good performances, better even than for "positive" frequent sets (base of the approximated computation of generalized rules) at the lowest support thresholds. Indeed, the frequent sets with lower support theshold could not be extracted on out platform, unlike the DBC.
We emphasize that the DBC has been extracted on the complemented binary database without constraints. This experiment leads to the following possible approach for mining generalized itemsets.
The DBC is a lossless representation of frequent sets, thus we can reconstruct all of them without accessing the data set and procede with association rule derivation. In

the previous discussions, we however proved that the number of generalized frequent sets is tremendous. So, the complete reconstruction is not viable. Neither it is useful, because the user cannot be provided with excessive number of potentially interesting rules. Supposing that the user can guide the process with more precise constraints, either syntactic, or based on the itemset supports, we could regenerate generalized frequent sets from the DBC under these constraints.
Interesting constraints based on the itemset supports in the context of rule discovery are constraints stating that extracted itemsets should be 0-free or closed. Judicious combination of these constraints with syntactic constraints can lead to covers of generalized association rules under constraints (similar to the one presented in Section 4.4), which alleviates the interpretation of extracted rules.

The combination of fast computation of the condensed representation and of direct regeneration of any collection of frequent sets (without accessing the data set), should be advantageous over mining same frequent sets from data. Moreother, the condensed representation may be extracted off-line before the rule mining session (with users).
Therefore, we could consider the DBC as a generic input collection for generalized association rule generation under constraints. As emphasized for **Regen-Freq** and **Regen-FreqCl**, considering a regeneration procedure with a diffrent $\sigma' > \sigma$ amounts to selecting from $DBC(r, \sigma)$ the σ'-frequent elements, and applying the procedure on the reduced collection. Therefore, it is reasonable to compute off-line the collection $DBC(r, \sigma)$ for a lowest σ possible and use it for extraction of generalized rules under various constraints and support thresholds.
In our experiment, we could extract the DBC for as low support threshold as 1 row (Figure 4.5, the lowest reported support threshold). In other cases, we could mine the DBC for a very low threshold and use the representation if the rule support theshold applied by the user is the same or above. Otherwise, previously studied methods or a combination of evaluation against the data and regeneration from condensed representations could be implemented.
It is also possible to conceive condensed representations of collections of itemsets under constraints (see [Jeudy 02] for a depth-in study).
A hypothetical system providing such functionalities will be mentioned in perspectives of the thesis.

Chapter 5

Conclusion, Future works and Perspectives

5.1 Conclusion

Descriptive pattern discovery is an important KDD technique, when we have a large data set and we want to improve our understanding of the system mirrored in the data set. Mining frequent sets, a basic kind of descriptive patterns, has succesfully been carried out on some data set. Despite a relative potential of frequent sets for understanding other data sets, several data sets lead to too heavy or untractable frequent set computations. Among other reasons, the failure is due to the high correlation of data and the resulting explosion of the number of frequent sets.

In Chapter 3, we proposed to avoid some of the load related to frequent set computation, based on the observed redundancy in the collections of frequent sets. First, we observed that some past approaches can be shown to follow the idea (in Section 3.1.5).

Then, we studied in depth two kinds of redundancies and based on them we proposed two major condensed representations. As shown in Sections 3.2 and 3.3, they are much smaller than the collections of frequent sets that they represent and more efficient to mine. They are also competitive when compared to frequent closed sets, probably the only structure proposed in the past that we could be alligned to.

We provided detailed theoretical fundamentals, algorithms and experimental proofs that some typical, highly correlated data sets can benefit from the findings. The resulting methods allow in general to widen the applicability of frequent set-based knowledge discovery, either to lower thresholds given the same resources, or to more challanging data sets.

We also proposed a framework for a uniform description of condensed representations.

In Chapter 4, we provided tools for taking advantage of condensed representations when putting in standard techniques based on frequent sets, frequent 0-free sets or frequent closed sets. Indeed, we provide methods, algorithms and experimental proofs that show

that the techniques mentioned above can still be provided with the patterns they need (frequent sets, frequent 0-free sets or frequent closed sets) and that with a considerable speed-up.
Finally, we consider a mining task proposed in the past, but which cannot be tackled in general (extracting association rules with negations). We provide the discussion concluding with a potentially very advantageous use of one of the original condensed representations proposed in the thesis.

The results of the work are centered on improving the ground for other already known techniques based on computing frequent sets. These results have already been starting points of further studies carried out independently, e.g., [Calders 02].
Moreover, the concrete prototypes developed in relation to the thesis have been used to validate the viability of a new research direction and to explore real data sets.
A new research direction, namely the automatic generation of propagation rules (logical rules used in contraint programming to solve groups of contraints), has been pre-studied for viability using the **Min-Ex** extractor. Further, very successful developement is described in [Abdennadher 00].
Publications showing the practical use of tools developed in this thesis to explore real data sets include medical data analysis [Boulicaut 01a, Robardet 02] et Web usage mining [Bykowski 00].

5.2 Future Works

The fundamental advances caused by the use of condensed representations will be complemented by their instances dedicated to more specific uses of frequent sets.
One application seems however to be essential in the immediate future. It is about the effective use of constraints. It has been studied previously, e.g., in [Srikant 97, Ng 98, Boulicaut 00d]. Guiding the KDD system to important patterns using constraints on patterns allows to explore the data in contexts when the total number of patterns in a pattern discovery task is intractable either to compute or to post-process.
Combining these ideas with the use of condensed representations seems very promissing. We observed that the combination is indeed very efficient (see [Bykowski 02a] for extracting an association rule summary) and we are going to formalize the ideas.

Finally, we point that other patterns (e.g., sequential) could directly benefit from condensed representation framework. Perhaps, we could apply it also to association rules. Could we thus filter out additional or different redundant rules than other methods, making the results easier to analyse or extract? We will also investigate this in the future.

5.3 Long-term Perspectives

At the horizon, we can imagine database managment systems that allow among others discovery of different classes of patterns using various, user specified evaluation functions. Such systems are called *inductive* database management systems (see e.g., [Mannila 97a]). They could benefit from materializing some redundant information (in a similar way as relational database systems benefit from indexes). Such redundant information would be gathered off-line before the mining process, to improve the on-line behavior of most mining processes.

Given that the patterns are usually quite complex from various points of view and require a high computational effort to get them, the speed-up might be worth a good investment. Typically, one would be interested in sufficient statistics as redundant information, i.e., summaries that can be substituted for the whole extensional data to avoid repetitive probing of the selection predicate for all candidate patterns. On the other hand, sufficient statistics for some queries might be too highly processed to benefit other classes of queries. Therefore, a good compromise between the generality (flexibility) of summaries and their contribution to the efficiency must be found.

The condensed representations proposed in this thesis are sufficienlty flexible to fulfill the task when frequent sets and their downstream uses are considered (we showed how to regenerate every frequent set). At the same time, the proposed condensed representations are efficient to obtain and the collections of frequent sets can be regenerated efficiently from them (even if in some cases we can circumvent the regeneration all frequent sets and take better advantage of condensed representations).

We believe that the condensed representations proposed in the thesis will be constituents of the inductive database management systems.

Bibliography

[Abdennadher 00] ABDENNADHER S. and RIGOTTI C. Automatic generation of propagation rules for finite domains. *Proc. of the Sixth International Conference on Principles and Practice of Constraint Programming (CP'00).* Singapore, September 2000, p 18–34.

[Abiteboul 95] ABITEBOUL SERGE, HULL RICHARD and VIANU VICTOR. *Foundations of Databases.* Addison-Wesley, 1995. 685p.

[Agarwal 99] AGARWAL RAMESH C., AGGARWAL CHARU C. and PRASAD V. V. A tree projection algorithm for finding frequent itemsets. *High Performance Data Mining Workshop.* San Juan, Puerto Rico, April 1999, p 15–31.

[Agarwal 01] AGARWAL RAMESH C., AGGARWAL CHARU C. and PRASAD V. V. A tree projection algorithm for finding frequent itemsets. *Journal of Parallel and Distributed Computing*, March 2001, vol 61, number 3, p 350–371.

[Aggarwal 98] AGGARWAL CHARU C. and YU PHILIP S. Online generation of association rules. *Proc. of the Fourteenth International Conference on Data Engineering (ICDE'98).* Orlando, Florida, USA, February 1998, p 402–411.

[Agrawal 93] AGRAWAL RAKESH, IMIELINSKI TOMASZ and SWAMI ARUN. Mining association rules between sets of items in large databases. Edited by BUNEMAN PETER and JAJODIA SUSHIL. *Proc. of the 1993 ACM SIGMOD International Conference on Management of Data.* Washington, D.C., USA, May 1993, p 207–216.

[Agrawal 94] AGRAWAL RAKESH and SRIKANT RAMAKRISHNAN. Fast algorithms for mining association rules in large databases. *Proc. of the Twentieth International Conference on Very Large Data Bases (VLDB'94).* Santiago de Chile, Chile, September 1994, p 487–499.

[Agrawal 95] AGRAWAL RAKESH and SRIKANT RAMAKRISHNAN. Mining sequential patterns. Edited by YU PHILIP S. and CHEN ARBEE L. P.. *Proc. of the Eleventh International Conference on Data Engineering (ICDE'95).* Taipei, Taiwan, March 1995, p 3–14. IEEE Computer Society.

[Agrawal 96] AGRAWAL RAKESH, MANNILA HEIKKI, SRIKANT RAMAKRISHNAN, TOIVONEN HANNU and VERKAMO

A. INKERI. *Fast discovery of association rules. Advances in Knowledge Discovery and Data Mining*, p 307–328. 1996.

[Bastide 00a] BASTIDE YVES, PASQUIER NICOLAS, TAOUIL RAFIK, LAKHAL LOTFI and STUMME GERD. Mining minimal non-redundant association rules using frequent closed itemsets. *Proc. of the Sixth International Conference on Rules and Objects in Databases (DOOD'00), London.* Lecture Notes in Computer Science. Springer, July 2000, p 972–986.

[Bastide 00b] BASTIDE YVES, TAOUIL RAFIK, PASQUIER NICOLAS, STUMME GERD and LAKHAL LOTFI. Mining frequent patterns with counting inference. *SIGKDD Explorations*, December 2000, vol 2, number 2, p 66–75.

[Bayardo, Jr. 98] BAYARDO, JR. ROBERTO J. Efficiently mining long patterns from databases. *Proc. of the 1998 ACM SIGMOD International Conference on Management of Data.* ACM Press, 1998, p 85–93.

[Bayardo, Jr. 99a] BAYARDO, JR. ROBERTO J., AGRAWAL RAKESH and GUNOPULOS DIMITRIOS. Constraint-based rule mining in large, dense databases. *Data Mining and Knowledge Discovery*, 1999, vol 4, number 2-3, p 217–240.

[Bayardo, Jr. 99b] BAYARDO, JR. ROBERTO J., AGRAWAL RAKESH and GUNOPULOS DIMITRIOS. Constraint-based rule mining in large, dense databases. *Proc. of the Fifteenth International Conference on Data Engineering (ICDE'99).* March 1999, p 188–197.

[Bernstein 76] BERNSTEIN PHILIP A. Synthesizing third normal form relations from functional dependencies. *Communications of the ACM*, December 1976, vol 1, number 4, p 277–298.

[Boulicaut 00a] BOULICAUT JEAN-FRANÇOIS and BYKOWSKI ARTUR. Frequent closures as a concise representation for binary data mining. *Proc. of the Fourth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00).* volume 1805 of *LNAI*, Kyoto, JP, April 2000, p 62–73. Springer-Verlag.

[Boulicaut 00b] BOULICAUT JEAN-FRANÇOIS, BYKOWSKI ARTUR and JEUDY BAPTISTE. Towards the tractable discovery of association rules with negations. *Proc. of the Fourth International Conference on Flexible Query Answering Systems (FQAS'00).* Warsaw, Poland, October 2000, p 425–434.

[Boulicaut 00c] BOULICAUT JEAN-FRANÇOIS, BYKOWSKI ARTUR and RIGOTTI CHRISTOPHE. Approximation of frequency queries by mean of free-sets. *Proc. of the Fourth European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'00).* Lyon, France, September 2000, p 75–85.

[Boulicaut 00d] Boulicaut Jean-François and Jeudy Baptiste. Using constraints during itemset mining: should we prune or not? *Actes des Seizièmes Journées Bases de Données Avancées (BDA'00).* Blois, France, October 2000, p 221–237.

[Boulicaut 01a] Boulicaut Jean-François and Crémilleux Bruno. Delta-strong classification rules for predicting collagen diseases. *Proc. of the PKDD'01 Discovery Challenge on Thrombosis Data co-located with the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases PKDD'01.* Freiburg, Germany, September 2001, p 29–38.

[Boulicaut 01b] Boulicaut Jean-François and Jeudy Baptiste. Mining free itemsets under constraints. *Proc. of the International Database Engineering and Application Symposium (IDEAS'01).* Grenoble, France, July 2001, p 322–329.

[Boulicaut 03] Boulicaut Jean-François, Bykowski Artur and Rigotti Christophe. Free-sets : a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery*, 2003. To appear.

[Brin 97a] Brin Sergey, Motwani Rajeev, Ullman Jeffrey D. and Tsur Shalom. Dynamic itemset counting and implication rules for market basket data. *Proc. of ACM SIGMOD Conference on Management of Data (SIGMOD'97).* Tucson, USA, May 1997, p 255–264. ACM Press.

[Brin 97b] Brin Sergey, Motwani Rejeev and Silverstein Craig. Beyond market baskets: Generalizing association rules to correlations. *Proc. of ACM SIGMOD Conference on Management of Data (SIGMOD'97).* Tucson, USA, May 1997, p 265–276.

[Bykowski 99] Bykowski Artur. *Frequent Set Discovery in Highly-correlated Data.* Technical report (Master of Science thesis), INSA-Lyon, LISI, Villeurbanne, France, July 1999. 30p.

[Bykowski 00] Bykowski Artur and Gomez-Chantada Lucia. Frequent itemset extraction in highly-correlated data: a web usage mining application. *Proc. of the Int. Workshop on Web Knowledge Discovery and Data Mining (WKDDM'00).* Kyoto, JP, April 2000, p 27–42.

[Bykowski 01] Bykowski Artur and Rigotti Christophe. A condensed representation to find frequent patterns. *Proc. of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'01).* Santa Barbara, CA, USA, May 2001, p 267–273. ACM.

[Bykowski 02a] Bykowski Artur, Daurel Thomas, Méger Nicolas and Rigotti Christophe. Association maps as integrity constraints in inductive databases. *Proc. of the Int. Workshop on Knowledge Discovery in Inductive Databases (KDID'02)*. Helsinki, Finland, August 2002. To appear.

[Bykowski 02b] Bykowski Artur, Seppänen Jouni K. and Hollmén Jaakko. Model-independent bounding of the supports of boolean formulae in binary data. *Proc. of the Int. Workshop on Knowledge Discovery in Inductive Databases (KDID'02)*. Helsinki, Finland, August 2002. To appear.

[Bykowski 03] Bykowski Artur and Rigotti Christophe. Dbc: A condensed representation of frequent patterns for efficient mining. *Information Systems*, 2003. To appear.

[Calders 02] Calders Toon and Goethals Bart. Mining all non-derivable frequent itemsets. *Proc. of the sixth European Conf. on Principles of Data Mining and Knowledge Discovery in Databases (PKDD'02)*. Helsinki, Finland, August 2002, p 74–85.

[Codd 70] Codd Edgar F. A relational model of data for large shared data banks. *Communications of the ACM*, June 1970, vol 13, number 6, p 377–387.

[Cristofor 02] Cristofor Laurentiu and Simovici Dan. *Generating an informative cover for association rules*. Technical report, University of Massachusetts, Boston, 2002. 16p.

[Das 98] Das Gautam, Mannila Heikki and Ronkainen Pirjo. Similarity of attributes by external probes. *Proc. of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*. New York City, New York, USA, August 1998, p 23–29.

[Diop 02] Diop Cheikh Talibouya, Giacometti Arnaud, Laurent Dominique and Spyratos Nicolas. Composition of mining contexts for efficient extraction of association rules. *Proc. of the Eighth International Conference on Extending Database Technology (EDBT'02)*. Prague, Czech Republic, March 2002, p 106–123.

[European Space Agency 02] European Space Agency . *Envisat — the detail's in the data...* **[on-line]**. ESA press release, World Wide Web, available at `<http://www.esa.int/export/esaCP/ESAS3ANSRWC_index_0.html>` (last accessed on October 2^{nd}, 2002), January 2002.

[Giacometti 02] Giacometti Arnaud, Laurent Dominique and Diop Cheikh Talibouya. Condensed representations for sets of mining queries. *Proc. of the Int. Workshop on Knowledge Discovery in Inductive Databases (KDID'02)*. Helsinki, Finland, August 2002. To appear.

[Guha 98] GUHA SUDIPTO, RASTOGI RAJEEV and SHIM KYUSEOK. Cure: An efficient clustering algorithm for large databases. Edited by HAAS LAURA M. and TIWARY ASHUTOSH. *Proc. of ACM SIGMOD International Conference on Management of Data (SIGMOD'98).* Seattle, Washington, USA, June 1998, p 73–84. ACM Press.

[Guillaume 98] GUILLAUME SYLVIE, GUILLET FABRICE and PHILIPPÉ JACQUES. Improving the discovery of association rules with intensity of implication (short paper). *Proc. of the Second European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'98).* volume 1510 of *LNAI*, Nantes, France, September 1998, p 318–327. Springer-Verlag.

[Han 98] HAN EUI-HONG (SAM), KARYPIS GEORGE, KUMAR VIPIN and MOBASHER BAMSHAD. Hypergraph based clustering in high-dimensional data sets: A summary of results. *IEEE Bulletin of the Technical Committee on Data Engineering*, March 1998, vol 21, number 1, p 15–22.

[Han 00] HAN JIAWEI, PEI JIAN and YIN YIWEN. Mining frequent patterns without candidate generation. *Proc. of ACM SIGMOD Conference on Management of Data (SIGMOD'00).* Dallas, Texas, May 2000, p 1–12.

[Hand 01] HAND DAVID J., MANNILA HEIKKI and SMYTH PADHRAIC. *Principles of Data Mining.* MIT Press, 2001. 425p.

[Huhtala 99] HUHTALA YKÄ, KÄRKKÄINEN JUHA, PORKKA PASI and TOIVONEN HANNU. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 1999, vol 42, number 2, p 100–111.

[Jeudy 02] JEUDY BAPTISTE. *Optimisation de requetes inductives : application à l'extraction sous contraintes de règles d'association. [in French].* PhD Thesis: INSA-Lyon, 2002. To appear.

[Kivinen 92] KIVINEN JYRKI and MANNILA HEIKKI. Approximate dependency inference from relations. Edited by BISKUP JOACHIM and HULL RICHARD. *Proc. of the fourth International Conference Database Theory (ICDT'92).* volume 646 of *Lecture Notes in Computer Science*, Berlin, Germany, October 1992, p 86–98.

[Klemettinen 94] KLEMETTINEN MIKA, MANNILA HEIKKI, RONKAINEN PIRJO, TOIVONEN HANNU and VERKAMO A. INKERI. Finding interesting rules from large sets of discovered association rules. *Proc. of the Third International Conference on Information and Knowledge Management (CIKM'94).* Gaithersburg, Maryland, USA, November 1994, p 401–407.

[Lin 98] LIN DAO-I and KEDEM ZVI M. Pincer search: A new algorithm for discovering the maximum frequent set. Edited by SCHEK HANS-JÖRG, SALTOR FÈLIX, RAMOS ISIDRO and ALONSO GUSTAVO. *Proc. of the 6th International Conference on Extending Database Technology (EDBT'98)*. Valencia, Spain, March 1998, p 105–119.

[MacQueen 67] MACQUEEN JAMES B. Some methods for classification and analysis of multivariate observations. *In Proc. of the Fifth Berkley Symposium on Mathematical Statistics and Probability*. Berkeley, California, USA, June 1967, p 281–297.

[Mannila 94] MANNILA HEIKKI, TOIVONEN HANNU and VERKAMO A. INKERI. Efficient algorithms for discovering association rules. *Proc. of the AAAI Workshop on Knowledge Discovery in Databases (KDD'94)*. Seattle, Washington, USA, July 1994, p 181–192.

[Mannila 95] MANNILA HEIKKI, TOIVONEN HANNU and VERKAMO A. INKERI. Discovering frequent episodes in sequences. *In Proc. of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*. Montreal, Canada, August 1995, p 210 – 215.

[Mannila 96a] MANNILA HEIKKI and TOIVONEN HANNU. Discovering generalized episodes using minimal occurrences. *Proc. of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. Portland, USA, August 1996, p 146–151.

[Mannila 96b] MANNILA HEIKKI and TOIVONEN HANNU. Multiple uses of frequent sets and condensed representations. *In Proc. of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. Portland, USA, August 1996, p 189–194.

[Mannila 97a] MANNILA HEIKKI. Inductive databases and condensed representations for data mining. *Proceedngs of the 1997 International Logic Programming Symposium (ILPS'97)*. Port Jefferson, NY, USA, October 1997, p 21–30.

[Mannila 97b] MANNILA HEIKKI and TOIVONEN HANNU. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1997, vol 1, number 3, p 241–258.

[Mannila 97c] MANNILA HEIKKI, TOIVONEN HANNU and VERKAMO A. INKERI. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1997, vol 1, number 3, p 259–289.

[Mueller 95] MUELLER ANDREAS. *Fast Sequential and Parallel Algorithms for Association Rule Mining: A Comparison*. Technical report CS-TR-3515, College Park, MD, USA, 1995. 76p.

[Ng 98] Ng Raymond, Lakshmanan Laks V.S., Han Jiawei and Pang Alex. Exploratory mining and pruning optimization of constrained association rules. *Proc. of ACM SIGMOD Conference on Management of Data (SIGMOD'98)*. Seattle, USA, 1998, p 13–24.

[Novelli 01] Novelli Noël and Cicchetti Rosine. Functional and embedded dependency inference: a data mining point of view. *Information Systems*, 2001, vol 26, number 7, p 477–506.

[Ore 44] Ore Oystein. Galois connexions. *Transactions of the American Mathematical Society*, February 1944, p 493–513.

[Pasquier 99a] Pasquier Nicolas, Bastide Yves, Taouil Rafik and Lakhal Lotfi. Closed set based discovery of small covers for association rules. *Actes des Quinzièmes Journées Bases de Données Avancées (BDA'99)*. Bordeaux, France, October 1999, p 361–381.

[Pasquier 99b] Pasquier Nicolas, Bastide Yves, Taouil Rafik and Lakhal Lotfi. Discovering frequent closed itemsets for association rules. Edited by Beeri Catriel and Buneman Peter. *Proc. of the seventh International Conference on Database Theory - ICDT '99, Jerusalem, Israel.* volume 1540 of *Lecture Notes in Computer Science*. Springer, January 1999, p 398–416.

[Pasquier 99c] Pasquier Nicolas, Bastide Yves, Taouil Rafik and Lakhal Lotfi. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 1999, vol 24, number 1, p 25–46.

[Pei 00] Pei Jian, Han Jiawei and Mao Runying. Closet: An efficient algorithm for mining frequent closed itemsets. *Proc. of the 2000 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery.* Dallas, Texas, May 2000, p 21–30.

[Piatetsky-Shapiro 91] Piatetsky-Shapiro Gregory. *Discovery, Analysis, and Presentation of Strong Rules. Knowledge Discovery in Databases*, p 229–248. AAAI Press, Menlo Park, CA, 1991.

[Robardet 02] Robardet Céline, Crémilleux Bruno and Boulicaut Jean-François. Characterization of unsupervized clusters by means of the simplest association rules: an application for child's meningitis. *Proc. of the 7th Workshop on Intelligent Data Analysis in Medicine and Pharmacology IDAMAP'02.* Lyon, France, July 2002. To appear.

[Smyth 92] Smyth Padhraic and Goodman Rodney M. An information theoretic approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering*, August 1992, vol 4, number 4, p 301–316.

[Srikant 95] SRIKANT RAMAKRISHNAN and AGRAWAL RAKESH. Mining generalized association rules. *Proc. of the Twenty Oneth International Conference on Very Large Data Bases (VLDB'95).* Zurich, Switzerland, September 1995, p 407–419.

[Srikant 96] SRIKANT RAMAKRISHNAN and AGRAWAL RAKESH. Mining quantitative association rules in large relational tables. Edited by JAGADISH H. V. and MUMICK INDERPAL SINGH. *Proc. of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD'96).* Montreal, Quebec, Canada, June 1996, p 1–12.

[Srikant 97] SRIKANT RAMAKRISHNAN, VU QUOC and AGRAWAL RAKESH. Mining association rules with item constraints. Edited by HECKERMAN DAVID, MANNILA HEIKKI, PREGIBON DARYL and UTHURUSAMY RAMASAMY. *Proc. of the 3rd International Conference on Knowledge Discovery and Data Mining, KDD'97.* AAAI Press, August 1997, p 67–73.

[Suzuki 98] SUZUKI EINOSHIN and KODRATOFF YVES. Discovery of surprising exception rules based on intensity of implication. *Proc. of the Second European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'00).* volume 1510 of *LNAI*, Nantes, France, September 1998, p 10–18. Springer-Verlag.

[Tan 02] TAN PANG-NING, KUMAR VIPIN and SRIVASTAVA JAIDEEP. Selecting the right interestingness measure for association patterns. *Proc. of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'02).* July 2002. To appear.

[Toivonen 95] TOIVONEN HANNU, KLEMETTINEN MIKA, RONKAINEN PIRJO, HÄTÖNEN KIMMO and MANNILA HEIKKI. Pruning and grouping of discovered association rules. *Workshop Notes of the ECML-95 Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases.* Heraklion, Crete, Greece, April 1995, p 47–52. MLnet.

[Toivonen 96] TOIVONEN HANNU. Sampling large databases for association rules. *Proc. of the Twenty Twoth International Conference on Very Large Data Bases (VLDB'96).* Mumbay, India, September 1996, p 134–145.

[Ullman 80] ULLMAN JEFFREY D. *Principles of Database Systems.* Computer Science Press, 1st edition, 1980. 484p.

[Vapnik 95] VAPNIK VLADIMIR N. *The Nature of Statistical Learning Theory.* SpringerVerlag, 1995. 304p.

[Wille 82] WILLE RUDOLF. Restructuring lattice theory: An approach based on hierarchies of concepts. Edited by RIVAL IVAN. *Proc. of the Conference on Ordered Sets.* 1982, p 445–470.

[Zaki 97] ZAKI MOHAMMED JAVEED, PARTHASARATHY SRINIVASAN, OGIHARA MITSUNORI and LI WEI. Parallel algorithms for discovery of association rules. *Data Mining and Knowledge Discovery*, 1997, vol 1, number 4, p 343–373.

[Zaki 00] ZAKI MOHAMMED JAVEED. Generating non-redundant association rules. *Proc. of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*. Boston, MA, USA, August 2000, p 34–43.

[Zaki 02] ZAKI MOHAMMED JAVEED and HSIAO CHING-JUI. Charm: An efficient algorithm for closed itemset mining. *Proc. of the Second SIAM International Conference on Data Mining (SDM'02)*. Arlington, VA, USA, April 2002, p 357–373.

Annex A
Résumé étendu en français

Ce chapitre présente en français une version synthétisée de la thèse exposée dans les chapitres précédents.

Résumé court

L'extraction des connaissances dans les données (ECD) est un domaine relativement nouveau qui vise à mieux exploiter l'information contenue dans de grandes collections de données stockées et disponibles. L'extraction de motifs intéressants est un sous-domaine de l'ECD.
Ce sous-domaine a connu récemment un développement impressionnant dû d'une part à une pression accrue des propriétaires de données sous-exploitées et d'autre part à la réponse des chercheurs par de nombreux résultats théoriques et pratiques. A l'origine du développement de ces méthodes la plupart des données analysées provenaient du domaine de la vente et les motifs intéressants se présentaient sous forme de règles d'association. Des solutions performantes à ce problème pratique ont été élaborées, la première étant l'algorithme **APRIORI**.
Puis, les propriétaires d'autres types de données se sont interrogés sur l'utilité de ces premières solutions pour analyser leurs données. En effet, ces données, et/ou la forme de motifs sous laquelle les propriétaires voulaient voir les connaissances exprimées, étaient malheureusement différentes. Souvent, dans ces cas-là, **APRIORI** était inefficace voire intractable. Par la suite, la recherche a pris le défi d'élargir le champ d'application d'extraction de motifs aux domaines où **APRIORI** ne pouvait pas être utilisé directement.
Mes collègues et moi avons étudié les problèmes liés à l'extraction de motifs intéressants dans des collections de données d'origine différentes, en particulier les problèmes liés au grand nombre de motifs fréquents valides dans les données non similaires aux données de ventes.
Nous avons obtenu des améliorations significatives des performances et ceci essentiellement grâce à l'utilisation de représentations condensées originales de motifs fréquents. Ceci est complémentaire à certaines techniques d'optimisation du calcul de fonction d'évaluation

de motifs, c'est-à-dire à des approches étudiées par la plupart des prolongements majeurs d'**APRIORI** proposées antérieurement aux travaux de cette thèse.
Nos méthodes extraient une collection de motifs qui peut être très différente de la collection cible de motifs, en estimant qu'elle sera beaucoup plus efficace à calculer dans certains types de données. De plus, cette collection, différente de la collection cible de motifs, doit permettre une régénération efficace de la collection cible, sans pour autant nécessiter d'accéder aux données analysées. Comme la représentation intermédiaire est souvent beaucoup plus petite que la collection cible, nous la désignons sous le terme *représentation condensée*.
L'utilisation de représentations condensées est relativement novatrice dans le domaine. Toutefois, comme nous avons démontré dans la thèse, quelques solutions existantes ont été construites implicitement sur des représentations condensées.
La contribution principale de cette thèse est la proposition de nouvelles représentations condensées pour des motifs élémentaires, ainsi que des algorithmes pour extraire ces représentations condensées et régénérer, à partir d'elles, les collections de motifs cibles. Nous montrons les avantages de ces représentations condensées par rapport aux méthodes existantes.

Mot-clés : extraction de connaissances, motifs descriptifs, règles d'association, représentations condensées

Introduction

L'extraction de connaissances dans les données (ECD), ou "Data Mining", est un processus interactif et itératif d'induction et de synthèse des faits présents dans de grands volumes de données. Il s'agit de faits nouveaux, intéressants, pourtant non explicites.
Les *règles d'association* sont une forme possible de représentation de connaissances (par ex. $A \wedge B \Rightarrow C \wedge D$ dit que lorsque les événements A et B sont observés, ils sont souvent accompagnés par les événements C et D).

Règles d'association. Les règles d'association s'avèrent très intéressantes dans de multiples domaines. Ces règles sont de type :

$$X \Rightarrow Y \quad [support, confiance]$$

où X et Y sont des conjonctions de propriétés qu'une transaction peut valider ou non. Le *support* mesure le nombre d'objets concérnés par la règle, la *confiance* mesure la validité de la règle (toutes les deux seront définies formelement plus tard).
Bien que difficile, l'extraction de telles règles est à présent possible dans plusieurs jeux de données réels dits *peu denses*. Dans des jeux de données dits *très denses*, l'extraction est trop coûteuse, voire intractable, si nous utilisons les approches actuellement disponibles.

Un projet d'extraction des connaissances demande la participation d'un expert en ECD, parce que des aspects techniques (combinatoires etc.) des outils d'extraction de connaissances souvent ne sont pas bien compris par les utilisateurs.
Par exemple, une croyance naïve par les utilisateurs potentiels des motifs extraits consiste à penser que l'énumération et l'évaluation de tous les motifs possibles est la bonne manière de trouver ceux qui sont intéressants.
Prenons l'exemple des règles d'association. Supposons qu'il y a seulement 200 produits dans un supermarché. Dans ce cas, le nombre de règles d'association possibles (en ignorant le support et la confiance pour l'instant) est $3^{200} - 2^{200}$, c'est-à-dire autour de 10^{95}. C'est plus que le nombre d'atomes dans l'univers, ce qui fait que c'est vraiment dur d'imaginer que les ordinateurs, même dans un futur lointain, pourraient les énumérer individuellement ou stocker exhaustivement.

Afin d'extraire des règles d'association, le processus typique commence par extraire des *ensembles fréquents* (nous allons définir formellement cette notion plus loin). Un traitement simple, décrit dans [Agrawal 96], permet de trouver toutes les règles d'association à partir d'ensembles fréquents.
C'est l'étape d'extraction d'ensembles fréquents qui pose problème dans les données denses. Le résultat de l'extraction est souvent de grande taille et la tractabilité n'est

pas toujours possible dans les conditions d'extraction que les utilisateurs veulent couvrir (par ex. seuils de support et de confiance trop bas).
Voici quelques exemples d'autres motifs liés aux ensembles fréquents.

Règles généralisées. Les règles généralisées sont une extension du concept de règles d'association. Une règle généralisée diffère par rapport à une règle d'association par le fait que l'antécédent et le conséquent de celle-ci peuvent être des expressions booléennes comprenant différents opérateurs : négation, disjonction et conjonction (dans le cas des règles d'association l'unique opérateur possible est la conjonction). Les règles généralisées ont été introduites parce qu'elles ont un potentiel beaucoup plus large que les premières — elles ont un potentiel de description de phénomènes plus riche, mais elles sont plus nombreuses que les règles d'association.

Règles d'épisode. Les règles d'épisode sont des expressions qui définissent une corrélation entre des événements étalés dans le temps (par ex. événements A, B et C séparés de moins de 4 secondes et apparaissant dans n'importe quel ordre sont souvent accompagnés d'un événement D qui suit le dernier des trois précédents en 3 secondes).
Un pré-traitement et post-traitement peuvent ramener le cas de recherche de règles d'épisode à celle de recherche de règles d'association (bien que cette méthode peut ne pas être la plus efficace). Ainsi, nous obtenons une autre application du concept d'ensembles fréquents.

Phrases-clés. Une sélection appropriée d'ensembles fréquents peut servir *in fine* à décrire sommairement des données. Par exemple, la collection d'ensembles fréquents maximaux (c'est à dire les ensembles fréquents dont tous les sur-ensembles ne sont pas fréquents) peut être considérée comme des phrases-clés d'un document texte dans le cas du "text mining".

Désormais, nous nous concentrerons sur les ensembles fréquents et nous mentionnerons seulement de temps en temps d'autres motifs. Le choix des ensembles fréquents a été dicté par les faits :

- les ensembles fréquents ont un simple et précise signification, ils sont une classe de motifs éducative — relativement facile à comprendre et observer la disparité entre la simplicité de la définition de tâche et les difficultés pratiques liées à leur calcul ; la simplicité rend également des résultats de la recherche facilement assimilables par d'autres chercheurs,

- ils sont génériques, c'est-à-dire, certains concepts et techniques définies pour les ensembles fréquents peuvent être adaptés à d'autres motifs,

- ils partagent plusieurs propriétés avec d'autres motifs, mais semblent être les plus fondamentaux,
- il est intéressant de noter que les ensembles fréquents ont été utilisés à d'autres fins que l'extraction de règles d'association. Par exemple, des règles de corrélation, des règles d'association généralisées et certaines mesures de similarité sont basés sur des ensembles fréquents. Egalement, certains autres types de motifs sont des extensions claires d'ensembles fréquents et règles d'association, par exemple les épisodes fréquents et les règles d'épisodes. Ainsi, les ensembles fréquents apparaissent comme un concept générique ayant des usages multiples,
- les ensembles fréquents sont utilisés pour effectuer une analyse exploratoire dans des domaines d'applications variés.

Les raisons similaires doivent avoir incité d'autres chercheurs à étudier cette classe des motifs, parce que la plupart de publications récentes au sein de la communauté d'extraction de motifs porte sur les ensembles fréquents et règles d'associations (qui sont dérivées des ensembles fréquents).
Ceci donne une idée claire des motivations pour étendre l'utilisation des ensembles fréquents aux données denses. Pour celà il fallait chercher des méthodes d'extraction d'ensembles fréquents qui fonctionnent même face à des données denses.
Cette thèse vise à trouver des voies d'extraction des ensembles fréquents et des règles descriptives (d'association) dans ces cas.

Approche utilisée

L'approche étudiée dans le cadre de cette thèse pour aborder ce problème utilise des représentations condensées. Les représentations condensées permettent de réduire la charge de calcul d'extraction de connaissances, en évitant une (grosse) partie des calculs faits directement à partir des données (très coûteux) pour évaluer de motifs candidats.
Or réduit la charge de calcul liée à l'extraction de motifs en ne traitant pas ceux qui, après examen, s'avèrent redondants. Pour compenser, nous fournissons des méthodes pour régénérer les motifs redondants, s'ils ont une valeur dans un processus particulier (parfois ils n'en ont aucune et nous pouvons nous en passer sans aucune perte au niveau du résultat final).

Etat de l'art

Notions du domaine

Pour bien comprendre la suite du résumé, nous introduissons formellement les termes utilisés.

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

Figure 5.1: Une relation binaire.

Les travaux décrits dans cette thèse sont dédiés à la recherche de règles descriptives dans des jeux de données représentés sous la forme d'une relation binaire. Les jeux de données peuvent généralement être transformés en une telle relation binaire par un pré-traitement, spécifique à chaque jeu de données. Ainsi, les méthodes d'extraction de motifs peuvent s'appliquer à des problèmes très divers, et ce de façon simple et uniforme.
Figure 5.1 montre un exemple de relation binaire entre un ensemble d'objets $\{1, 2, 3, 4, 5, 6\}$ et un ensemble d'attributs (ou de propriétés) $\{A, B, C, D, E\}$, où les "x" indiquent les éléments de la relation.
Dans cette figure, nous pouvons lire que l'objet 2 est en relation avec l'attribut C, mais ne l'est pas avec l'attribut D. Plus couramment, nous dirons que l'objet 2 a la propriété C, mais n'a pas la propriété D.
Dans une application particulière, l'analyse du panier, les objets peuvent être des transactions, et les attributs — les produits qui peuvent être achetés lors d'une transaction.
L'extraction de règles d'association est fortement basée sur l'extraction d'ensembles fréquents [Agrawal 96] que nous définissons ci-dessous. Le support d'un ensemble de propriétés X est le ratio d'objets ayant toutes les propriétés de X. Par exemple, les propriétés de l'ensemble $\{A, B, C\}$ apparaissent simultanément dans les objets 1, 3 et 5, ce qui implique que le support de X vaut $3/6$. Un ensemble de propriétés est fréquent dans une relation binaire si toutes ses propriétés apparaissent simultanément dans un nombre suffisant d'objets. Un seuil σ, fixé par l'utilisateur, permet de trancher : si le support de X est supérieur ou égal à σ, l'ensemble X est dit fréquent. Dans la suite, nous utiliserons le terme tuple pour designer une ligne de la relation binaire (qui décrit un objet).
Les règles d'association dans ce contexte sont des expressions de type :

$$X \Rightarrow Y \quad [support, confiance]$$

où X et Y sont des ensembles d'attributs. Le support de la règle est définit comme le support de l'ensemble $X \cup Y$, et la confiance comme le ratio d'objets ayant les propriétés Y parmi tous les objets ayant les propriétés X.
Une propriété intéressante dit que tous les sous-ensembles d'un ensemble fréquent sont fréquents [Agrawal 96]. Cette propriété est utilisée dans l'algorithme appelé **APRIORI**

(voir ci-dessous), qui extrait une collection de tous les ensembles fréquents en temps proportionnel à la taille de cette collection, ce qui est considéré comme efficace dans les applications réelles visées à l'origine de cet algorithme (notamment l'analyse du panier, mentionnée ci-dessus).
Un traitement simple, décrit dans [Agrawal 96], permet de trouver toutes les règles d'association dont les supports dépassent le seuil σ et la confiance dépassent le seuil ρ (σ et ρ sont définis par l'utilisateur) à partir d'ensembles fréquents dont le seuil de fréquence est σ.

Algorithmes existants

Maintenant regardons les algorithmes récents qui calculent les ensembles fréquents. Nous avons constaté que tous ces algorithmes sont assez sensibles à la qualité de l'implementation et aux structures de données employées. Notamment, l'utilisation d'arbres de prefixes pour représenter des collections d'ensembles ou de tuples s'avèrent très efficace.

APRIORI

Le premier algorithme réalisant efficacement (dans des données transactionnelles) la tâche d'extraction de tous les ensembles fréquents s'appelle **APRIORI** [Agrawal 94, Mannila 94].
Cet algorithme emploie une stratégie d'élagage qui est à la fois simple, sûre et très efficace : les sur-ensembles d'un ensemble qui n'est pas fréquent ne peuvent pas être fréquents. **APRIORI** parcourt le treillis des ensembles (d'attributs) à partir des singletons et identifie, niveau par niveau, de plus grands ensembles fréquents jusqu'à ce que les plus grands ensembles fréquents soient trouvés. Un procédé basé sur une auto-jointure a été proposé dans [Agrawal 94] pour produire efficacement les candidats (ensembles dont tous les sous-ensembles sont fréquents). Une lecture linéaire du jeu de données (appelée ensuite une *passe*) est effectuée pour calculer les supports de tous les candidats d'une même taille et vérifier quels sont ceux qui sont fréquents. **APRIORI** traite un tuple à la fois pendant la passe.
En integrant dans cet algorithme la structure d'arbre de prefixe (inspirée de [Mueller 95]) pour stocker et accéder les candidats, l'efficacité d'**APRIORI** augmente nettement.

SAMPLING

L'algorithme **sampling** vise à réduire jusqu'à 1 le nombre de passes (concluant éventuellement avec une deuxième passe dans le pire des cas). D'abord, il essaye d'anticiper des ensembles qui sont fréquents dans le jeu de données, et puis réalise un passe complète pour vérifier si ces ensembles sont en effet fréquents. La vérification d'un item-

set consiste à calculer le support et à le comparer au seuil σ de support défini par l'utilisateur. L'anticipation se fait en calculant avec **APRIORI** la vraie collection d'ensembles fréquents, mais sur un échantillon de données, ce qui devrait permettre de l'accomplir plus vite que sur le jeu de données complet.
Il y a 2 types d'erreurs dues à l'échantillonnage :
① un ensemble anticipé fréquent qui s'avère infréquent (ce cas ne provoque pas un accroissement lourd de la charge de calcul),
② un ensemble anticipé infréquent qui s'avère fréquent (ce cas peut provoquer accroissement lourd de la charge de calcul). En cas d'une mauvaise conjoncture, une deuxième passe est nécessaire. La procédure de comptage de support est similaire à celle d'**APRIORI**.

DIC

L'essence de cette approche est de considérer une collection de candidats qui change dynamiquement pendant une passe. Observez qu'**APRIORI** considère un ensemble en tant que candidat quand tous ses sous-ensembles sont connus comme étant fréquents. **DIC** considère un ensemble fréquent si son compteur de support montre une valeur plus grande ou égale à σ, mais contrairement à **APRIORI**, **DIC** ne vérifie pas si cette condition est satisfaite seulement apès une passe entière, mais aussi pendant la passe, plus exactement aux quelques points prédéfinis du jeu de données.
Si un nouveau candidat est détecté (tous ses sous-ensembles immédiats se sont avérés fréquents), il sera considéré à partir de ce point de la passe par la procédure de comptage de support. Ceci peut mener à compter simultanément son support et les supports de ses sous-ensembles (à ce moment-là ces derniers sont connus comme d'étant fréquents). La procédure de comptage de support est similaire à celle d'**APRIORI**.

Max-Miner

Max-Miner [Bayardo, Jr. 98] est développé en se basant sur l'observation suivante. Si X est fréquent et $Y \subset X$, alors Y est fréquent, aussi. Par conséquent, Bayardo propose dans [Bayardo, Jr. 98] d'extraire seulement les ensembles fréquents maximaux, c'est-à-dire, les ensembles fréquents n'ayant aucun ensemble fréquent comme sur-ensemble stricte pour un σ donné.

FP-growth

FP-Growth considère un ensemble comme candidat seulement si son parent est fréquent. Le parent est déterminé comme le parent dans un arbre complet d'énumération d'ensembles. Ceci permet une approche de parcours en profondeur de l'espace de motifs.

Très importants sont les optimisations qui permettent de calculer les supports de candidats de manière très efficace en accord avec le parcours en profondeur. Le comptage de supports d'un grand nombre de candidats ne se combine pas avec l'approche de parcours en profondeur de l'espace de motifs, donc les optimisations trouvées sont très distinctes des autres approches. **FP-Growth** agrège différents tuples dans un seul, et peut incrémenter ainsi des compteurs de support par plus de 1 dans une opération simple. Plusieurs autres optimisations accompagnant cette idée la rendent tout à fait profitable en général. Notamment, la relation binaire est chargée en mémoire, sans quoi la charge d'entrées/sorties serait excessive. Egalement, les tuples de la relation sont agrégés, pour des mêmes tuples voire pour des mêmes parties de tuples (quand s'est possible).
Tree-Projection réalise différemment l'extraction d'ensembles fréquent en parcourant l'espace de recherche en profondeur, mais contrairement à **FP-Growth Tree-Projection** ne charge pas le jeu de données entier en mémoire.

CLOSE

Une idée différente est la quintessence de l'algorithme **CLOSE**. Cet algorithme incrémente les compteurs de support 1 par 1, mais il considère beaucoup moins de candidats qu'**APRIORI**, **DIC**, l'algorithme **sampling** et le **FP-Growth**[1]. **CLOSE** sait considérer moins de candidats sans perdre de l'information en détectant et en excluant la forme la plus évidente de redondance. L'algorithme en question extrait des ensembles fréquents dits fermés. Ce résultat est cependant complet dans le sens qu'il permet d'aboutir à la même connaissance que le résultat d'**APRIORI**, donc l'avantage est ici incontestable.
La différence par rapport à des ensembles fréquents (simples) est importante au niveau théorique (la définition est plus bas). Puisque l'algorithme extrait directement les ensembles précisement choisis (c'est-à-dire les ensembles fermés fréquents), il considère typiquement beaucoup moins de candidats qu'**APRIORI**, ce qui économise des ressources : le temps du processeur et la mémoire.
Dans le contexte d'un jeu de données r, un ensemble X est fermé si et seulement si X comprend chaque attribut qui est présent dans tous les objets de r ayant tous les attributs de X. En d'autres termes, pour chaque attribut A du schéma de la base de données qui n'est pas dans un ensemble fermé X, il y a un tuple t dans la base de données qui contient tous les attributs de X ($X \subseteq t$), et qui ne comprend pas l'attribut A.
L'extraction des ensembles fermés fréquents au lieu de tous les ensembles fréquents est un grand pas vers des techniques plus efficaces pour les données fortement corrélées. **CLOSE** extrait du jeu de données seulement des ensembles fréquent dûment choisis et leurs supports. Les ensembles fréquents restants et leurs supports peuvent être dérivés

[1] **Max-Miner** considère également beaucoup moins de candidats que ces algorithmes, mais il extrait seulement les ensembles fréquents maximaux, ce qui peut être vu comme perte importante d'informations sur tous les ensembles fréquents. Afin de calculer les supports de tous les ensembles fréquents, **Max-Miner** recule et les considère tous dans la procédure de comptage des supports.

des autres d'une façon correcte et complète. D'ailleurs, cette dérivation peut s'effectuer sans accès au jeu de données original (donc très efficacement), contrairement à l'approche de **Max-Miner**.

Compositions des idées

Une remarque intéressante sur les algorithmes présentés ci-dessus est que parmi les idées présentées certaines sont complémentaires et pourraient être combinées. La combinaison la plus prometteur est de considérer le parcours en profondeur de l'espace de motifs (agrégeant des tuples ce qui permet les opérations d'incrémentation de compteurs de supports) combiné avec l'extraction d'ensembles fréquents fermés, ce qui pourraient plus efficace que ces idées considérées séparément. Cette combinaison a été récemment considérée par Pei *et al.* dans [Pei 00], et l'algorithme résultant s'appelle **CLOSET**.

CLOSET

CLOSET [Pei 00] calcule les ensembles fermés fréquents et leurs supports correspondants en utilisant une technique similaire à **FP-Growth**. **CLOSET** parcourt l'espace de motifs en profondeur conformément à un arbre complet d'énumération d'ensembles. Un ensemble est considéré comme candidat si son parent est fréquent, mais la condition supplémentaire est que le support de l'enfant est différent (c'est-à-dire, strictement plus petit) de celui du parent.
En plus, l'algorithme calcule les fermetures des ensembles candidats.
Des optimisations analogues à celles utilisées par **FP-Growth** sont intégrées dans **CLOSET**, notamment celles visant à agréger beaucoup de tuples de la relation binaire quand ça peut se faire sans perdre de l'information pertinente (voir **FP-Growth**).
En résumé, **CLOSET** est une technique mariant deux idées les plus prometteuses des outils existants en extraction d'ensembles fréquents. Dans cette thèse, l'algorithme **CLOSET** est une référence pour les approches de parcours en profondeur.

Voie de recherche engagée

Nous avons observé que certaines des solutions présentées ci-dessus extraient moins de motifs que d'autres, néanmoins l'utilisateur peut dans certains cas être aussi satisfait du résultat qu'avant. Ceci vient du fait que l'utilisateur a un objectif et pour l'accomplir on peut lui fournir une collection de motifs équivalente, mais par exemple plus petite, et plus facile à extraire, à stocker ou à accéder. La collection équivalente va souvent être plus petite, de ce fait nous nous y réfèrons par le terme *représentation condensée.*
L'utilisation des représentations condensées dans l'extraction de connaissances a été suggérée dans [Mannila 97a] comme une direction prometteuse de recherche. De telles

représentations condensées peuvent être calculées avant la session interactive (avec utilisateur) du processus d'extraction. Dans ce cas, le but est d'améliorer la session interactive impliquant les humains.

Principaux Résultats de la Thèse

Nous avons étudié des représentations condensées décrites dans la littérature. Certains résultats présentés ci-dessus peuvent être assimilés à cette vision — l'extraction d'ensembles maximaux fréquents [Bayardo, Jr. 98] ou l'extraction d'ensembles fréquent fermés [Pasquier 99c].
Nous avons également proposé un cadre théorique pour décrire uniformément des représentations condensées.
Egalement, nous avons proposé des représentations condensées originales, ce que nous allons présenter en détails. La première s'appelle la condensation bornée par des règles fortes, désignée par SRBC[2] dans la suite, la deuxième — la condensation bornée par des règles disjonctives, désignée par DBC[3] dans la suite.

Représentation Condensée Bornée par des Règles Fortes

Dans cette section, nous considérons une représentation condensée originale basée sur les ensembles fréquents δ-libres (définition sera donnée plus loin). Nous la décrivons d'abord puisqu'elle a été proposée en premier [Boulicaut 00a, Boulicaut 00c, Boulicaut 03]. Néanmoins, nous décrivons une petite modification de cette représentation, afin de rendre les deux représentations condensées originales de cette thèse plus uniformes. Cette première représentation c'est SRBC.
L'intuition principale de la condensation bornée par des règles fortes est illustrée sur l'exemple suivant. Soient A, B, C, D des attributs binaires et soit r la relation binaire représentée comme Table 5.1. Supposons que nous sommes intéressés par le support de l'ensemble $\{A, B, C\}$ dans r. Si nous savons que la règle $\{A, B\} \Rightarrow \{C\}$ tient presque dans r (c'est-à-dire, quand A et B sont présents dans un tuple alors, excepté dans quelques cas dans r, C est également présents dans ce tuple) alors nous pouvons approximer le support de l'ensemble $\{A, B, C\}$ par le support de $\{A, B\}$. Dans Table 5.1, la règle $\{A, B\} \Rightarrow \{C\}$ a seulement une exception. Ainsi, nous pouvons utiliser le support de $\{A, B\}$ comme une valeur approchée pour le support de $\{A, B, C\}$. D'ailleurs, nous pouvons rapprocher le support de n'importe quel ensemble X tel que $\{A, B, C\} \subseteq X$ par le support de $X \setminus \{C\}$ parce que la règle $X \setminus \{C\} \Rightarrow \{C\}$ tient également avec au plus le même nombre d'exceptions (c'est-à-dire, jamais plus grand). Par exemple, le support de $X = \{A, B, C, D\}$ peut être rapproché par le support de $\{A, B, D\}$ puisque la règle

[2] Ang. strong-rule-bordered condensation
[3] Ang. disjunction-bordered condensation

$\{A, B, D\} \Rightarrow \{C\}$ ne peut pas avoir plus d'exceptions que $\{A, B\} \Rightarrow \{C\}$. D'ailleurs, le support de $\{A, B, D\}$ n'a pas besoin d'être connu exactement, mais peut également être rapproché. Par exemple, la règle $\{A, D\} \Rightarrow \{B\}$ tient dans Table 5.1 avec une exception, ainsi le support de $\{A, D\}$ peut être utilisé comme approximation du support de $\{A, B, D\}$ et puis également du support de $\{A, D, C, D\}$.

A	B	C	D
x	x		
x			x
	x	x	x
x		x	x
x	x		x
x	x	x	
x			
x			
	x		
	x		
		x	
		x	
			x
			x
x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x

Table 5.1: Une table relationnelle basée sur quatre attributs binaires ("x" dénote que l'attribut décrit l'objet).

Un ensemble X est *δ-libre* si et seulement si aucune règle construite avec les attributs de X ne tient pas dans la relation binaire avec au plus δ exceptions.

Comme δ est censé être plutôt petit, intuitivement, un ensemble δ-libre est un ensemble d'attributs tel que ses sous-ensembles (vus comme des conjonctions des propriétés) ne sont reliés par aucune corrélation positive très forte.
Une des propriétés les plus intéressantes des ensembles δ-libres est l'anti-monotonicité par rapport à l'inclusion ensembliste.
L'anti-monotonicité a été identifiée comme une propriété principale pour l'extraction efficace de motifs [Mannila 97b, Ng 98], puisque c'est la base formelle d'un critère d'élagage sûr. En effet, les algorithmes efficaces d'extraction d'ensembles fréquents, comme **APRIORI** et **DIC** (présentés précédemment), se servent de la propriété (anti-monotone) "l'ensemble est fréquent" pour l'élagage de l'espace de recherche.
Les ensembles δ-libres (et leurs supports) peuvent être employés pour répondre à des requêtes de support de n'import quel ensemble d'attributs, répondant avec une erreur

bornée, et cette borne reste proportionnelle à la valeur de δ employée. De même, les ensembles δ-libres *fréquents* et les ensembles *fréquents minimaux qui ne sont pas δ-libres* (et les supports associés) peuvent être employés pour répondre à des requêtes de support de n'import quel ensemble d'attributs, en répondant presque aussi bien que la collection d'ensembles fréquents. La borne de l'erreur reste la même pour les ensembles fréquents (ceci en général est ce que nous cherchons) que dans le cas d'approximation en utilisant tous les ensembles δ-libres.
Dans la pratique, le calcul de la collection entière des ensembles δ-libres (et leurs supports) est souvent intractable, mais les ensembles δ-libres *fréquents* peuvent être extraits pour des seuils de supports même plus bas que pour les ensembles fréquents. Ceci apparaît claire si nous remarquons que la collection d'ensembles de la condensation bornée par des règles fortes est une sous-collection de tous les ensembles fréquents.

SRBC est composé simplement de la collection de tous les ensembles δ-libres fréquents et des ensembles fréquents minimaux qui ne sont pas δ-libres. A chaque ensemble nous ajoutons l'information correspondante à son support.

Figure 5.2 met en évidence, à l'aide des axes à échelle logarithmique, la différence observée des tailles des différents représentations. A gauche, nous avons affiché la taille des représentations en octets (supposant que la représentation est sauvegardée en explicitant tous les ensembles et leurs supports, 4 octets par élément d'un ensemble et 4 octets par support). A droite, nous avons affiché seulement le nombre d'ensembles de la représentation.
Des valeurs plus petites correspondent aux représentations plus condensées.

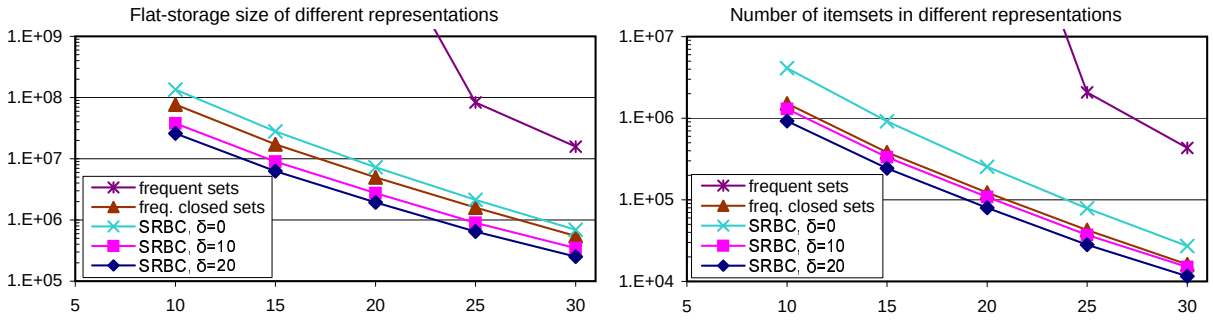

Figure 5.2: Tailles de différentes représentations.

Nous avons observé, comme prévu, qu'en augmentant δ le niveau de condensation est plus élévé. Entre $\delta = 0$ et $\delta = 20$ nous avons observé une réduction de taille d'environ 4 fois, pour les deux mesures utilisées.
Une comparaison exprérimentale intéressante est entre SRBC et les ensembles fermés fréquents. La taille de la représentation basée sur les ensembles fermés fréquents est comparable à SRBC pour $\delta = 10$ en utilisant comme mesure le nombre d'ensembles, ou comparable à SRBC avec un δ entre 0 et 10 pour la taille en octets.
Pour les deux mesures, la condensation de SRBC pour $\delta = 20$ est plus élévé que celle des

ensembles fermés fréquents.
La comparaison exprérimentale avec les ensembles fréquents était moins aventureuse — nous savions *a priori* que SRBC n'est jamais plus grand.
Comme attendu, nous observons une condensation plus élevée du SRBC que des ensembles fréquents. La différence est très marquée et s'étend d'une à plus de cinq ordres de grandeur.

L'algorithme **Min-Ex** proposé dans cette thèse pour extraire SRBC peut être vu comme instance de l'algorithme de recherche par niveau présenté dans [Mannila 97b]. Il parcourt le treillis d'ensembles niveau par niveau, à partir de l'ensemble vide et s'arrête au niveau des ensembles δ-libres fréquents de taille maximale.
L'anti-monotonicité de δ-liberté par rapport à l'inclusion ensembliste est employée pour l'élagage. En fait, la propriété combinée "fréquent et δ-libre" est employée. Elle est aussi anti-monotone.
Nous avons mené des expériences qui montrent que sur les données fortement corrélées la condensation bornée par des règles fortes est calculée très efficacement, une fois comparée à l'extraction d'ensembles fréquents ou même comparée à l'extraction d'ensembles fermés fréquents.
Les techniques mentionnées ci-dessus présentent les avantages importants sur les données fortement corrélées, mais si nous considérons les données peu corrélées, nous ne pouvons espérer avoir beaucoup de règles exactes ou presque exactes qui tiennent, et ainsi toutes les techniques sont susceptibles d'être peu avantageuses. Nous pouvons même nous attendre à ce que, sur des données peu corrélées, ces techniques soient un peu plus lentes que l'extraction des ensembles fréquents sans élagage additionnel, puisqu'elles ne peuvent pas tirer profit des réductions importantes de l'espace de recherche. Ceci est dû au calcul supplémentaire nécessaire pour détecter (improductivement) des règles δ-fortes.

Représentation Condensée Bornée par des Règles Disjonctives

Dans cette section, nous décrivons une représentation condensée originale basée sur les ensembles appelés disjonction-libres (la définition suit). Cette deuxième représentation originale, appelée représentation condensée bornée par des règles disjonctives, contient la même quantité d'information que la collection de tous les ensembles fréquents.
Une étude préliminaire au sujet de cette représentation est parue comme [Bykowski 01] et la recherche complète à ce sujet est décrite dans [Bykowski 03].
Comme la condensation bornée par des règles fortes (SRBC), la condensation bornée par des règles disjonctives (DBC) se compose d'une sous-collection des ensembles fréquents et peut être extraite plus efficacement que tous les ensembles fréquents. Cette collection est sans perte d'information. DBC peut donc être employé pour réduire considérablement le temps de d'extraction et l'espace mémoire utilisé tout en garantissant au moins la même

A	B	C	D
x	x	x	
	x	x	x
x			x
			x
		x	
x		x	x
	x	x	
x	x	x	

Figure 5.3: Jeu de données pour illustrer les définitions.

qualité du résultat extrait que dans des processus d'extraction impliquant des ensembles fréquents voire ensembles fermés fréquents.
La réduction de taille de DBC (par rapport à la collection de tous les ensembles fréquents) est due à une propriété qui lie des supports de quelques ensembles par des équations. Cette propriété est basée sur des expressions appelées les règles disjonctives simples. La forme générale de une telle règle est $A_1 \wedge A_2 \wedge \ldots \wedge A_{n-2} \Rightarrow A_{n-1} \vee A_n$, où les A_i représentent des différents attributs (à l'exception des 2 derniers attributs, c'est-à-dire, A_{n-1} et A_n, que nous autorisons à être le même attribut). Cette règle déclare que si $A_1, \ldots, A_{n-2}$ sont en relation avec un tuple, alors A_{n-1} ou A_n sont aussi en relation avec le même tuple. La règle peut tenir ou peut ne pas tenir dans un tuple. Le dernier cas se présente quand ni A_{n-1} ni A_n n'est en relation avec le tuple bien que tous les $A_1, A_2, \ldots, A_{n-2}$ soient en relation avec ce même tuple.
Considérons la table r' représentée dans Figure 5.3. La règle $B \wedge C \Rightarrow A \vee D$ tient par exemple dans le deuxième tuple (les deux, l'antécédent et le conséquent sont vrais) et dans le troisième tuple (l'antécédent est faux), mais ne tient pas dans l'avant dernier. La réduction de taille de DBC est basée sur les règles qui tiennent dans tous les tuples de la table, comme par exemple $A \Rightarrow C \vee D$ dans r'.

Un ensemble contenant les attributs qui peuvent être utilisés pour former une règle disjonctive simple tenant dans toutes les tuples de la relation s'appelle un ensemble non-disjonction-libre. Les autres ensembles s'appellent disjonction-libres. Par exemple $\{A, C, D\}$ et $\{A, B, C, D\}$ ne sont pas disjonction-libres, à cause de la règle $A \Rightarrow C \vee D$.

DBC est composé simplement de la collection de tous les ensembles disjonction-libres fréquents et des ensembles fréquents minimaux qui ne sont pas disjonction-libres. A chaque ensemble nous ajoutons l'information correspondante à son support.

Les tailles de différentes représentations pour plusieurs seuils de support sont indiquées sur la Figure 5.4 (certaines axes sont à échelle logarithmique). Nous avons utilisé les mêmes mesures que dans le cas d'expériences avec la taille de SRBC (voir plus haut).
Les valeurs plus petites correspondent aux représentations plus condensées.

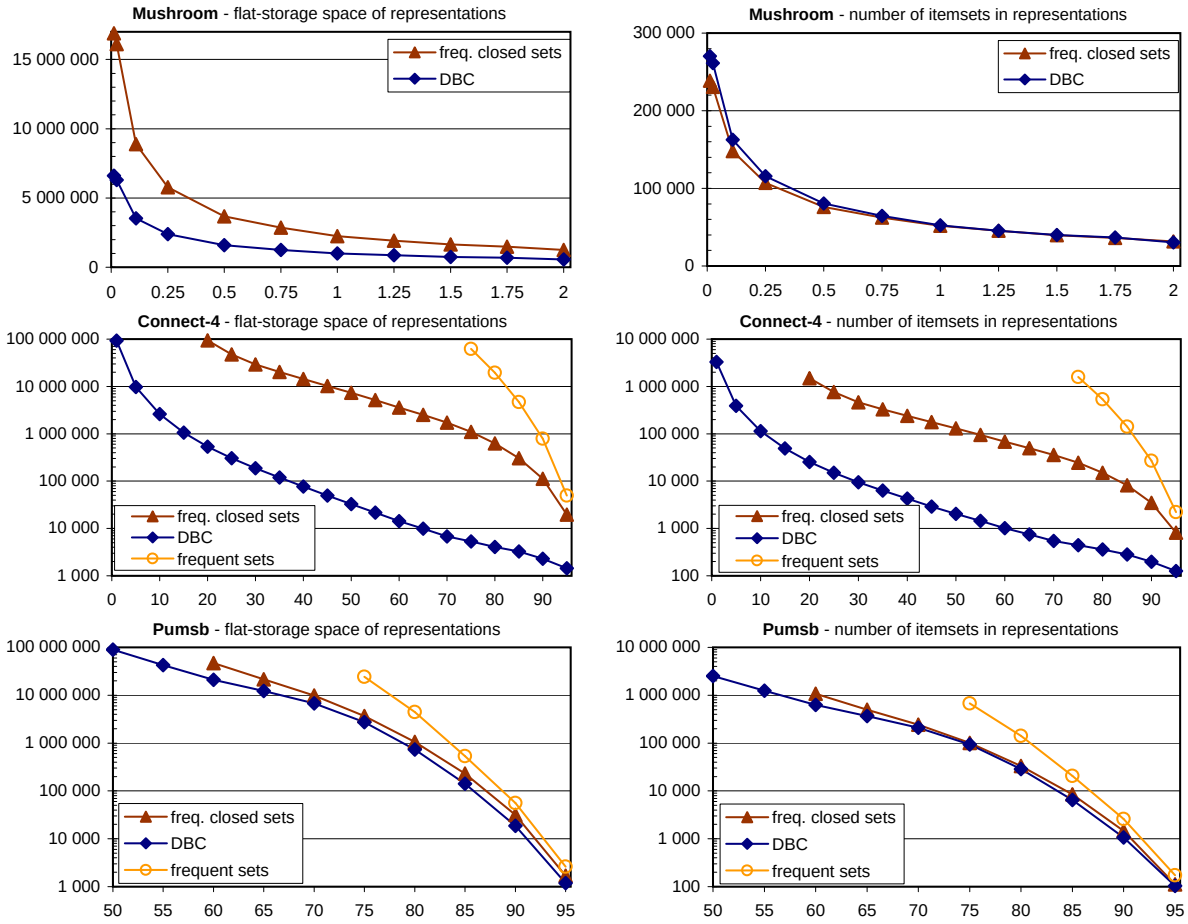

Figure 5.4: Tailles de différentes représentations sur trois jeux de données.

Sur Figure 5.4 à gauche, nous pouvons voir que la représentation DBC est plus petite que la représentation à base d'ensembles fermés fréquents dans toutes les expériences quand nous considérons leurs volumes en octets. Le même tient à peu d'exceptions près si nous comptons le nombre d'ensembles composant chaque représentation (Figure 5.4, à droite). Pour la comparaison de la taille de DBC avec celle de la collection de tous les ensembles fréquents, nous savions *a priori* que le DBC est toujours plus petit ou égal dans la taille. Dans les expériences présentées dans la Figure 5.4, la différence est très importante et s'étend d'une à quatre ordres de grandeur.

La propriété la plus intéressante de DBC est que cette collection peut être extraite très efficacement et elle est suffisante pour calculer tous les ensembles fréquents et leurs supports.

La propriété "est disjonction-libre" est anti-monotone, ce qui permet (comme dans le cas des ensembles δ-libres) d'utiliser un algorithme dérivé de l'algorithme générique par niveau présenté dans [Mannila 97b]. En fait, la propriété combinée "fréquent et disjonction-libre" est employée. Elle est également anti-monotone. L'algorithme résultant s'appelle **HLin-Ex**.

Nous avons également proposé un algorithme réalisant l'extraction de cette représentation en parcourant en profondeur l'espace de motifs. Il marie des avantages de **Max-Miner**,

de **FP-Growth** et de **Tree-Projection**. L'algorithme résultant, **VLin-Ex**, agrège également différents tuples en un seul, et peut incrémenter ainsi des compteurs de support par plus de 1 en une opération simple. Dans le cas de **VLin-Ex**, la relation binaire est chargée en mémoire et compactée en début d'exécution.
Dans presque toutes les expériences sur trois jeux de données différents choisis aveuglement parmi des jeux de donnés de référence fortement corrélés, l'extraction de DBC est sensiblement plus efficace que l'extraction des ensembles fermés fréquents correspondants et que l'extraction des ensembles fréquents. Pour les cas très difficiles, DBC peut être extrait aux seuils de support inférieurs que les deux autres collections en utilisant les mêmes ressources.
Comme en cas de SRBC, DBC présente très peu ou pas d'avantages si nous considérons les jeux de données très peu corrélés. L'extraction peut même être un peu plus lente.

Principales Applications

La dérivation des ensembles fréquents à partir de SRBC ou de DBC (les représentations décrites ci-dessus) est l'application la plus fondamentale de ces représentations condensées. Puisque ces représentations condensées représentent souvent des collections d'ensembles fréquents "intractablement" grandes, la dérivation de tous les ensembles fréquents n'est pas toujours possible. Dans ce cas, ceci ne signifie pas que nous ne pouvons pas procéder aux tâches d'extraction. L'extraction des ensembles fréquents est souvent suivie d'autres tâches d'extraction de connaissances qui se servent des ensembles fréquents. Il est courant d'employer une sous-collection bien définie des ensembles fréquents et d'abandonner les autres. Dans ces cas-ci, la dérivation d'ensembles fréquents peut être raffinée pour régénérer de telles sous-collections, en court-circuitant des collections, souvent volumineuses, de tous les ensembles fréquents. Par exemple, les procédures de régénération de tous les ensembles fréquents (**Regen-Freq**, **Regen-Approx-Freq**, décrite plus loin) peuvent être facilement modifiées pour extraire les ensembles fréquents sous des contraintes (l'extraction des ensembles fréquents sous des contraintes a été largement étudiée, voir, par exemple, [Srikant 97, Ng 98, Boulicaut 00d]).
Les applications finales incluent toutes les applications des ensembles fréquents, telles que la dérivation de règles d'association (avec une réserve sur la tractabilité équivalente à celle sur les ensembles fréquents), dérivation de règles d'association sous contraintes [Srikant 97, Ng 98, Boulicaut 00d], calcul des couvertures de règles d'association, extraction des règles d'association avec des négations ou leurs couvertures, la dérivation des supports des formules généralisées arbitraires (voir [Bykowski 02b]), les techniques de classification [Das 98, Han 98] et ainsi de suite.
Nous ne discutons pas toutes les applications. Nous nous concentrons sur les principes fondamentaux qui peuvent apporter des bénéfices à ces applications.

Le calcul de toutes les règles d'association à partir des représentations condensées proposées est simple si l'on régénère tous les ensembles fréquents puis que nous appliquons les algorithmes classiques de dérivation de ces règles à partir des ensembles fréquents. Le résultat mérite quand-même une évaluation de qualité si nous nous servions des représentations avec perte d'information (et donc, potentiellement, perte de qualité). Pour ce choix particulier, nous avons besoin de faire attention aux erreurs sur les fonctions d'évaluation (par exemple, le support et la confiance) propagées à partir des surestimations des supports des ensembles fréquents dues à SRBC avec un $\delta > 0$.
Un avantage éprouvé est celui lié au calcul des couvertures de règles d'association. Ce calcul peut se faire sans extraire tous les ensembles fréquents (afin de court-circuiter des collections volumineuses d'ensembles fréquents). Dans [Bastide 00a] et dans [Zaki 00] respectivement Bastide *et al.* et Zaki ont décrit des méthodes pour calculer des couvertures de règles d'association. Ces deux méthodes se servent des ensembles fréquents fermés pour calculer ces couvertures. En réalité, les deux se servent aussi des ensembles fréquents 0-libres (l'utilisation des ensembles fréquents fermés est explicitement énoncée dans les deux articles, mais le besoin des ensembles fréquents 0-libres n'est énoncé qu'à travers des propriétés qui peuvent se réduire à la définition des ensembles 0-libres).
En conséquence, nous nous sommes pariculièrement attachés à montrer comment convertir DBC en deux collections : les ensembles fréquents fermés et les ensembles fréquents 0-libres (pour éventuellement en dériver une couverture de règles d'association). Ceci peut être fait même lorsque la collection de tous les ensembles fréquents est "intractablement" grande. En ce qui concerne l'utilisation de SRBC pour générer des couvertures de règles, la variante sans perte de SRBC est déjà constituée des ensembles fréquents 0-libres, donc seule la régénération des ensembles fréquents fermés est à envisager. Ce procédé peut facilement se déduire du procédé convertissant DBC en ensembles fréquents fermés et ensembles fréquents 0-libres, et n'est donc pas spécifiquement détaillé ni expérimenté.
Des applications très spécifiques, profitant mieux des tailles plus petites des représentations condensées ont été déjà proposées ou sont en cours de développement (voir [Bykowski 02a, Bykowski 00, Bykowski 02b]).

Régénération des ensembles fréquents à partir de DBC

L'algorithme de régénération des ensembles fréquents à partir de DBC s'appelle **Regen-Freq**. Comme énoncé précédemment, il n'a besoin que de DBC (en particulier, il n'a pas besoin du jeu de données, dont la lecture combinée avec un comptage de supports est normalement très coûteuse). Ceci est dû au fait que DBC est une représentation des ensembles fréquents sans perte d'information.
Comme DBC est une collection "anti-monotone" (c'est-à-dire, le critère d'appartenance d'un ensemble à DBC est anti-monotone) et composée d'une sous-collection d'ensembles fréquents, un certain nombre d'ensembles fréquents à régénérer y est déjà. Les autres

ensembles fréquents peuvent être dérivés à partir de DBC. Nous avons proposé d'effectuer cette dérivation niveau par niveau, en commençant par les ensembles fréquents les plus petits et en passant aux ensembles fréquents de taille $i+1$ une fois que tous les ensembles de taille i ont été calculés.
A l'initialisation, nous considérons tous les ensembles faisant partie de DBC comme une collection d'ensembles fréquents de départ. Ensuite, les ensembles qui ne sont pas dans DBC sont dérivés à partir des ensembles fréquents déjà connus en utilisant une relation entre les supports des ensembles dérivée des règles disjonctives qui tiennent dans tous les tuples (voir la définition des ensembles disjonction-libres plus haut). Le processus s'effectue itérativement pour tous les ensembles (qui sont à dériver) de même taille, jusqu'à ce qu'il n'y aie plus d'ensembles fréquents sur un niveau. A ce moment, le programme termine.

Régénération des ensembles fréquents à partir de SRBC

La régénération des ensembles fréquents à partir de SRBC nécessite une attention particulière due au fait que SRBC n'est pas une représentation sans perte d'information (quand δ est différent de 0). Nous avons déterminé (en fonction du paramètre δ) une limite théorique d'erreur du support lors de dérivation utilisant les ensembles δ-libres. Cette limite d'erreur croit linéairement avec la valeur de δ. Mais nous sommes allés plus loin et nous avons mené des expériences spécifiques pour évaluer cette perte d'information en pratique et la comparer à la limite théorique. Ainsi, nous avons observé que pour les données réelles et un δ petit (de l'ordre de quelques tuples), non seulement l'erreur pratique est petite (ce qui soutient le résultat théorique disant que l'erreur est linéaire avec δ), mais elle est d'un ordre de grandeur plus petit que la limite théorique correspondante.
Par conséquent, la représentation peut être utilisée même pour matérialiser tous les ensembles fréquents (et leur supports), sans craindre une imprécision majeure. L'algorithme réalisant cette dérivation s'appelle **Regen-Approx-Freq**. Il fonctionne de façon similaire à **Regen-Freq** (algorithme par niveau, utilisant les ensembles fréquents, jusqu'à une certaine taille tous déjà connus, pour dériver des ensembles fréquents de taille supérieure). Cet algorithme calcule les supports de tous les ensembles fréquents en utilisant tous les ensembles contenus dans SRBC, ce qui en général améliore la précision du résultat, comparé à celui déduit seulement des ensembles δ-libres.
Regen-Approx-Freq n'a pas besoin du jeu de données de départ, dont la lecture combinée avec un comptage de supports est normalement très coûteuse.

Régénération des règles d'association à partir de SRBC

Le calcul de toutes les règles d'association à partir des représentations condensées proposées dans cette thèse est direct en régénérant tous les ensembles fréquents, puis

en employant l'algorithme standard de dérivation de règles à partir des ensembles fréquents [Agrawal 94], à moins que nous nous servons des représentations avec perte d'information, notamment de SRBC.
C'est dans ce cas que nous avons cherché à quantifier l'impact de l'approximation les supports des ensembles fréquents quand les supports et les confiances des règles d'association sont dérivés de SRBC. Nous avons donné des limites pour l'erreur faite sur ces mesures des règles d'association et comme les erreurs sur supports des ensembles fréquents étaient très petites en pratique, de petites erreurs de supports et de confiances des règles suivent.

Régénération des ensembles fréquents fermés à partir de DBC

Un résultat important d'étude des ensembles disjonction-libres est la comparaison de différentes qualités de ces ensembles avec les qualités d'ensembles fermés. A travers des expériences multiples, nous avons montré que l'extraction de DBC est quasiment toujours plus rapide que celle des ensembles fréquents fermés correspondants et que les tailles de ces représentations respectives sont quasiment toujours à l'avantage de DBC. Mais, la notion des ensembles fermés est issue d'un domaine avec des bases théoriques développées pendant des décennies (cf. [Ore 44, Wille 82]). Ainsi, nous avons étudié DBC comme une représentation condensée intéressante des ensembles fermés fréquents. Pour cela, nous avons proposé un cadre d'utilisation de DBC qui permet de réaliser des avantages importants par rapport à l'utilisation directe des ensembles fermés fréquents. Parmi eux, nous pouvons mentionner la capacité de dériver les ensembles fermés fréquents partir de DBC dont la durée ajoutée à celle d'extraction de DBC reste inférieure à la durée d'extraction des ensembles fermés fréquents par les algorithmes correspondants. Ainsi, nous calculons des ensembles fermés fréquents plus efficacement qu'avant. Un autre avantage qui peut être remarqué est la capacité de DBC à produire les ensembles fermés fréquents et les ensembles 0-libres fréquents correspondants, ce qui facilite beaucoup le calcul des couvertures de règles d'association (par exemple, les couvertures proposées dans [Bastide 00a, Zaki 00] se servent toutes les deux de ces ensembles). L'intérêt de calculer des couvertures de règles d'association est très important, les couvertures présentent à l'expert un résumé compact des collections (habituellement de très grande taille) de règles d'association.
Nous pouvons ainsi dire que DBC donne fonctionnellement accès aux mêmes résultats (connaissances) que les ensembles fermés fréquents, mais nous pouvons le faire plus efficacement avec DBC qu'en calculant directement les ensembles fermés fréquents. On peut également dire qu'avec de mêmes ressources, l'utilisateur peut analyser plus des phénomènes, le surplus correspond aux phénomènes les plus difficiles à extraire (à cause du support qui est plus petit).

Régénération de SRBC à partir de DBC et des ensembles fréquents fermés à partir de SRBC

SRBC avec $\delta = 0$ est une représentation sans perte d'information. Sa reconstruction à partir de DBC (une autre représentation sans perte d'information) devrait être possible. Egalement, la régénération des ensembles fermés fréquents devrait être possible à partir de SRBC.
En effet, elles sont toutes les deux possibles grâce à l'algorithme de conversion de DBC en ensembles fermés fréquents qui consiste en une composition de ces deux algorithmes. Les temps de conversion élémentaires résultant sont certainement inférieurs à la conversion combinée DBC$\rightarrow$ ensembles fermés fréquents.

Pistes vers la découverte des règles d'association avec des négations

Il arrive parfois que nous voulons trouver des associations entre les attributs positifs (la présence d'une propriété) et les attributs négatif (des attributs dont la présence traduit l'absence d'une propriété "positive"). Dans la suite, on emploie le terme *règles généralisées* pour de telles règles. Le calcul de règles généralisées fréquentes s'avère intractable dans de nombreux cas pratiques, pour les mêmes seuils de support que ceux accessibles pour la découverte des règles d'association "standard".
Lors de l'extraction des règles généralisées, l'étape coûteuse concerne le calcul des ensembles généralisés fréquents desquels les règles généralisées sont dérivées. C'est pourquoi nous nous sommes intéressés à cette étape de l'extraction.
Nous avons étudié des possibilités pour un calcul tractable des ensembles fréquents généralisés et leur post-traitement en règles d'association généralisées.
Nous avons essayé d'utiliser des contraintes adéquates pour élaguer au plus tôt des règles généralisées surabondantes (telles que les règles faisant intervenir uniquement des attributs négatifs) ou l'utilisation de représentations condensées. Les deux approches permettent de gagner en tractabilité de calcul des règles généralisées pour des seuils de supports très petits.

Conclusion, Travaux Futures et Perspectives

Conclusion

L'extraction de motifs est une technique importante d'extraction de connaissances quand nous disposons d'une large base de données et que nous voulons améliorer notre compréhension du système générant les observations stockées dans la base.
L'extraction des ensembles fréquents (un type de base de motifs descriptifs) a été effectuée avec succès sur quelques jeux de données. Malgré un potentiel relatif des ensembles

fréquents pour comprendre d'autres jeux de données, plusieurs d'entre eux nécessitent des calculs trop lourds ou non tractables. Entre autres raisons, l'échec est dû à la corrélation forte des attributs et à l'explosion résultante du nombre d'ensembles fréquents.
Dans cette thèse, nous avons proposé d'éviter une partie de la charge liée au calcul d'ensembles fréquents. Pour ce faire, nous avons proposé des méthodes basées sur l'exploitation de la redondance observée dans les collections d'ensembles fréquents. D'abord, on a observé que quelques approches du passé peuvent être ramenées à cette idée.
Puis, nous avons étudié en détails deux types de redondances ce qui a parmis de proposer deux principales représentations condensées des collections d'ensembles fréquents. En pratique, elles sont beaucoup plus petites que les collections d'ensembles fréquents qu'elles représentent et plus efficaces à calculer.
Nous observons également ces avantages une fois les nouvelle représentations sont comparées aux ensembles fermés fréquents, probablement la seule structure proposée dans le passé à laquelle nous pouvons nous comparer vraiment.
Nous avons fourni des principes théoriques détaillés, des algorithmes et des preuves expérimentales sur quelques jeux de données fortement corrélés typiques pouvant bénéficier des résultats de la thèse. Les méthodes résultantes permettent en général d'élargir l'applicabilité de la découverte des connaissances basée sur les ensembles fréquents, pour réduire des seuils de supports (approfondir l'analyse de données) avec les mêmes ressources ou pour accroître l'applicabilité des méthodes connues à des donnés plus difficiles.
Nous avons également proposé un cadre pour décrire uniformément des représentations condensées de motifs.
Nous avons fourni des outils pour tirer profit des représentations condensées même pour des techniques standard basées sur des ensembles fréquents, des ensembles fermés fréquents ou des ensembles 0-libres fréquents. En effet, les techniques basées sur des représentations condensées proposées dans la thèse peuvent simplement être utilisées pour produire plus rapidement de motifs dont les méthodes mentionnées ci-dessus ont besoin (c'est-à-dire les ensembles fréquents, les ensembles fermés fréquents ou les ensembles 0-libres fréquents).
Finalement, nous avons considéré une tâche d'extraction proposée dans le passé, mais qui n'est pas tracktable en général, l'extraction de règles d'association avec des négations. On propose des pistes pour pouvoir accomplir cette tâche et l'une d'elles consiste à utiliser une des représentations condensées originales proposées dans la thèse.

Les résultats de la thèse sont centrés sur amélioration des dispositifs pour les autres techniques basées sur le calcul d'ensembles fréquents, déjà bien connues. Ces résultats ont déjà été la base d'autres travaux, réalisés indépendamment, par ex. [Calders 02].
En plus, les prototypes concrets réalisés en relation avec la thèse ont servi à valider la viabilité d'une nouvelle direction de recherche entreprise et à explorer des jeux de données

réels. La nouvelle direction de recherche entreprise, mentionné ci-dessus a été pré-étudiée avec **Min-Ex** pour voir si elle est viable, avant d'entamer des recherches à échelle normale (cette recherche est décrite dans [Abdennadher 00]). La nouvelle direction de recherche en question est la génération automatique des règles de propagation (des règles logiques utilisées dans le domaine de programmation par contraintes pour résoudre des groupes de contraintes).
De nombreuses publications montrent l'utilisation pratique des prototypes pour explorer des données réelles [Boulicaut 01a, Robardet 02] (analyse de données médicales) et [Bykowski 00] (extraction de connaissances dans des logs Web).

Travaux Futurs

L'avancée fondamentale due à l'utilisation de représentations condensées pourra être pousuivie par des travaux dédiés à des usages spécifiques des ensembles fréquents.
Une application semble essentielle dans un futur immédiat. Il s'agit d'utilisation efficace de contraintes, dont les études précédentes incluent [Srikant 97, Ng 98, Boulicaut 00d] combinée avec l'utilisation des représentations condensées.
Un guidage d'un système d'extraction de motifs vers des motifs importants en utilisant des contraintes permet d'analyser des données dans les contextes où le nombre total de motifs fréquents est intractable à calculer ou post-traiter.
En combinant l'idée des contraintes avec l'utilisation de représentations condensées semble être très prometteur. En effet, nous avons observé que la combinaison est efficace lorsque l'on a étudié l'extraction d'une représentation synthétique des collections de règles d'association (cf [Bykowski 02a]). Un travail de formalisation est ici nécessaire.

Finalement, d'autres motifs (par ex. séquentiels) pourraient directement profiter du cadre des représentations condensées. Peut-être, pourrions-nous même l'appliquer aux règles d'association. Est-ce qu'il serait possible de filtrer des règles redondantes autres que par les méthodes concurrentes, rendant de résultat plus facile à analyser ou extraire? Ce point nécessite d'être approfondi et nous allons y atteler dans le futur.

Une perspective à long terme

A l'horizon, nous pouvons imaginer un système de gestion des données permettant, parmi autres choses, de calculer des différentes classes de motifs en utilisant des fonctions d'évaluation spécifiées par un utilisateur.
De tels systèmes s'appellent des bases de données *inductives* (cf par ex. [Mannila 97a]). Ils pourraient bénéficier de la matérialisation d'une certaine information redondante (d'une manière semblable comme les systèmes de gestion de bases de données relationnelles bénéficient des indexes). Une telle information redondante serait recueillie avant une session

interactive d'extraction de connaissances, pour améliorer le comportement pendant la session interactive dans la plupart des cas.

Etant donné que les motifs sont en général assez difficiles à manipuler et exigent un effort élevé pour les calculer, un bon investissement en un calcul d'information redondante (comme des index dans le cas d'une base de données relationnelles) permettrait une accélération des réponses durant la session interactive. Typiquement, nous serions intéressés par une collection de statistiques calculées telle qu'elle peut être entièrement substituée aux données extensionnelles pour éviter des lectures répétées du jeu de données lui-même pour évaluer les motifs candidats. D'autre part, des statistiques suffisantes pour certaines requêtes pourraient être trop proches du résultat de ces requêtes pour bénéficier d'autres requêtes. Par conséquent, un bon compromis entre la généralité (flexibilité) des sommaires et leur contribution à l'efficacité doit être trouvée.

Les représentations condensées proposées dans cette thèse sont suffisamment flexibles pour accomplir chaque tâche qui se sert des ensembles fréquents en tant que seule information au départ (car nous avons montré comment régénérer tous les ensembles fréquents). En même temps, les représentations condensées proposées sont efficaces pour obtenir les ensembles fréquents, et donc même en se servant d'elles pour obtenir efficacement les ensembles fréquents (ce qui peut dans certains cas être même court-circuité pour mieux bénéficier des représentations condensées), le processus d'extraction s'améliore.

Nous croyons que les représentations condensées proposées dans la thèse seront des composants des systèmes opérationnels de bases de données inductives.