Alexandria University - Faculty of Engineering

A NEW ALGORITHM FOR INCREMENTAL MINING OF CONSTRAINED ASSOCIATION RULES

A thesis submitted to the

Department of Computer Science and Automatic Control

in partial fulfillment of the requirements for the degree of

Master of Science

By

Eng. Ahmed Medhat Ayad

Alexandria University - 2000

To Nour,

The new shining light of my life.

We certify that we have read this thesis and that, in our opinion, it is fully adequate, in scope and quality, as a dissertation for the degree of Master of Science.

Examination Committee:

1. ***Prof. Dr. Abd El-Moneim Youssef Belal***
Faculty of Engineering,
Cairo University.

2. ***Prof. Dr. Khalil Mohamed Ahmed***
Faculty of Engineering,
Alexandria University.

3. ***Prof. Dr. Nagwa Moustafa El-Makky***
Faculty of Engineering,
Alexandria University.

For the Faculty Council:

Prof. Dr. Hassan Farag
Vice Dean for Graduate Studies and Research
Faculty of Engineering, Alexandria University.

C.V.

Name: Ahmed Medhat Ayad

Date of birth: 14-2-1975

Place of Birth: Alexandria.

Nationality: Egyptian

Home Address:

17b. Syria St.,

Roushdy, Alexandria,

Egypt.

Office Address:

Department of Computer Science and Automatic Control,

Faculty of Engineering,

Alexandria University,

Alexandria, Egypt.

Current profession:

Teaching Assistant at:

Department of Computer Science and Automatic Control,

Faculty of Engineering,

Alexandria University.

Educational Record:

- The General Secondary Certificate (Al-Thanawya Al Aama), 1992 , Egypt.
- B.Sc.: Department of Computer Science and Automatic Control, Faculty of Engineering, Alexandria University, 1997

Supervisors

Prof. Dr. Nagwa M. El-Makky

Associate professor at:

Department of Computer Science and Automatic Control,

Faculty of Engineering,

Alexandria University,

Egypt.

Dr. Yousry Taha

Assistant Professor at:

Department of Computer Science and Automatic Control,

Faculty of Engineering,

Alexandria University,

Egypt.

ABSTRACT

The problem of mining association rules has attracted a lot of attention in the research community. Several techniques for efficient discovery of association rules have appeared. However, it is nontrivial to perform incremental mining or efficient mining of constrained association rules, in spite of their practical benefits. The research community has recently focused on providing separate solutions for these two problems.

Since many believe that constrained mining will be the standard, incremental mining of constrained rules will be necessary. In this thesis, a new algorithm, *ICAP*, for incremental mining of constrained association rules is introduced. The concept of constrained negative border is also introduced and its usage for the maintenance of constrained association rules when new transaction data is added to a transactional database is proposed. This fast incremental mining technique is applied to the constrained association mining algorithm *CAP*. A key feature of the proposed algorithm is that it requires at most one scan of the original database only if the database insertions cause the constrained negative border to expand. Thus the speed-up of incremental mining is combined with the flexibility of the framework of constrained frequent-set queries, *CFQs*, used for specifying user constraints on the produced association rules. The correctness of the proposed algorithm is studied and a proof of it is given.

Several experiments were conducted to measure the relative performance of the new algorithm compared to the single alternative approach available so far, which is rerunning the *CAP* algorithm on the whole updated database. The results of the experiments show a significant improvement over rerunning *CAP* in almost all of the cases.

ACKNOWLEDGMENTS

Thanks GOD before everything and after everything for giving me the knowledge and ability to complete this work in this final form.

I am very grateful and indebted to Prof. Dr. Nagwa M. El-Makky for her valuable and thorough supervision throughout all phases of this work, without which it would have never been complete. Her continuous and strong support and encouragement had some very profound effects on me that went beyond scientific supervision.

I would like to express my special thanks and gratitude to Dr. Yousry Taha for his supervision. His unlimited help and genuine support and encouragement are greatly appreciated.

Acknowledging my beloved wife for her unlimited support and encouragement through the hard times, her understanding and appreciation of my work and her indispensable assistance makes me feel much like I am acknowledging *myself!*

I am forever indebted to my family especially my mother and father for all their help both materially and morally. I would like to thank my mother for constantly including me in her prayers.

I would like to thank all the honorable faculty members of the department and my dear friends and colleagues for their help. My special thanks go to my dear friend Islam Mahmoud for his *JIT* support. He proved that he is my friend forever, especially in *need*.

TABLE OF CONTENTS

TABLE OF FIGURES

Chapter 1

INTRODUCTION

1.1. General

Following the explosive growth of the amount of data gathered by transactional systems, a challenge for finding new techniques to extract useful patterns from such a huge amount of data arose. Data mining emerged as the new research area to meet this challenge. Recently, data mining attracted a lot of research attention.

Two important issues demonstrate themselves in the data mining field, namely interactive mining and incremental mining. The first one is due to the huge amount of time it takes virtually all mining tasks to complete. Besides, if those tasks are left unguided, they can produce a great number of patterns leaving the user confused trying to extract what is really interesting for him. Thus, interactive mining through user-defined constrained queries in which the user specifies his interests (constrained mining) is essential to prevent data mining from becoming itself a knowledge problem [15, 26, 29, 43, 27, 28, 46, 1, 14].

The second issue, incremental mining, demonstrates itself in normal working environments. After initial investigation of the stored data, data starts to accumulate in small increments. It would be natural to try to adapt the mining tasks to handle those increments in an efficient manner using previously discovered patterns rather than mining from scratch [12, 20, 21, 23, 24, 30, 36, 39, 44].

Mining association rules, as one of the several data mining tasks, has a big share in the data mining research. This is attributed to its wide area of applications. Applications of association rule mining span a wide area of business from market basket analysis, to analysis of promotions and catalog design and from designing store layout to customer segmentation based on buying patterns. Other applications include health insurance, fraudulent discovery and loss-leader analysis [5, 8].

Mining association rules has the same challenges facing data mining in general. It is non trivial to perform incremental or constrained mining of association rules in spite of their obvious practical benefits. Many research efforts tried to present separate solutions for these two problems. Furthermore, the future is for constrained association rule mining and many

see that it should become the standard [26, 29, 35, 27]. A challenge lies in how to efficiently maintain the discovered rules that satisfy certain constraints incrementally without having to redo the whole task from scratch.

1.2. Motivation Of The Work

The great practical benefits of mining association rules and its wide area of applications have led to several proposals for fast mining of association rules. Those proposals, although contributed towards making the process more applicable in practical systems, still suffer from the problem of the huge amount of generated rules that are both confusing and most of the time not useful to the user. That's why constrained mining of association rules is a necessity for interactive mining. This is best described in what is quoted from [26]: "*Data mining is most effective when the computer does what it does best - like searching large databases or counting - and users do what they do best, like specifying the current mining session's focus. This division of labor is best achieved through constraint-based mining, in which the user provides restraints that guide a search.*"

Incremental mining is also a necessity if we are ever going to implement true practical data mining systems. Several research efforts showed that we could benefit greatly from previously discovered association rules to speed-up the process of producing the new set of rules after updates to the database. This gives a great improvement over the naïve approach of running a traditional algorithm over the new set of data no matter how good such an algorithm.

The research community focused on providing solutions for the two problems separately. Incremental association mining algorithms focused on maintaining the discovered rules that have the same support constraints, namely those which qualify for the same setting of *minimum support* as the original database before updates. On the other hand, the framework introduced in [35] and the introduction of other kinds of constraints together with the notion of anti-monotonicity and succinctness of constraints served the purpose of optimizing the execution of the mining algorithm for a set of user-defined constraints.

The previous points can be summarized by saying that the importance of mining association rules and the necessity of interactive and ad-hoc mining together with the need for incremental mining of associations highly motivate the need for effective techniques to incrementally mine association rules with constraints other than the traditional minimum support constraint [27]. So far, no such techniques have been proposed in the literature.

1.3. Scope Of The Work

In this thesis, a new algorithm for incremental mining of constrained association rules is proposed. The proposed algorithm fills the existing gap between incremental mining researches and constrained mining researches. The algorithm incrementally maintains the constrained frequent-sets after a database update. These frequent-sets were generated according to a user-defined constrained frequent-set query and originally discovered using the constrained association rule mining algorithm *CAP* (Constrained APriori) [35]. The output of the algorithm is the frequent-sets in the updated database that obey the original constraints specified in the query.

For the purpose of developing the algorithm, the concept of *constrained negative border* is introduced. It is the counterpart of the original negative border introduced in [45]. Proof is provided that it is an indicator for the necessity of checking the original database.

The key benefit of the proposed algorithm is that it requires at most one scan of the original database only when database updates cause the constrained negative border to expand.

Several experiments were conducted to explore the relative merits of the new algorithm when compared to rerunning the *CAP* algorithm on the whole updated database from scratch.

1.4. Thesis Organization

This thesis is organized as follows:

Chapter 2 presents the necessary background for the thesis. It presents the formal definition of the problem of mining association rules and the classical approaches to the solution of the problem. It also presents the recent improvements to the classical approaches and extensions to the original problem.

Chapter 3 includes the necessary theoretical foundations for the proposed algorithm. First, attention is devoted to the problems of incremental and constrained mining of association rules since they perform the groundwork upon which the proposed algorithm is built. Both problems are formally defined and the framework of constrained frequent-set queries is presented. The important theoretical bases of both problems are also given. The necessary formal definition of the problem of incremental mining of constrained association rules is then given. The concept of the constrained negative border is introduced and it is proved that it is an indicator for the necessity of checking the original database. Chapter 4 presents the proposed algorithm in details and proves its correctness. The chapter also discussed when to

use the algorithm and gives some of the important implementation details. Chapter 5 presents the performance study conducted on the proposed algorithm. Each conducted experiment is discussed and detailed comments on the results are given. Finally, Chapter 6 concludes the thesis and gives some suggestions for future work.

Chapter 2

BACKGROUND

This chapter surveys the background necessary for the rest of the thesis. It starts in section 2.1 by a discussion of the importance of the recently emerging field "knowledge discovery in databases" (*KDD*). It differentiates between the two terms KDD and Data Mining and then presents the different data mining tasks. Focus is rapidly moved in section 2.2 to the task of mining association rules. The section formally presents the problem and discusses the different applications of association rule mining. In section 2.3, the classical algorithms for finding frequent itemsets are presented. Section 2.4 surveys the different optimizations to the classical methods. Section 2.5 discusses the many natural extensions to the association mining problem that were discussed in the literature. Sections 2.6 and 2.7 are devoted to the discussion of two of the main extensions to the problem, namely constrained and incremental mining of association rules. Due to the specific relevance of these two areas to the subject of the thesis, both sections survey the research done in the two areas and set the necessary background before formally discussing the two problems in more detail in Chapter 3.

2.1. Knowledge Discovery in Databases

This section briefly introduces the general area of *Knowledge Discovery in Databases (KDD)* to which this thesis belongs. Section 2.1.1 presents the motivations of this important research area in brief. Section 2.1.2 differentiates between the general term *KDD* and the specific step of *Data Mining* by formally defining both terms, and finally section 2.1.3 skims on the rather diverse tasks of data mining.

2.1.1 Necessities of KDD

Recently, our capabilities of both generating and collecting data have been increasing rapidly. The widespread use of bar codes for most commercial products, the computerization of many business and government transactions, and the advances in data collection tools have provided us with huge amounts of data. Millions of databases have been used in business

management, government administration, scientific and engineering data management, and many other applications. It is noted that the number of such databases keeps growing rapidly because of the availability of powerful and affordable database systems. This explosive growth in data and databases has generated an urgent need for new techniques and tools that can intelligently and automatically transform the processed data into useful information and knowledge. Consequently, data mining has become a research area with increasing importance [19]

In [16], three questions were posed, in which Data Mining would make a good answer:

a- The query formulation problem: how can we provide access to data when the user does not know how to describe the goal in terms of a specific query?
b- How can we visualize and understand a large data set?
c- How can we cope with the increasing volume of data with time?

In the first problem, a good – and a rather natural – answer would be to *state the query by example.* In this case, the analyst would label a training set of cases of one class versus another and let the data mining system build a model for distinguishing one class from another. A typical example of such task is *classification*.

As for the second question, the problem is that data increases in both the number of fields (dimensions) and the number of cases (tuples). A solution of the number of dimension increase is to project the data set along a fewer-dimension space. However, this both increases the difficulty in modeling the data set for analysis and increases the number of possible combination for dimension reduction for higher numbers of dimensions. The Data Mining solution is to help perform the appropriate reductions by exploring all possibilities and selecting the right samples and attributes, then it may proceed to visualize the results for the user or analyst.

Concerning the fast increase in the rate of data volume growth, data mining can help in employing all available data in the analysis by utilizing the results performed on the previous one an adding the results of the new set in an incremental manner in such a way that no *manual techniques* could cope with. This is actually the distinguishing goal of *KDD*; dealing with a huge amount of data.

2.1.2 KDD or data mining?

Many publications use the term Knowledge Discovery in Databases as a synonym for Data Mining. Among the definitions for Data Mining are: *"The efficient discovery of previously unknown patterns in large databases."* [6] and *"The non-trivial extraction of implicit,*

previously unknown and potentially useful information (such as rules, constraints and regularities) from data in databases." [19]. However in [16] there is a preferable distinction between the two terms which will be chosen as their formal definition.

Definition 2.1: Knowledge Discovery in Databases (KDD)

It is the process of identifying valid, novel, potentially useful, and ultimately understandable structure in data. ❑

Definition 2.2: Data Mining

It is a step in the KDD process that, under acceptable computational efficiency limitations, enumerates structures (patterns or models) over the data. ❑

Data Mining refers to a particular step in the process of KDD. The additional steps in the KDD process, such as data preparation, data selection, data cleaning, incorporating appropriate prior knowledge, and proper interpretation of the results of mining, are essential to ensure that useful knowledge is derived from the data. The following figure represents the required processes for effective knowledge discovery.

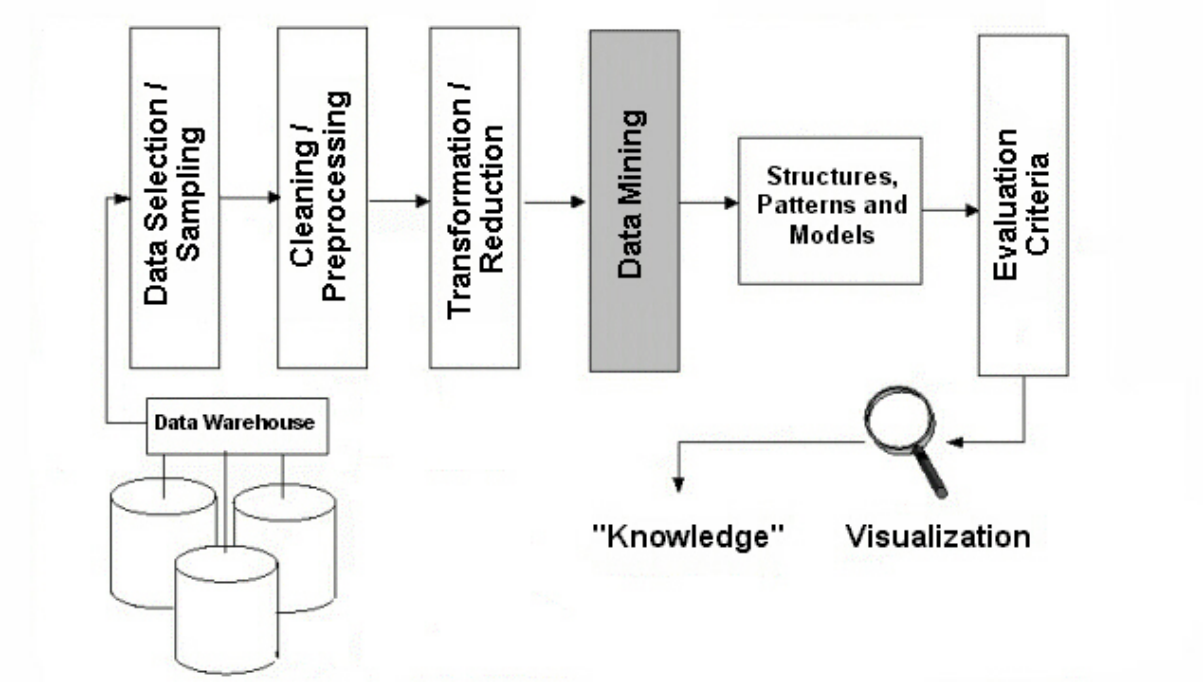

Figure 2.1: An overview of the steps comprising the KDD process [16]

2.1.3 Data mining tasks

Several data mining tasks exist. There are many attempts towards classifying and enumerating them (e.g. [16, 19]). However, the field is experiencing quite an explosion in research directions and extensions that might deem any attempt to enumerate those tasks and their variations and extensions incomplete. In the following, the most famous and agreed upon tasks are enumerated.

Association Rule Mining: It is the task of inferring rules which states that certain values occur with other values above a certain frequency in data with some pre-specified certainty. It is the focus of this thesis and thus will be given more attention shortly.

Classification: It is the process which finds the common properties among a set of objects in a database and classifies them into different classes, according to a classification model. The objective of the classification is to first analyze the training data and develop an accurate description or a model for each class using the features available in the data. Such class descriptions are then used to classify future test data in the database or to develop a better description (called classification rules) for each class in the database.

Clustering: It is the process of grouping physical or abstract objects into classes of similar objects. The term *unsupervised classification* is also often used. The term comes from the differentiation between Clustering and classification. The difference is that – though both methods produce a set of clusters with similar properties – in clustering, we don't know the number of output classes in advance. Clustering analysis helps construct meaningful partitioning of a large set of objects based on a "divide and conquer" methodology which decomposes a large scale system into smaller components to simplify design and implementation.

Mining Discriminant Rules: A discriminant rule is an assertion which discriminates concepts of the class being examined (*the target class*) from other classes (called *contrasting classes*). For example, to distinguish one disease from others, a discriminant rule should summarize the symptoms that discriminate this disease from others.

Mining Sequential Patterns: It can be stated informally as follows: The input data is a set of sequences, called data-sequences. Each data sequence is a list of transactions, where each transaction is a sets of items (literals). Typically there is a transaction-time associated with each transaction. A *sequential pattern* consists of a list of sets of items. The problem is to

find all sequential patterns with a user-specified minimum support, where the support of a sequential pattern is the percentage of data sequences that contain the pattern.

Pattern-Based Similarity Search: This is closely related to mining sequential patterns. Here, however, it is required to identify patterns in the data that are similar to each other. This similarity search can be defined as relative to a certain object or within a specified range of a user-specified metric, or else the task might be to find all pairs of similar patterns within a pre-specified range of the metric function.

Change and Deviation Detection: Deviations have been considered in many applications (e.g. AI) as outliers or noise in the data. However, they are the main focus of deviation detection schemes. The role of these schemes is to spot and isolate deviations in the data. The deviation detection problem can be considered a special case of the clustering problem as requiring to cluster data into deviations and non-deviations clusters. Deviation detection techniques rely on the redundancy present in the data to spot deviations as opposed to explicit information sources that resides outside the data as integrity constraints or predefined error/non-error patterns.

2.2. Association rule mining

2.2.1 Problem definition

The problem of finding association rules can be stated as follows [19]: given a database of sales transactions, it is desirable to discover the important associations among items such that the presence of some items in a transaction will imply the presence of other items in the same transaction. An example of an association rule is: 30% of transactions that contain beer also contain diapers; 2% of all transactions contain both of these items. Here 30% is called the *confidence* of the rule, and 2% the *support* of the rule. The problem is to find all association rules that satisfy user-specified minimum support and minimum confidence constraints [3].
The problem of mining association rules was first introduced in [5] and the following formal definition was proposed in [8] to address the problem.

Definition 2.3: Association Rules

Let $Item = \{i_1, i_2, ..., i_m\}$ be a set of literals called *items*, *DB* be a database of transactions where each transaction $T \subseteq Item$ and has a unique identifier, *TID*. Given an itemset $X \subseteq Item$, X is contained in T iff $X \subseteq T$. An association rule is an implication of the form $S_a \Rightarrow S_c$, where both S_a *(rule antecedent)* and S_c *(rule consequent)* are itemsets and $S_a \cap S_c = \Phi$. A rule has support s iff *s%* of the transactions in *DB* contain $S_a \cup S_c$. A rule has confidence *c* iff *c%* of the transactions containing S_a also contain $S_a \cup S_c$. An itemset is *frequent* iff its support exceeds a certain support threshold *minsup*. ❑

The problem of discovering all association rules can be decomposed into two sub-problems [5]:

a- Find all sets of items (itemsets) that have transaction support above the minimum support. These are the *frequent* itemsets. Other itemset called *infrequent* itemsets.

b- Use the frequent itemsets to generate the desired rules.

There is a wide agreement among the literature that the first sub-problem is the most important of the two. This is because it is more time consuming due to the huge search space (the power set of the set of all items) and the rule generation phase can be done in main memory in a straightforward manner once the frequent itemsets are found [8]. That is the reason for the great attention researchers paid to this problem in the recent years. Another reason for such attention may be due to the dispute among the research community about the *confidence* as a measure of rule importance or interestingness. Contrary to support, *confidence* received much criticism and many alternatives have been suggested with no global acceptance of one measure (e.g. [10, 17]).

2.2.2 Benefits and applications

The most famous application of association rules is its use for *market basket analysis* [4]. Consider a supermarket setting where the database records items purchased by a customer at a single time as a transaction. The planning department may be interested in finding "associations" between sets of items with some minimum specified confidence. Such associations might be helpful in designing promotions and discounts or shelf organization and store layout.

However, association rules have many other fields in which it have been helpful. In [11], two successful examples for the application of association rules in the telecommunications and medical fields for performing *partial classification* is reported. Association rule mining has been also used on other types of data sets. It has been used to mine web servers log files to discover the patterns that access different resources consistently and occur together or the access of a particular place occurring at regular times. Other types of data include census data and text documents as in [17] for example. Other examples of applications of association rules include catalog design, customer segmentation based on buying patterns, fraudulent discovery and health insurance.

2.3. Classical algorithms for finding frequent itemsets

All classical approaches to the problem benefit from the downward closure property of frequent itemsets with respect to set inclusion. It is proved that if an itemset is frequent then so must be all its subsets [8]. Using this property, all classical approaches, lest for the first two algorithms *AIS* and *SETM*, start by finding frequent itemsets of size 1 and continue extending the search to their supersets of size 2 as candidates of which large 2-itemsets are generated and so on. This is why such approach is often called the level-wise or the bottom-up approach because of the way they search the lattice of subsets.

2.3.1 AIS and SETM

2.3.1.1 The AIS algorithm [5, 8]

Candidate itemsets are generated and counted on-the-fly as the database is scanned. After reading a transaction, it is determined which of the itemsets that were found to be large in the previous pass are contained in this transaction. New candidate itemsets are generated by extending these large itemsets with other items in the transaction. A large itemset l is extended with only those items that are large and occur later in the lexicographic ordering of items than any of the items in l. The candidates generated from a transaction are added to the set of candidate itemsets maintained for the pass, or the counts of the corresponding entries are increased if they were created by an earlier transaction.

2.3.1.2 The SETM algorithm [8]

The SETM algorithm was motivated by the desire to use SQL to compute large itemsets. Like AIS, the SETM algorithm also generates candidates on-the-fly based on transactions read

from the database. It thus generates and counts every candidate itemset that the AIS algorithm generates. However, to use the standard SQL join operation for candidate generation, SETM separates candidate generation from counting. It saves a copy of the candidate itemset together with the transaction identifier (*TID*) of the generating transaction in a sequential structure. At the end of the pass, the support count of candidate itemsets is determined by sorting and aggregating this sequential structure.

The problem with these two algorithms was the size of the candidate sets generated of which many will be found infrequent. However, those algorithms remain important as the first attempts to generate association rules.

2.3.2 The Apriori algorithm

This pioneering work appeared first in 1994 [8] and remained the standard reference of all algorithms for finding association rules. It outperformed the previous algorithms by a great margin. The major difference in *Apriori* was the much less candidate set of itemsets it generates for testing in every database pass. Here comes the time saving. The algorithm benefited of the fact that for an itemset to be frequent, all its subsets must be frequent.

The algorithm starts by collecting all the frequent 1-itemsets in the first pass. It uses this set (called L_1) to generate the candidate sets to be frequent in the next pass (called C_2) by joining L_1 with itself. The trick here is that the algorithm eliminates from C_2 any set that has a subset not in L_1, because it knows *a-priori* that it can't be frequent, hence reducing its size dramatically. The algorithm proceeds in the same manner generating the candidates of size k from the large itemsets of size k-1, then reduces the candidate set by eliminating all those which have infrequent k-1 subsets, then counts occurrences of the remaining candidates in the next pass to find the frequent k itemsets. The algorithm terminates when there are no candidates to be counted in the next pass. Figure 2.2 gives the high-level description of the *Apriori* algorithm.

From the above, it can be seen that the algorithm makes k passes on the database where k is the size of the largest frequent itemset. This is the main drawback of the algorithm, especially if this k is large enough. Despite this drawback, the algorithm inspired the research community to further investigate the problem, and this lead to several optimizations and extensions to the algorithm and other approaches to the problem that differed from the level-wise approach. These extensions will be surveyed in the next sections.

```
Procedure AprioriAlg( )
1   Begin
2       L1 = {frequent 1-itemsets};
3       for (k = 2; Lk-1 ≠ ϕ ; k++ ) do {
4           Ck = apriori-gen( Lk-1 ); // New candidates
5           forall transactions t in the dataset do {
6               forall candidates c ∈ Ck contained in t do
7                   c.count++;
8           }
9           Lk = {c ∈ Ck | c.count ≥ min-support}
10      }
11      Answer = ⋃k Lk ;
12  End
```

Figure 2.2: Algorithm *Apriori* [8]

2.4. Improvements on the classical algorithms

Following the presentation of the *Apriori* algorithm, which greatly improved the performance of discovering association rules, making it more practical and appealing, the research literature was flooded by several efforts trying to improve on it. These efforts came in many forms and are the subject of this section.

2.4.1 Transaction and item pruning

This is one of the main optimizations of the *Apriori* algorithm. It is not needed to inspect the whole database each time it is needed to count occurrences of the candidate itemsets. This optimization reduces drastically the time needed to count the support for the candidate sets and enhances performance. Transaction pruning was present in two algorithms; *AprioriTid* [8] and *DHP* [37].

2.4.1.1 AprioriTid and AprioriHybrid

AprioriTid was proposed in the same paper with *Apriori* [8]. Though it does not state it explicitly, it uses transaction pruning to improve *Apriori's* performance. The main difference from *Apriori* is that it does not use the whole database to count support for candidate sets, rather, it uses another set it called C_k' in which every member is in the form of $<TID, \{X_k\}>$, where each X_k is a potentially large k-itemset present in the transaction with identifier *TID*.

Thus, if a transaction does not contain any large k-itemset, it will not be present in C_k' hence dropped out of consideration. This reduces the database size drastically especially in later phases when k increases. The rationale behind the operation of *AprioriTid* is obvious, since for a transaction to contain any frequent k+1-itemset, it must contain all its subsets which are also frequent, hence it must contain at least one frequent k-itemset.

The main drawback of this algorithm is that the size of the alternative set C_k' that represents the database may exceed the size of the actual database in early stages thus loosing its edge on *Apriori*. This motivated the authors of the algorithm to propose another algorithm, *AprioriHybrid*, that uses *Apriori* at the first stages and then switches to *AprioriTid* when transaction pruning becomes more effective. When to switch between the algorithms is not clear, however, and the authors showed that it can be determined heuristically.

2.4.1.2 DHP

In 1995, Park et al. [37] proposed *DHP*, which stands for *Direct Hashing and Pruning*. They observed that the bottleneck of *Apriori* is in the early stages, thus they proposed an effective hashing technique to further reduce the size of the candidate 2-itemsets. The reduction of the number of candidate sets gives the opportunity for effectively reducing the database size for the next counting pass. The authors observed that for a transaction to contain a frequent k+1-itemset, it must contain at least k+1 frequent k-itemsets. The algorithm then counts the number of candidate k-itemsets a transaction contains while counting support for candidates. If the transaction does not meet the required threshold, it can be eliminated from consideration in the next pass.

Another optimization, item pruning, in which the transaction size is reduced by eliminating certain items from consideration was also proposed. The authors observed that for an item to appear in a frequent k+1-itemset, it should appear in at least k frequent k-itemsets (the possible k subsets of size k of the frequent k+1-itemset that includes it). Thus while counting support for candidate k-itemsets, an item is pruned off the transaction if it is found to appear in less than k candidates.

DHP, with its optimizations, effectively improved on *Apriori* and turned attention towards methods to decrease the I/O overhead. However, it still required the same number of database passes as *Apriori* leaving this problem still unsolved. Another critique to the pruning methods presented in *DHP*, cited in [48], is that to exploit their benefits, the whole database should be rewritten several times which is obviously worse than just reading it. Otherwise, such pruning methods can be done logically by filtering the transactions.

2.4.2 Reducing the number of database passes

As mentioned before, the main drawback of the classical *Apriori* is the several passes it has to do on the database the number of which is equal to the length of the longest frequent itemset (pattern)* present in the database. Many optimization efforts focused on eliminating the number of database passes. They differed, however, in how the number of passes is reduced. This is the focus of this section.

2.4.2.1 Database partitioning

This technique was exploited in the *Partition* algorithm, proposed by Savasere et al. in 1995 [40]. The algorithm reduces the database activity by computing all frequent sets in two passes over the database. The algorithm works also in a level-wise manner, but the idea is to partition the database into sections small enough to be handled in main memory. That is, a part is read once from the disk, and level-wise generation and evaluation of candidates for that part are performed in main memory without further database activity. The first database pass consists of identifying the collection of all *locally frequent* sets in each database part. For the second pass, the union of the collections of locally frequent sets is used as the candidate set. The first pass is guaranteed to locate a superset of the collection of frequent itemsets; the second pass is needed to merely compute the frequencies of the sets to extract the *global frequent* sets.

The main achievement of *Partition* is the reduction of the database activity. It was shown that this reduction was not obtained at the expense of more CPU utilization, which is another achievement. It was shown, however, that the number of partitions greatly affects the performance of the algorithm by affecting the number of *locally frequent* itemsets that turn to be *globally infrequent.* The algorithm was also shown to be vulnerable to data skew. However, a remedy was proposed for this problem in the same publication.

2.4.2.2 Dynamic itemset counting

Another algorithm called *DIC* (*Dynamic Itemset Counting*) was proposed by Brin et al. in 1997 [18]. It tries to reduce the database activity by counting candidate itemsets earlier than *Apriori* does. In *Apriori*, candidate k+1-itemsets are not counted until the k+1st pass on the database (although an optimization called pass bundling [8, 33] permits counting candidates

* The term *frequent pattern* has been used freely in the literature to mean *frequent itemset.* Both refer to a subset of the universal set of *items*.

of different sizes in the same pass if memory is available). In *DIC*, on the other hand, candidate k+1-itemsets are counted as soon as the algorithm discovers that all its subsets of size k have exceeded the support threshold and will be frequent. This is done by stopping at various points in the database to examine the possibility of including other itemsets in the counting procedure. It has been found that such technique, with reasonable setting of the number of transactions passed before stopping for recalculation, can reduce the number of database passes dramatically while maintaining the number of candidate sets that need to be counted relatively low compared to other proposed techniques.

The algorithm, though more efficient than *Apriori*, was found to be vulnerable to data characteristics. A remedy was suggested for this problem with experimental success. It was also shown that by changing the logical order of items according to certain criteria, the performance of the algorithm increases dramatically. However, this *item reordering* strategy incurs a lot of overhead.

2.4.2.3 Using sampling techniques

In 1996, H. Toivonnen [45] proposed an algorithm that uses sampling to produce association rules. The algorithm can produce exact association rules in one full pass and two passes in the worst case. The idea of the algorithm is to pick a random sample of the data, find all the frequent itemsets in the sample and then verify the result on the database. The choice of the size of the sample and the support threshold is done by statistical methods. The support is slacken such that the frequent itemsets found in the sample are guaranteed to be a superset of the true frequent itemsets. The algorithm uses the concept of the *negative border* (see section 3.3.2) to ascertain that the itemsets found are truly the whole set of frequent itemsets. If a *miss* is reported in the first pass, a second pass is needed to gather information on the missed itemsets. A *miss* means an itemset in the negative border of the discovered set of itemsets in the sample which was found to be frequent, this indicates that it may have a superset that is also frequent. Such supersets are the ones that need to be tested in the second pass. However, this happens rarely if the sample size is well chosen according to the required support threshold.

The algorithm was shown to perform well compared to other level-wise algorithms and to the *Partition* algorithm. The database activity is reduced effectively to one pass. The only drawback of the algorithm, however, is that it has to test many spurious candidates due to the reduced support threshold and to guarantee a superset of the actual frequent itemsets.

2.4.3 Handling the problem of long frequent itemsets

The previous algorithms, were proved to scale almost linearly with the size of the database. This was supported theoretically in [49]. However, all such algorithms deteriorate exponentially with the length of the longest frequent itemset [13, 31, 7, 13]. This is an inherent problem due to the closure property of frequent itemsets with respect to set inclusion. Thus, if a long frequent pattern exists, it is accompanied by an exponential number of other frequent patterns which represent its powerset. This triggered research aiming at improving the performance of the classical algorithms both in terms of the execution speed and the size of the data structures needed to hold the required candidates to be counted when the database includes very long frequent itemsets. These efforts are the subject of the next two subsections.

2.4.3.1 Finding maximal frequent-sets

This new approach to the problem observes that we need not produce the whole set of frequent itemsets, instead we can produce the set of *maximal frequent sets.* A *maximal frequent set* is the one that has no proper frequent superset. Thus, the set of maximal frequent sets describes concisely the whole set of frequent itemsets and hence it is enough to produce them as an output. The search for the maximal frequent itemsets does not proceed in the ordinary bottom-up fashion of the level-wise algorithms and their variants. Rather, it may start searching for maximal frequent sets either by traversing up the lattice of subsets in earlier stages *looking ahead* for candidates (this approach is presented in the algorithm *MaxMiner* by Bayardo [13]) or by starting a top-down search of the lattice together with the normal bottom-up search (as presented in many algorithms, e.g. *Pincer-Search* by Lin et al. [31] and *MaxEclat* and *MaxClique* by Zaki et al. [7, 13]).

The above algorithms perform an order of magnitude faster than the best implementations of ordinary level-wise algorithms that produce the whole set of frequent itemsets.

2.4.3.2 FP-growth and TreeProjection

Two very recent studies [27, 2] tackled the problem of long frequent patterns. In [27], a novel data structure called the *FP-tree* (Frequent Pattern tree), which is an extended *prefix tree* structure [33], to compactly store the set of frequent patterns. The benefit of this data structure is that it can be efficiently constructed from the database and then mining can be done on it without further reference to the original database. Furthermore, the ratio of the size of the *FP-tree* to the size of the database it represents can be up to 1:150 [27]. This is a

tremendous gain since the structure holds all the necessary information needed to perform the mining process. The reason this data structure can achieve such very high reduction in database size is that it benefits from the fact that long frequent patterns share long prefixes, this structure stores the shared prefix only once. An efficient algorithm *FP-growth* was presented to find all the frequent itemsets from such data structure in a recursive manner and was shown to be an order of magnitude better than the classical *Apriori*. It also has the desired property of linear growth in terms of memory requirement with the length of the longest frequent pattern and database size. The previous data structure and technique are the subject of further investigations to be adapted for the different variations of the problem, including the generation of the *maximal frequent* itemsets discussed above. The vairarions will be discussed in the next section.

A similar recent work [2] presented another algorithm called *TreeProjection* that uses another compact tree structure and performs mining recursively on smaller portions of the database to achieve better performance gain over the classical methods.

2.5. Extensions and variations of the problem of mining association rules

Since the introduction of the basic problem of mining association rules, several extensions and variations of the problem were presented. In this section we skim on these extensions and variations for the sake of completeness in surveying the research work related to the problem.

Two natural variations or extensions to the problem are the problems of mining constrained association rules and incremental mining of association rules. Since proposing a solution to both is the subject of this thesis, they will be surveyed in more detail in the next two sections and will be formally defined in the next chapter.

Another natural extension to the problem is parallel mining of association rules. Several algorithms were proposed, virtually all of them are parallel versions of the existing sequential algorithms. A very good survey and a classification of all parallel and distributed algorithms for mining association rules exist in [48].

In many applications, interesting associations among data items often occur at a relatively high concept level. For example, the purchase patterns in a transaction database may not show any substantial regularities at the primitive data level, such as at the bar code level, but may show some interesting regularities at some high concept level(s), such as milk and bread.

Therefore, it is important to study mining association rules at a generalized abstraction level. This initiated the study of mining generalized association rules [25, 42]

Interactive mining has always been a hot subject, and efforts to adapt the problem to be more on-line existed. Among the efforts is the work in [22] which presented an algorithm for discovering association rules in two phases where the user can interactively monitor the discovered frequent patterns in the first phase and change the support threshold accordingly until the first full pass on the database, then a second pass outputs the frequent patterns and association rules according to the last setting by the user. In [34], the authors suggested using a knowledge cache for the discovered patterns to speed up the response time of the algorithm.

Another inherent problem in the classical definition of association rules is that sometimes items that rarely exist in the database are of interest to the user. However, for such items to participate in the discovered rules, the support threshold should be set at a relatively low value. Such a setting results in many other non-interesting rules. On the other hand, raising the support threshold blocks any knowledge about the interesting item. This problem, called the *rare item* problem, was presented in [32] and an algorithm that enables the user to specify several minimum supports to reflect the nature of the items and their varied frequencies in the database was proposed. In rule mining, different rules might need to satisfy different minimum supports depending on what items are in the rules.

Mining inter-transaction association rules [47] and mining sequential patterns [9] add a temporal component to the problem in different manners. In these problems, the object of the mining process is to discover frequent patterns that occur ordered within some specified period of time.

Another variation of the original problem is mining for negative association rules. The intuition behind this is that sometimes it is informative to know which items *do not* occur together. The direct approach to the problem suggests considering the disappearance of an item from a transaction another virtual item and apply the same classical techniques. This is clearly inefficient and produces a horrendous amount of unnecessary and uninteresting rules. Efficient solutions to the problem are present in [41].

2.6. Constrained association rule mining

After the first introduction of the problem of mining associations in the context of market basket analysis, it has been tested on other types of data [15]. Domains other than market-basket data tend to be *dense*. Properties of dense data sets are found in [15]. Mining dense

data cause an exponential blow up in the size of the generated rules if we only use support and confidence constraints [15]. Paradoxically, data mining produces such great amounts of data that there is a new knowledge management problem [28, 46].

Another observation that follows immediately from the previous observation is that despite the huge volume of produced rules, a small fraction of which may be useful or of practical interest to the user. This is because the user may be interested in only a subset of items, or in the presence or absence of a certain item in either the rule antecedent or consequent or both. In an interactive mining environment, it becomes a necessity to enable the user to express his interests through constraints on the discovered rules, and to change these interests interactively.

The previous reasons motivated the emerging of the problem of discovering constrained association rules which will be discussed in the next subsections.

2.6.1 Types of rule constraints

The most famous rule constraints are *item constraints*, which are those that impose restrictions on the presence or absence of items in a rule. These constraints can be in the form of a conjunction or a disjunction. Such constraints have been introduced first in [43] where a new method, for incorporating the constraints into the candidate generation phase of the *Apriori* algorithm, was proposed. In this way, candidates are assured to obey the item constraints besides the original support and confidence constraints. The definition given in [43] also extends to constraints for items with a defined *is-a* hierarchy. In [28], a framework of mining such constraints using user-defined templates, that can be acquired through an appropriate interface, is investigated together with a tool for rule visualisation. Other famous types of item constraints are rules with fixed consequent or with one item in the consequent [15].

2.6.2 Constrained frequent-set queries

Item constraints are not the only possible type of rule constraints. Ng et al. [35] presented a wide range of constraints on rules that extends from relational operators on values of the items to constraints on the value of some aggregate functions calculated on the rule items. They defined what is called *Constrained Association Queries* (*CAQs*) (latter renamed to the more appropriate name *Constrained Frequent-set Queries* (*CFQs*) in [29]) and presented an excellent classification of constraint constructs that can be exploited in them by introducing the notions of succinct and anti-monotone constraints. The *CAP* (Constrained APriori)

algorithm was presented for efficient discovery of constrained association rules. See section 3.1 for a detailed formal discussion of the problem and presentation of the *CAP* algorithm.

2.7. Incremental mining of association rules

As mentioned in chapter 1 and in section 2.5, incremental mining of association rules is a natural extension to the classical problem. Its importance stems from its practicality and from the nature of real data which tend to accumulate incrementally. Several proposals to solve the problem were introduced. It is the aim of this section to briefly survey such work before formally introducing the problem in section 3.3.

2.7.1 FUP and FUP_2

The work in [20] is credited for being the first to draw attention to the problem of incremental mining of association rules. In their paper, in which they presented an incremental algorithm for maintaining association rules called *FUP* (Fast UPdate), Cheung et al. [20] stated that maintenance of association rules involve searching for two things:

a- Losers, frequent itemsets that became infrequent after adding the increment data to the database.

b- Winners, infrequent itemsets that became frequent after adding the increment data to the database.

The second problem is harder to solve. *FUP* tries to benefit from the previously discovered rules to generate a relatively small candidate set to be checked against the original database by eliminating earlier itemsets that are either known to be still frequent or are deemed infrequent just by checking the increment. This way, *FUP* improves performance drastically compared to using the straightforward manner of rerunning the mining algorithm against the whole updated database from scratch.

FUP deals with the situation in which the change in the database is in the form of inserting new transactions. However, a database can be changed by either inserting, deleting or modifying transactions. FUP_2 [21], was introduced to handle the general case of database update by handling both insertions and deletions (note that a modification can be viewed as a deletion followed by an insertion). The algorithm degenerates to *FUP* in case of only insertions. In case of only deletions the algorithm shows excellent performance especially for small deleted sets with respect to the original one (which is normally the case since the original database tends to be huge in size). In the general case, the algorithm performs better

than rerunning traditional algorithms from scratch. However, the performance of FUP_2 was found to degrade with the increase in the difference between the old and new sets of transactions (which is quite intuitive and understandable). It was found that when the size of the difference reaches around 40% of the original database, the incremental update algorithm looses its edge.

2.7.2 Using sampling techniques to trigger update

A certain drawback of an incremental algorithm is that it incurs a large overhead if applied too frequently (the extreme case is with every new update) with no practical benefit. On the other hand, if it is applied at long intervals, either we may miss several important new rules or, if the interval is too long or the updates are too frequent, the size of the update makes it a bad choice compared to running a traditional algorithm to get the new rules from scratch. The question is when it is efficient to use an incremental algorithm? *DELI* (Difference Estimation of Large Itemsets) [30] is an algorithm that uses sampling to estimate the difference between the old and the new sets of rules. If the difference is large enough to justify the overhead, an incremental algorithm is used. If not, the old set of rules can be used as a good approximation of the current situation. Clearly, we need a measure to tell when it is necessary to update. DELI uses the *set symmetric difference* between the old set of large itemsets L and the new one L', denoted by $L \Theta L'$, to measure the difference between them. Its size can vary from zero to $|L| + |L'|$. So the ratio $\frac{|L \Theta L'|}{|L| + |L'|}$ is intuitively a good candidate to indicate the need for an update. However, since we need $|L'|$ to calculate the measure, and our aim is to estimate it, the ratio $\frac{|L \Theta L'|}{|L|}$ is used instead and the aim of the algorithm is to get a good estimate of this measure through sampling.

2.7.3 Using early pruning

A recent study [12] proposed an efficient algorithm, *UWEP* (Update With Early Pruning), for solving the incremental mining problem. The algorithm benefits from the *anti-monotone* property of support [35] to prune early candidates that need to be checked against the database. It was proved that the number of candidates generated by *UWEP* is the minimum possible. However, in spite of what was stated in the paper that the algorithm needs only one scan on the updated database, *UWEP* actually passes on the updated database several times, but instead of actually reading the database transactions it uses *TIDLists* as in [40]. However,

we can argue that creating and maintaining such lists in the way presented in [12] may incur a comparable overhead with actual passing on the database especially in the first couple of passes.

2.7.4 Using the negative border

The major drawback of *FUP* and all its modifications is that they build on the *Apriori* algorithm and its modifications. This means requiring as many passes on the database as the length of the largest frequent itemset in the updated database. To remedy this problem, the two research papers [23, 44] simultaneously proposed a marvelous algorithm for incremental mining that uses the concept of the *negative border* introduced by Toivonen [45] to indicate the need for an update of rules. It was proved that for an itemset to become a winner it should have a subset in the negative border that is also a winner. Hence, if no winners appeared in the negative border, there is no need for checking for more in the original database (see section 3.3.2 for a proof of this result). The proposed algorithm managed to get the required update in only one scan of the original database (which is typically an order of magnitude more than the increment database). A similar study [39] uses also the concept of negative borders to signal the need for updating the database. The main power of these algorithms rely in that no scanning of the original database is done unless it is *absolutely* necessary. The proposed algorithm in [39] requires re-computation of the negative border whenever a change is detected.

The algorithms in [23, 44] (the BORDERS algorithm) are considered the best incremental algorithms for mining classical association rules [24]. In fact, the proposed algorithm of this thesis applies exactly the same concept but on the problem of mining constrained association rules.

2.7.5 ECUT and ECUT$^+$

The very recent work in [24] argues that, typically, data evolves dynamically with time through additions and deletions of blocks of data. It introduces a new dimension called the *data span* dimension, which allows user-defined selections of a temporal subset of the database. As a result of the extra degree of freedom, efficient model-maintenance algorithms for frequent itemsets and clusters are presented.

ECUT and ECUT$^+$ (Efficient Counting Using TIDLists) are the proposed algorithms for maintaining frequent patterns. They proceed exactly like the BORDERS algorithm in [23, 44] in the detection phase of new candidates for counting against the updated database. The

improvement is in the method used for counting these candidates. The *ECUT* and *ECUT*$^+$ algorithms cleverly *selects* the relevant portions of the database which are needed to count the support of the new candidates (much like an index structure efficiently selects the relevant portions of data from the database). This is achieved through maintaining the database in a *TIDList* format [40]. The difference between the two algorithms rely in the method of constructing the *TIDLists*.

Chapter 3

FOUNDATIONS OF THE PROPOSED ALGORITHM ICAP

This chapter presents the theoretical foundations upon which the proposed algorithm, *ICAP* (Incremental Constrained APriori), is built. These foundations include the theoretical bases of incremental and constrained mining of association rules and the *CAP* (Constrained APriori) algorithm on which the proposed algorithm is based.

The chapter is organized as follows: section 3.1 informally discusses the problem and the several challenges that is faced when trying to tackle it. section 3.2 discusses constrained mining of association rules. It begins by presenting the framework of constrained frequent-set queries and then it presents the constrained association rule mining algorithm *CAP* in details. Section 3.3 is devoted to incremental mining of association rules. The formal definition of this problem is given. Since the negative border has an important role in solving the problem, the formal definition of the negative border is also given. Section 3.4 formally presents the new problem of incremental mining of constrained association rules. The constrained negative border is proposed and proved to have the same importance as the original negative border; namely being an indicator for triggering the scan of the original database after an update.

3.1. The problem

This thesis proposes an algorithm to efficiently solve the problem of incremental mining of constrained association rules. This problem, as described in the previous chapters, has two separate foundations; namely the problems of incremental and constrained mining of association rules. In the previous chapter, the general motivation of these problems were discussed and previous work was surveyed. It would be appropriate, however, before proceeding with the theoretical foundations upon which the proposed algorithm is based to informally discuss the technical challenges faced when trying to tackle these problems.

The main technical challenges that faced researchers seeking a solution for incremental mining of association rules were:

a- How to efficiently use the information previously collected about the original database. This information was used both to reduce the number of candidates that needed counting [12, 20, 21, 36] and to minimize the number of database passes.

b- How to use this information to effectively discover the necessity for updating the discovered association rules [44, 23, 39, 30]. Latter on, the concept of the *negative border* proved useful to reduce the number of passes to a single pass and only when it is *absolutely* necessary [44, 23, 39].

On the other hand, constrained mining of association rules is faced with several other challenges, for example:

a- How to provide the user with a rich tool to explore and exploit the semantics of data more effectively?

b- The main challenge is how to integrate the constraints earlier in the mining procedure (push the constraints *deeply* into the rule generation process) rather than using the naïve approach of running a traditional algorithm then using a post-processing pass to filter the generated rules [26, 35].

Upon examination of the above challenges and seeing the proposed solutions, it would seem natural then, to try to further increase the performance gain achieved in constrained mining of association rules by applying the techniques of incremental mining. The motivations in this case are similar to the original motivations of the problem of incremental mining of association rules. This, however, is not at all trivial [12, 26, 25]. Several questions still need answers:

a- How can we benefit from the properties of the constraints (either succinctness or anti-monotonicity or both) in reducing the number of candidates that need to be tested against the original database? It is to be noted that this is the primary challenge for an incremental mining algorithm.

b- What is the effect of the succinctness or anti-monotonicity properties of constraints on the notion of the *negative border*? It is to be noted that negative borders were extremely useful in signaling the need for scanning the original database.

c- What data about the discovered frequent itemsets should be kept besides the support count?

d- What happens if a database update introduced changes in the characteristics of the items (e.g. a change in the items' price or a change in the items' is-a hierarchy)?

Answers to some of the above questions are the concern of this thesis. The next sections present the theoretical basis on which the proposed work is based.

3.2. Constrained mining of association rules

As discussed in the previous chapter, association rule mining can produce a huge amount of patterns that are most of the time not useful to the user. This *lack-of-focus* problem [35] necessitates new means of expressing the user needs using convenient methods. The framework of Constrained Frequent-set Queries, *CFQs*, was introduced in [35] to give a solution to the problem. The queries were used as a means of specifying the constraints to be satisfied by the antecedent and consequent of a mined association. In this section, this framework is formally presented, focus is then moved to defining the notion of anti-monotonicity and succinctness. The section concludes with a detailed description of the *CAP* [35] algorithm which makes use of anti-monotonicity and succinctness to speed up the process of discovering the constrained frequent-sets.

3.2.1 The framework of constrained frequent-set queries

In [35], an architecture for exploratory mining of association rules was presented. This architecture was divided into two phases. Phase I concerned specifying the relevant part of the transactional database (*minable view*) and then iteratively constraining the required frequent-sets through a series of user decisions. Phase II was devoted to letting the user choose the required metric to measure rule interestingness and specify the required metric threshold among several metrics (e.g. interest, correlation and confidence).
The corner stone of Phase I of this architecture was the *CFQs*, which represent the tool by which the user expresses his interest and narrows the focus of the search for interesting rules.

3.2.1.1 Definition of Constrained Frequent-set Queries

Definition 3.1: Constrained Frequent-set Queries (CFQs)

Following the definition in [35], a *CFQ* is defined to be a query of the form: $\{(S_a, S_c) | C\}$, where C is a set of constraints on S_a and S_c (*rule antecedent* and *rule consequent* respectively). The syntax of constraint constructs is as described in Figure 3.1. A *set variable* is either an identifier of the form *S* or an expression of the form *S.A*, where *A* is an attribute in the minable view. Added to the these constructs is the frequency constraint of the form *freq(S)*, saying that the support of *S* should exceed a certain threshold. ❑

1. *Single Variable Constraints:* A single variable (*1-var*) constraint is of one of the following forms.
   (a) *Class Constraint:* It is of the form $S \subset A$, where S is a set variable and A is an attribute. It says S is a set of values from the domain of attribute A.
   (b) *Domain Constraint:* It is one of the following forms.
      i. $S \ \theta \ v$, where S is a set variable, v is a constant from the domain that S comes from, and θ is one of the boolean operators $=, \neq, <, \leq, >, \geq$. It says *every* element of S stands in relationship θ with the constant value v.
      ii. $v \ \theta \ S$, where S, v are as above, and θ is one of the boolean operators $\in, \notin$. This simply says the element v belongs (or not) to the set S.
      iii. $V \ \theta \ S$, *or* $S \ \theta \ V$, where S is a set variable, V is a set of constants from the domain S ranges over, and θ is one of $\subseteq, \subset, \not\subset, =, \neq$.
   (c) *Aggregate Constraint:* It is of the form $agg(S) \ \theta \ v$, where *agg* is one of the aggregate functions *min, max, sum, count, avg* and θ is one of the boolean operators $=, \neq, <, \leq, >, \geq$. It says the aggregate of the set of numeric values in S stands in relationship θ to v.
2. *Two Variable Constraints:* A two variable constraint (*2-var*) is of one of the following forms.
   (a) $S_1 \ \theta \ S_2$, where S_i is a set variable and θ is one of $\subseteq, \subset, \not\subset$.
   (b) $(S_1 \ \Diamond \ S_2) \ \theta \ V$, where S_1 and S_2 are set variables, V is a set of constants or ϕ, $\Diamond$ is one of $\cup, \cap$, and θ is one of $\subseteq, \subset, \not\subset, =, \neq$.
   (c) $agg_1(S_1) \ \theta \ agg_2(S_2)$, where agg_1, agg_2 are aggregate functions and θ is one of the boolean operators $=, \neq, <, \leq, >, \geq$.

Figure 3.1: Syntax of Constraint Constructs [35]

Assuming that the minable view consists of the relations **`trans(`**`TID, Itemset`**`)`** and **`itemInfo(`**`Item, Type, Price`**`)`**, some examples of *CFQs* are as follows: The query $\{(S_1, S_2) \mid S_1 \subset Item \ \& \ S_2 \subset Item \ \& \ count(S_1) = 1 \ \& \ count(S_2) = 1 \ \& \ freq(S_1) \ \& \ freq(S_2)\}$ asks for all pairs of single items satisfying frequency constraints. The query $\{(S_1, S_2) \mid S_1.Type \supseteq \{snacks, sodas\} \ \& \ S_2.price \geq 30\}$ asks for itemsets in the antecedent of the rule, which contain at least one snack item and one soda item, that are associated with itemsets that contain items with a price of at least \$30. Finally, the query $\{(S_1, S_2) \mid S_1.Type \subseteq \{snacks\} \ \& \ S_2.Type \subseteq \{beverages\} \ \& \ max(S_1.price) \leq min(S_2.price)\}$ finds pairs of sets of cheaper snack items and sets of more expensive beverage items. In the last

two examples, the domain and frequency constraints (those stating that itemsets should belong to the universal set *Item* and that they should be frequent) are omitted for brevity.

As shown in Figure 3.1, constraints in [35] were divided into 1-variable constraints (those constraints concerning only one *set variable,* as in the first and second examples) and 2-variable constraints (those relating two *set variables,* as in the third example). In this thesis, however, focus is only on 1-variable constraints.

The constraints in [35] were classified according to two orthogonal properties; anti-monotonicity and succinctness. Informally, an anti-monotone constraint is a *frequency-like* constraint, meaning that it satisfies the same closure property of the support constraint, which states that if an itemset satisfies the constraint, then all its subsets will do. A succinct constraint, on the other hand, is a constraint having the property that all itemsets following it can be generated, using some *member generating function (MGF)*, once and for all before any iterations take place. The importance of this classification relies in that one can exploit both properties while trying to discover frequent sets that follow the constraints. This gives a great advantage over the naïve approach of counting the frequent sets first and then filtering out those which do not follow the constraints. The next subsection includes the formal definitions of both properties. The *CAP* algorithm [35], which uses both properties to optimize the process of discovering constrained frequent-sets, will be discussed in the next section.

3.2.1.2 Anti-monotonicity and succinctness

Before proceeding with the presentation of the meaning of anti-monotonicity and succinctness, it would be convenient to list the notations used henceforth in the thesis. Table 3.1 summarizes these notations.

DB	The original database
Db	The increment database
DB+	The updated database
$\|A\|$	Number of transactions in database A
Minsup	Minimum support threshold
C	A constraint on association rules
C_k^c	The candidate set of size *k*. The superscript *c* (if present) represents a constraint *C*
C_{freq}, C_{am}, C_{succ}, C_{sam}	The frequency, anti-monotone, succinct and succinct-anti-monotone components of the constraint C respectively
C-C_x	The components of the constraint *C* excluding the component *x*, where *x* stands for *succ, am* or *freq*.
L_c^A	The set of frequent itemsets in database *A*. The subscript *c* (if present) represents a constraint *C*
$t_A(s)$	Support count of itemset *s* in database *A*

Table 3.1: List of used notations

Definition 3.2: anti-monotonicity

A 1-variable constraint *C* is *anti-monotone* iff for any set *S*: *S* does not satisfy $C \Rightarrow \forall S' \supseteq S, S'$ does not satisfy *C*. ❑

An example of an anti-monotone constraint is the constraint $S.price \geq 30$ in the second example of subsection 3.2.1.1. This constraint enforces every item in the resulting itemset *S* to have its price greater than or equal to \$30. This is clearly anti-monotone, since a single item in *S* that has a price lower than \$30 will be included in all supersets *S'* of *S* and hence violating the constraint. Referring to the example database of Figure 3.2, the itemset $S_1 = CD$[†] satisfies the constraint, while $S_2 = CDE$ and all its supersets will violate the constraint because of the inclusion of *E* which has a price of \$25 < \$30. Another example is the constraint $S.Type \subseteq \{snacks, soda\}$ which is also obviously anti-monotone by a similar argument. As mentioned before, these types of constraints are frequency like, which may lead

[†] From now on, itemsets will be written as a series of continuous item identifiers for simplicity. For example the itemset {C, D} will be written as CD and the itemset {A, B, C} will be written as ABC.

to the belief that techniques similar to those used to prune non-frequent itemsets might be used.

itemInfo

Item	Type	Price
A	Dairy	10
B	Soda	20
C	Snack	50
D	Juice	30
E	Candy	25
F	Snack	45

trans

TID		Itemset
DB	1	A B
	2	A B C
	3	B C E
	4	C D F
	5	B E
	6	A
	7	B
db	8	A B C F
	9	A E
	10	A B C D E

Figure 3.2: A database example

Definition 3.3: Succinctness

1. $I \subseteq$ Item is a *succinct* set if it can be expressed as $\sigma_p(Item)$ for some selection predicate p.
2. $SP \subseteq 2^{Item}$ is a *succinct powerset* if there is a fixed number of succinct sets $Item_1, \ldots Item_k \subseteq Item$ such that SP can be expressed in terms of the strict powerset of $Item_1, \ldots Item_k$ using union and minus.
3. A 1-variable constraint C is *succinct* provided $SAT_C(Item)$ is a succinct powerset. ❑

Where $SAT_C(Item)$ is the set consisting of all the subsets of *Item* that satisfy C, and 2^{Item} is the strict powerset of *Item,* i.e. the set of all subsets of *Item* except the empty set.

For example, the constraint $min(S.Price) \leq 500$ is succinct because we can precisely generate all the set of items satisfying the constraint without recourse to a generate-everything-and-test approach. In contrast, the constraint $avg(S.Price) \leq 500$ is not succinct because the average is inherently dependant on all the items in an itemset and this cannot be reduced to a simple selection on individual items of the entire set.

An important property of a succinct constraint is that its solution space can be expressed using what was called *member generating function* in [35]. Using the member generating function, one can avoid the generate-and-test environment of other types of constraints. This

property was used for optimizing *CAP* [35]. The definition of member generating functions from [35] is included for the sake of completeness.

Definition 3.4: Member Generating Functions (MGFs)

1. We say that $SP \subseteq 2^{Item}$ has a *member generating function (MGF)* provided there is a function that can enumerate all and only elements of *SP*, and that can be expressed in the form $\left\{X_1 \cup ... \cup X_n \middle| X_i \subseteq \sigma_{p_i}(Item), 1 \le i \le n \;\&\; \exists k \le n : X_j \ne \phi, 1 \le j \le k\right\}$, for some n ≥ 1 and some selection predicates $p_1, \ldots, p_n$.
2. A 1-variable constraint *C* is *pre-counting prunable* provided $SAT_C(Item)$ has an *MGF*. ❑

It was proved in [35] that any succinct constraint *C* is pre-counting prunable, meaning that it has an *MGF* associated with it.

An example of a succinct constraint is the constraint $S.Type \supseteq \{snack\}$ which enforces all itemsets to have at least one snack item. The corresponding *MGF* of this constraint is $\left\{X_1 \cup X_2 \middle| X_1 \subseteq \sigma_{Type='snack'}(Item), X_2 \subseteq \sigma_{Type\neq'snack'}(Item), X_1 \ne \phi\right\}$. To see the intuition behind this, one should notice that all itemsets following the constraint should have an item from the set of items which are of type *'snack'*. This is expressed using the member generating function by stating that the required itemsets can be generated by combining items from X_1 (snack items) and items from X_2 (non-snack items). The last condition, $X_1 \ne \phi$, ensures that at least one snack item is selected.

Applying the constraint on our example database of Figure 3.2, it can be seen that $X_1 \subseteq CF$, $X_2 \subseteq ABDE$ such that the itemsets *C, AC, and CDF* for example satisfy the constraint since their intersection with *CF* is not the empty set. While the itemsets *A, BD, and DE* for example do not satisfy the constraint for the opposite reason.

The previous definitions open the door for the generate-only optimization strategy for succinct constraints used by *CAP* in which all generated itemsets for counting are guaranteed to satisfy the constraint without the need of further testing.

3.2.2 The Constrained APriori (*CAP*) algorithm

In this section the *CAP* algorithm [35] is presented in detail. The optimization strategies for every type of constraint are presented and then a high level description of the algorithm is given in Figure 3.3.

CAP is based on classifying the constraints according to the anti-monotonicity and succinctness properties. Since they are orthogonal, there are 4 distinct cases:

I. Constraints that are both anti-monotone and succinct.

II. Constraints that are succinct but not anti-monotone.

III. Constraints that are anti-monotone but not succinct.

IV. Constraints that are neither succinct nor anti-monotone.

The *CAP* algorithm deals with each case with a different strategy for optimization. Each strategy will be discussed briefly before describing the algorithm itself. Since the *CAP* algorithm is based on the *Apriori* algorithm, the strategies are presented as optimizations to the *Apriori* algorithm (see section 2.3.2 for the discussion of *Apriori*).

Case I: Succinct and anti-monotone constraints

It is proved in [35] that the *MGFs* of such constraints are in the form of $\{X | X \subseteq \sigma_p(Item) \,\&\, X \neq \phi\}$. Thus, the set C_1 of candidates of size 1 is replaced by the set $C_1^c = \{e | e \in C_1 \,\&\, e \in \sigma_p(Item)\}$ in the *Apriori* algorithm. Since this type of constraints is also anti-monotone, no items other than those in the modified candidate set need to be considered. This leads to the following strategy:

Strategy I:

Replace C_1 in the *Apriori* algorithm by C_1^c as defined above. ❑

The intuition behind this strategy is that, since the member generating function of such constraints restricts the items in the itemsets to only those coming from $\sigma_p(Item)$, this set simply replaces the universal *Item* set.

Case II: Succinct but non-anti-monotone constraints

CAP uses the structure given by the *MGF* for such constraints. Following [35], the strategy is described assuming the succinct constraint has an *MGF* defined as follows:

$$M = \{S_1 \cup S_2 | S_1 \subseteq \sigma_{p_1}(Item) \,\&\, S_1 \neq \phi \,\&\, S_2 \subseteq \sigma_{p_2}(Item)\}.$$

Strategy II:

1. Define $C_1^c = \sigma_{p_1}(Item)$ and $C_1^{\neg c} = \sigma_{p_2}(Item)$. Define corresponding sets of frequent sets of size 1: $L_1^c = \{e | e \in C_1^c \,\&\, freq(e)\}$, and $L_1^{\neg c} = \{e | e \in C_1^{\neg c} \,\&\, freq(e)\}$.

2. Define $C_2 = \left\{\{e, f\} \middle| e \in L_1^c \;\&\; f \in \left(L_1^c \cup L_1^{\neg c}\right)\right\}$, and L_2 as the set of frequent sets in C_2.

3. In general, C_{k+1} can be obtained from L_k in exactly the same way as in *Apriori* – with one modification. In the classical case, a set S is in C_{k+1}, if all its subsets of size k are frequent. In this case, since a subset of size k may contain items from only $L_1^{\neg c}$; i.e. not satisfying the constraint, and since such subsets are not generated, hence there is no idea about their support, they are given the merit of doubt. This means that the condition of *Apriori* is restricted to include only those subsets of size k that are themselves satisfying the succinct constraint. Formally, a set S is in C_{k+1} if:

$$\forall S' : S' \subset S \; and \; |S'| = k \; and \; S' \cap L_1^c \neq \phi \Rightarrow S' \in L_k$$

❑

Extensions to the above strategy for member generating functions of the general form is straightforward [35].

From the definition of member generating functions, it can be noticed that for succinct constraints, there is a minimum cardinality for itemsets following the constraint (this property will be formally proved in subsection 3.4.2). The above strategy guarantees that every step of candidate generation, before reaching this minimum cardinality, will consist of only frequent sets. It is to be noted that, in the above strategy, it was possible to start from the beginning by $C_2 = C_1^c \times C_1$ and proceed with counting to get L_2. However, such technique will generate unneeded candidates that are known to be not frequent. Thus, the above strategy minimizes the number of generated candidates.

Case III: Anti-monotone but not succinct constraints

Such constraints are frequency-like constraints having the same closure property of the frequency constraint. This suggests using the same technique as in the *Apriori* algorithm. Itemsets are generated and tested for satisfaction of the anti-monotone constraint *before* being counted.

Strategy III:

Define C_k as in the classical *Apriori* algorithm. Drop a set $S \in C_k$ from counting if S fails to satisfy the constraint C. ❑

Case IV: Non-succinct and non-anti-monotone constraints

Such constraints do not have a strategy of optimization. However, what can be done is to *induce* a weaker constraint such that all itemsets satisfying the original constraint will also satisfy the weaker constraint. Using the weaker constraint will restrict the itemsets to a smaller superset of those satisfying the original constraint. Then, the generated itemsets are checked in a final round for satisfaction of the original constraint.

Strategy IV:

1. Induce any weaker constraint *C'* from the original constraint *C*. Depending on the type of *C'*, use one of the previous optimization strategies for optimization of the generation of frequent sets.
2. Test the generated frequent itemsets for the satisfaction of *C*. ❑

```
Algorithm CAP
1    If C_sam ∪ C_succ ∪ C_none is non-empty then
2          Prepare C_1 as indicated in strategies I, II and IV; k=1
3    If C_succ is non-empty then
4          Conduct a database scan to form L_1 as indicated in strategy II
5          Form C_2 as indicated in strategy 2; k=2
6    While C_k not empty
7          Conduct a database scan to form L_k from C_k.
8          Form C_k+1 from L_k based on strategy II if C_succ is non-empty, and
           strategy III for constraints in C_am
9    If C_none is empty then
10         Ans = ⋃_k L_k
11   Else an itemset S is in Ans iff S satisfies C_none.
```

Figure 3.3: High level description of the *CAP* algorithm

As an example for such a strategy, consider the constraint $avg(S.price) \leq v$ which is neither succinct nor anti-monotone. However, it can be easily seen that every itemset satisfying this constraint will also satisfy the succinct non anti-monotone constraint $min(S.price) \leq v$. The above strategy suggests using the second constraint to try to restrict the size of the generated set by strategy II and then test the generated itemsets for satisfaction of the first constraint. Figure 3.3 gives a high-level description of the *CAP* algorithm.

3.3. Incremental mining of association rules

This section is devoted to the second corner point of the proposed algorithm, namely incremental mining of association rules. First the problem is formally defined and the important theoretical results are presented. Next, the negative border introduced in [45] by Toivonen is formally presented with the proof that it is an indicator for the necessity to check the original database after a database update.

3.3.1 Problem definition

In its first introduction [20], the problem of incremental mining was defined as follows: let L^{DB} be a set of frequent itemsets in a transactional database *DB*. After some update of the database, another set of transactions *db* is added to *DB* (the general problem also includes deletion) to get the updated database *DB*+. With respect to the same *minsup* threshold, it is required to discover L^{DB+} ; the new set of frequent itemsets for the updated database.

In [20], Cheung et al. also stated that maintenance of frequent itemsets involves searching for two kinds of itemsets:

a- Losers: frequent itemsets that became infrequent after adding the increment data to the database.

b- Winners: infrequent itemsets that became frequent after adding the increment data to the database.

Searching for winners is the most difficult of the two problems since it requires rescanning the original database (which is typically orders of magnitude larger than the increment database) to count the support of winners. Several results were presented in the same work that became the ground upon which virtually all approaches to the problem rely. These results serve to limit the search space for the winners to the minimum possible. In the next paragraphs these results are presented without proof.

Lemma 1

An infrequent itemset *S* in *DB* can become a *winner* in the updated database (*DB*+) only if it is frequent in the increment database *db* (i.e. only if $S \in L^{db}$).

Lemma 2

Let *S* be an itemset. If $S \in L^{DB}$ and $S \in L^{db}$ then $S \in L^{DB+}$.

The first lemma is the most important one since it greatly restricts the number of the candidates that need to be checked against the original database, which is the most demanding task of the whole process with respect to time and resources.

3.3.2 Definition of the negative border

The concept of the negative border, as mentioned earlier, was introduced by Toivonen [45] for the purpose of using sampling to generate the frequent itemsets. The same concept was used effectively in the context of incremental mining in [44, 23] to greatly speed-up the process of maintaining frequent sets after a database update. The main contribution in [44, 23] was the proof that the negative border has the property of being an indicator for the necessity of checking the original database for winners. Next, the negative border is formally defined and the proof of its property is given.

Definition 3.5: Negative border (NBd(L)) of a set of frequent itemsets L

Given a collection $L \subseteq P(R)$ of sets (where $P(R)$ is the powerset of some universal set of items R), closed with respect to the set inclusion relation, the negative border $NBd(L)$ of L consists of the set of minimal itemsets $X \subseteq R$ not in L. ❑

Informally, the negative border of a set of frequent itemsets L consists of all the itemsets that are not in L but have all their subsets in L. The intuition behind the concept is that the itemsets in the negative border are the *closest* itemsets that could be frequent, too [45].

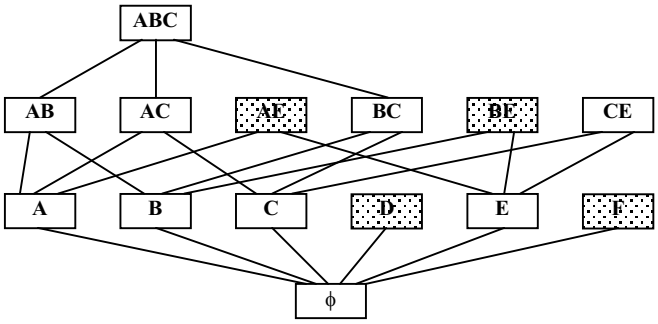

Figure 3.4: The set of frequent itemsets with their negative border (shaded)

With respect to the example database of Figure 3.2, and considering a minimum support threshold of *0.3* (3 transactions in this case), Figure 3.4 illustrates the set of frequent itemsets and their negative border according to the previous definition.

Lemma 3

All 1-itemsets should be present in $L \cup NBd(L)$.

Proof

Since the empty set is frequent by definition, then if a 1-itemset is not in L, all its proper subsets (only the empty set in this case) are frequent, hence it should be in $NBd(L)$ by definition of the negative border. ❑

Theorem 1

Let S be an itemset, $S \in L^{DB+} \,\&\, S \notin L^{DB} \cup NBd\left(L^{DB}\right)$, then there exists an itemset t such that $t \subset S, t \in NBd\left(L^{DB}\right) \& t \in L^{DB+}$. That is some itemset has moved from $NBd(L^{DB})$ to L^{DB+}.

Proof

Since $S \in L^{DB+}$, all possible subsets of S should also be in L^{DB+}. But all the subsets of S cannot be in L^{DB} because if that was the case, then S should be present in at least $NBd(L^{DB})$. By assumption, $S \notin L^{DB} \cup NBd(L^{DB})$. Therefore, there exists an itemset t such that $t \subset S$ and $t \notin L^{DB}$. Now, there are two cases:

Case i: $t \in NBd(L^{DB})$. Thus $t \in L^{DB+}$ and is the required subset which moved from $NBd(L^{DB})$ to L^{DB+} and the theorem is proved.

Case ii: $t \notin NBd(L^{DB})$ which means $t \notin L^{DB} \cup NBd(L^{DB})$ and $t \in L^{DB+}$. Thus the theorem can be recursively applied on t. Note that the size of t is less than that of S. Eventually, either *case i* will be reached or t will be a 1-itemset. By *Lemma 3*, all such itemsets are in $NBd(L^{DB})$ if not in L. Thus the theorem is proved. ❑

The importance of the previous theorem is that it shows that it is necessary for a *winner* itemset to have one of its subsets in $NBd(L^{DB})$ moving to L^{DB+} (i.e. some border itemset will also become a *winner*). Taking the contrapositive of the theorem, one can deduce that if no itemsets in $NBd(L^{DB})$ move to L^{DB+} then there exist no winners. Hence, it suffices to check for the itemsets in the negative border to see if a scan of the original database is really needed to find any new winners.

3.4. Incremental mining of constrained association rules

In this section, the required formal description of the tackled problem is set. Section 3.4.1 defines the problem of incremental mining of constrained association rules in the context of *CFQs*. Next, section 3.4.2 defines the proposed constrained negative border to be used in this context. Then the proof that the new constrained border is a valid indicator for the necessity of checking the original database after adding transactions is given.

3.4.1 Formal problem definition

In the context of constrained frequent-set queries, the problem of incremental mining of association rules can be restated as follows: let L_c^{DB} be a set of frequent itemsets in a database *DB* that satisfies a certain constraint *C* defined through some *CFQ* $\{(S_a, S_c) | C\}$. After some update of the database, another set of transactions *db* is added to *DB* to get the updated database *DB+*. It is required to discover the new set L_c^{DB+} of frequent itemsets that satisfies the same constraint *C* in the updated database.

As mentioned before, the concept of the negative border proved useful in the problem of incremental mining of association rules. The challenge is to try to adapt the concept of negative border for *CFQs*. It can be noticed that the negative border as defined before represents the candidate itemsets counted by the *Apriori* algorithm that turned to be not frequent. This may give an insight to use the same sets counted by the *CAP* algorithm but found small, as the new negative border. This leads to the following definition of the new negative border which is named the *constrained negative border*.

3.4.2 The constrained negative border

Definition 3.6: Constrained Negative Border CNBd(L_c) of a set of constrained frequent itemsets L_c

The constrained negative border of a set of frequent itemsets L_c, satisfying a constraint *C*, henceforth referred to as *CNBd(L_c)* is defined as follows:

$$CNBd(L_c) = \{S \mid S \text{ is an itemset satisfying only } C - C_{freq} \;\&\; S' \subset S \Rightarrow S' \in L_c \text{ or } S' \text{ does not satisfy } C_{succ}\}$$

❑

Informally, the constrained negative border consists of all the itemsets that are not frequent but have all their subsets (of which their frequencies are known) frequent. The reason for our uncertainty is the generate-only paradigm of succinct constraints which leaves us with no knowledge of the frequency of itemsets which do not satisfy the constraint since they are not generated from the first place. Just like strategy II of the *CAP* algorithm, if an itemset in the border has such a subset, it is given the merit of doubt by assuming it to be frequent.

Given that the normal negative border was an indicator for the necessity of checking the original database for winners, it would seem natural if the constrained negative border would play the same role. The original negative border possessed the property of having all its members minimally small, meaning that all their *proper* subsets are large. In the context of *CFQs*, the same concept can be generalized to include all anti-monotone constraints. But since succinct constraints do not posses the same closure property, it is not guaranteed that all proper subsets satisfy the constraint. The following lemmas help in better understanding of the succinct constraints. They clear the way for the proof of the next theorem which states that the constrained negative border is still indeed an indicator for checking the original database.

Lemma 4

Let C be a succinct constraint with an *MGF* M defined as follows: $M = \left\{X_1 \cup ... \cup X_n \middle| X_i \subseteq \sigma_{p_i}(Item), 1 \le i \le n \;\&\; \exists k \le n : X_j \ne \phi, 1 \le j \le k\right\}$. It is assumed, without loss of generality, that the satisfying sets of the selection predicates (σ_{p_i}'s) of M are all disjoint for $1 \le j \le k$.

Let S be an itemset satisfying C. It follows that $|S| \ge k$ where k is as defined in M.

Proof

Since S satisfies C, S is generated by M, then $S \cap \sigma_{p_j}(Item) = X_j \ne \phi, 1 \le j \le k$. Since all X_j's are non-empty and disjoint, then $|t| \ge k \;\&\; t \subseteq S$, *where* $t = X_1 \cup ... \cup X_k$. This implies our result and completes the proof. ❑

In other words, k is the lower bound of the size of any itemset that satisfies the succinct constraint. If the X_j's were not disjoint, this means that there might be an item included in the satisfying set of two or more selection predicates. We then have a lower bound than k for the size of an itemset satisfying a succinct *constraint*. This lower bound, m, is between 1 and k. This will not affect the result of the lemma.

To see the result of this lemma, consider the following simple example. Referring to the example database of Figure 3.2, let *C* be the succinct constraint $S.Type \supseteq \{snack, soda\}$. The *MGF* of this constraint is:

$$M = \{X_1 \cup X_2 \cup X_3 \mid X_1 \subseteq \sigma_{p_1}(Item), X_2 \subseteq \sigma_{p_2}(Item), X_3 \subseteq \sigma_{p_3}(Item), X_1 \neq \phi, X_2 \neq \phi\}$$

where $p_1 = (Type = 'soda')$, $p_2 = (Type = 'snack')$, and $p_3 = (Type \neq 'soda' \wedge Type \neq 'snack')$. The value of k here is 2. The lemma simply states that any itemset satisfying the constraint should have *at least* 2 items. This is clear since a satisfying itemset must contain both a 'snack' and a 'soda' item (this is of course assuming no single item is *both* a 'snack' and a 'soda' item at the same time). For the example database, those minimum sized itemsets satisfying the constraint are the itemsets *BC* and *BF*.

Lemma 5

Assuming k is as defined in the *MGF* of *Lemma 4*, all itemsets of size k satisfying $C\text{-}C_{freq}$ are in $L_c \cup CNBd(L_c)$. Furthermore, all itemsets of size $k+1$ having all their k-sized subsets which satisfy C_{succ} in L_c are in $L_c \cup CNBd(L_c)$.

Proof

Assume S is an itemset of size k satisfying $C\text{-}C_{freq}$. If S satisfies C_{freq} then $S \in L_c$, else $S \notin L_c$ and by *Lemma 4* no proper subset of S satisfy C_{succ} which implies that $S \in CNBd(L_c)$ by definition.

Similarly, assuming S is an itemset of size $k+1$ satisfying $C\text{-}C_{freq}$ and all its k-sized subsets which satisfy C_{succ} are in L_c (notice that by *Lemma* 4, all other proper subsets of S do not satisfy C_{succ}). If S satisfies C_{freq} then $S \in L_c$, else $S \notin L_c$ and for all $S' \subset S$ & S' satisfies C_{succ}, $S' \in L_c$ which implies that $S \in CNBd(L_c)$ by definition. This completes the proof. ❑

To see this by example, consider the same constraint $S.Type \supseteq \{snack, soda\}$. As discussed after *Lemma* 4 the set of itemsets of size 2 satisfying the constraint is {BC, BF}. It is clear that both itemsets should be in $L_c \cup CNBd(L_c)$. Now, consider the itemset *ABC*. It sure satisfies the constraint, however, all its proper subsets except *BC* do not satisfy the constraint. Hence, if *BC* is in L_c and *ABC* is not frequent, then the definition of the constrained negative border will apply to it. Hence, *ABC* is in $L_c \cup CNBd(L_c)$.

The previous lemma sets a lower bound on the size of itemsets in the constrained negative border, much as *Lemma 3* sets a lower bound for the size of itemsets in the negative border.

Lemma 6

Let C be a succinct constraint with an MGF M similar to that defined in *Lemma 4*, and let S be an itemset satisfying C. If $|S| > k$, where k is as defined in M, then $\exists\, t \subset S$ such that t satisfies C.

Proof

Keeping the same assumption, that the satisfying sets of the selection predicates (σ_{p_i}'s) of M are all disjoint for $1 \leq j \leq k$, let $X_j = S \cap \sigma_{p_j}(Item), 1 \leq j \leq k$. From the above assumption and since S satisfies C, it follows that all the X_j's are non-empty and disjoint. Now, there are two cases:

1. $\forall X_j$, $|X_j| = 1$ (i.e. S contains only one item from every mandatory set). Let $t = X_1 \cup \ldots \cup X_k$. We have $|t| = k$ and $t \subseteq S$. Clearly, t is generated by M and hence satisfies the constraint C and since $|S| > k$ then $t \subset S$. This gives the stated result in the lemma.
2. $\exists X_m$ such that $|X_m| > 1$, $1 \leq m \leq k$. Now, let X_m' be a proper subset of X_m such that X_m' is non-empty, clearly such a proper subset exists. Let $t = X_1 \cup \ldots \cup X_m' \cup \ldots \cup X_k$. We have $t \cap \sigma_{p_j}(Item) = X_j \neq \phi, 1 \leq j \leq k \text{ and } j \neq m, t \cap \sigma_{p_m}(Item) = X_m' \neq \phi$ then t is generated by M and t satisfies C. Also, since $t \cap S = t$ and $X_m \subset S$ and $X_m \not\subset t$ then $t \subset S$.

This completes the proof. ❑

Informally, the previous lemma states that any itemset satisfying the succinct constraint with a greater cardinality than the minimum cardinality set by *Lemma 4* will have a *proper* subset of it that also satisfies the constraint.

To see this by example, consider the same constraint and example database. We know that the minimum sized itemsets satisfying C are {BC, BF}. Consider the itemset *ABC*, it has exactly one soda item, '*B*', and one snack item, '*C*'. Then, it has the itemset set *BC* as the required proper subset satisfying the constraint. Furthermore, consider the itemset *BCF* which also satisfies the constraint. Here, there are more than one item of type 'snack', namely '*C*' and '*F*'. This makes both *BC* and *BF* qualify as the required proper subsets.

The previous results clear the road for proving the next theorem.

Theorem 2

Let S be an itemset, $S \in L_c^{DB+}$ *and* $S \notin L_c^{DB} \cup CNBd\left(L_c^{DB}\right)$, then there exists an itemset t such that $t \subset S$ *and* $t \in CNBd\left(L_c^{DB}\right)$.

Proof

Let the *MGF* of C_{succ} be as defined in *Lemma 4*. Let t be an itemset such that $t \subset S$. Since $S \in L_c^{DB+}$ we know that t must satisfy C_{freq} and C_{am} of C in the updated database (by definition of anti-monotonicity). By *Lemma 5* and *Lemma 6* we know that there exists at least one subset t of S that satisfies C_{succ} of C (since S satisfies C_{succ} and $S \notin L_c^{DB} \cup CNBd\left(L_c^{DB}\right)$ it follows that its cardinality is greater than the minimum cardinality). Let t be such a subset, then t satisfies C and hence $t \in L_c^{DB+}$. By assuming that $S \notin L_c^{DB} \cup CNBd\left(L_c^{DB}\right)$, then there exists a proper subset t of S such that t satisfies C_{succ} and $t \notin L_c^{DB}$, otherwise, if all subsets of S satisfying C_{succ} were in L_c^{DB}, then S should have been in $CNBd\left(L_c^{DB}\right)$ by definition. We choose t also to be such a subset. Now we have $t \subset S$, $t \in L_c^{DB+}$, and $t \notin L_c^{DB}$. There are two cases:

Case i: $t \in CNBd\left(L_c^{DB}\right)$, which means that our theorem is proved.

Case ii: $t \notin CNBd\left(L_c^{DB}\right)$. Therefore, we have $t \in L_c^{DB+}$ and $t \notin L_c^{DB} \cup CNBd\left(L_c^{DB}\right)$ and we can recursively apply the theorem. Eventually we will either encounter *case i* or we will have $|t| = k$. In the later case, by *Lemma 5*, all such itemsets are in the constrained border if they are not frequent. This completes the proof. ❑

The previous theorem proved that the constrained negative border is playing the same role played by the normal negative border in [44]. This result will be employed in the proposed algorithm *ICAP*. *ICAP* uses the *CAP* algorithm to discover the initial set of constrained frequent itemsets along with its constrained negative border and assumes that the support count of each itemset in both sets is kept in the database. The next chapter is devoted for the detailed presentation of *ICAP*.

Chapter 4

THE PROPOSED ALGORITHM *ICAP*

This chapter is devoted for the presentation of the proposed algorithm *ICAP*. *ICAP* is built upon the foundations and results discussed in the previous chapter. It uses the constrained negative border to decide when to check the original database after a database update. As a result, *ICAP* performs at most one scan of the original database only if it is *absolutely* necessary.

The rest of the chapter is organized as follows: section 4.1 gives a high level description of the algorithm. Computing the constrained negative border turns out to be not as straightforward as it was in the case of the original negative border, thus, section 4.2 discusses how to compute it. Section 4.3 formally proves the correctness of the algorithm. Next, a descriptive example is given to demonstrate the relative merits of the proposed algorithm against the naïve alternative of rerunning the algorithm *CAP* on the updated database from scratch. Section 4.4 discusses the question of when to use *ICAP*. Finally, section 4.5 touches upon some of the important implementation details.

4.1. Algorithm description

Before proceeding with the description of the algorithm it should be noted that only updates of the transactional database is taken into consideration (i.e. no changes occur in the data describing the items). Therefore, it should be clear that adding and removing transactions can only affect the support of an itemset and has no effect on the satisfiability of C-C_{freq}. Formally, if an itemset S satisfied C-C_{freq}, before the database update, S will satisfy C-C_{freq} after the database update. The algorithm uses as input the set of constrained frequent sets L_c^{DB} of the original database *DB* and the constrained negative border $CNBd\left(L_c^{DB}\right)$ discovered using the algorithm *CAP* along with the frequencies of the itemsets in both sets.

The proposed algorithm can be summarized in the following steps:

1. Use algorithm *CAP* to discover all frequent itemsets following the constraint in the increment database *db* (see *section 3.2.2* for a description of the *CAP* algorithm).

2. Count the frequencies of the itemsets in the previously discovered set L_c^{DB} along with the constrained negative border $CNBd\left(L_c^{DB}\right)$ in the increment database *db* to discover the losers and those itemsets which will remain frequent.
3. If no new itemsets qualify as candidate frequent itemsets (i.e., no new frequent itemsets are discovered in the increment database *db* and candidates in the negative border remain infrequent), then there is no need to rescan the original database. The new negative border can be recomputed as in [44] (only if losers exist).
4. If there are new qualifying candidates, generate the negative border closure in the same manner as in [44] and count the candidates in this closure against the original database.

It is clear from the description of *ICAP* that it requires at most one scan of the original database (done in step 4) only if the database insertions cause the constrained negative border to expand. Expansion of the border means that there are potential itemsets that can become winners other than those in $L_c \cup CNBd(L_c)$. This happens if there exist winners that can be joined with other frequent itemsets or that can allow other itemsets to be candidates for testing. To see this, consider the following example. Assuming the set of items is {A, B, C, D}. After initial investigation of *DB*, itemsets *AB, AC, BC* were found frequent and *ABC* was found infrequent and hence was in the constrained border. After adding *db*, assume that *ABC* became a winner. Although such a movement might seem to necessitate a check for the original database *DB*, another look will reveal that this is not important. This is because moving *ABC* from the negative border to the set of frequent itemsets cannot induce any more itemsets since it would be the only frequent itemset of size 3 (the border does not *expand*). On the other hand, if *BC* was not frequent and became frequent after adding *db*, this might indicate that *ABC* is also a winner (the border *expands*).

The fact that *ICAP* can exploit and use the constrained negative border makes it superior compared to the classical level-wise algorithms. A high-level description of the proposed algorithm *ICAP* is shown in Figure 4.1.

The operations of computing and maintaining the constrained negative border are not as straightforward as those for the negative border in [44]. The next section describes the problems faced when trying to compute the constrained negative border and explains how it can be generated.

Function ICAP(L_c^{DB} , $CNBd\left(L_c^{DB}\right)$, *db*)

1 Compute L_c^{db} using *CAP*

2 Count support of all items of L_c^{DB} and $CNBd\left(L_c^{DB}\right)$ in *db*.

3 $L_c^{DB+} = \Phi$

4 **For each** itemset *s* in $L_c^{DB} \cap L_c^{db}$ **do** //itemsets still frequent

5 $t_{DB+}(s) = t_{DB}(s) + t_{db}(s)$ //calculate new support count

6 $L_c^{DB+} = L_c^{DB+} \cup \{s\}$

7 **For each** itemset *s* in $CNBd\left(L_c^{DB}\right) \cap L_c^{db}$ **do** //cand. of the border

8 $t_{DB+}(s) = t_{DB}(s) + t_{db}(s)$ //calculate new support count

9 **If** $t_{DB+}(s) \geq minsup * |DB+|$ **then**

10 $L_c^{DB+} = L_c^{DB+} \cup \{s\}$

11 **For each** itemset *s* in $\left(L_c^{DB} - L_c^{db}\right)$ **do**

12 **If** $t_{db}(s) + t_{DB}(s) \geq minsup * |DB+|$ **then**

13 $L_c^{DB+} = L_c^{DB+} \cup \{s\}$

14 **If** $L_c^{DB+} \neq L_c^{DB}$ **then**

15 $CNBd\left(L_c^{DB+}\right) = constrained - negativeborder - gen\left(L_c^{DB+}, C\right)$

16 **Else** $CNBd\left(L_c^{DB+}\right) = CNBd\left(L_c^{DB}\right)$

17 **If** $L_c^{DB} \cup CNBd\left(L_c^{DB}\right) \neq L_c^{DB+} \cup CNBd\left(L_c^{DB+}\right)$ **then** *//is a scan of DB necessary?*

18 $S = L_c^{DB+}$

19 **Repeat**

20 $S = S \cup constrained - negativeborder - gen(S, C)$

21 **Until** *S* does not grow

22 **For each** itemset *s* in $S - L_c^{DB+}$

23 *//count candidate itemsets against the updated database*

24 **If** $t_{DB+}(s) \geq minsup * |DB+|$ **then**

25 $L_c^{DB+} = L_c^{DB+} \cup \{s\}$

26 $CNBd\left(L_c^{DB+}\right) = constrained - negativeborder - gen\left(L_c^{DB+}, C\right)$

27 **Return** $L_c^{DB+} \cup CNBd\left(L_c^{DB+}\right)$

Figure 4.1: A high level description of the algorithm *ICAP*

4.2. Computing the constrained negative border

The original border in [44] was easily produced and maintained as a by product of the *Apriori* algorithm since it represents all candidate itemsets that did not pass the counting test, which means that those itemsets are counted by the algorithm anyway.

The constrained negative border, on the other hand, does not posses the same property. According to *Lemma 5*, all itemsets of minimum size that satisfy C-C_{freq} should be in the constrained border. *CAP*, however, does not produce, and hence does not count, all those itemsets since it guarantees that all generated itemsets satisfying C-C_{freq} of minimum size, will have all their subsets satisfying C-C_{succ} (*strategy II, section 3.2.2*). It can also be noticed that, assuming the minimum size of itemsets satisfying C-C_{freq} is k, all itemsets of size $k+1$, having all their k-size subsets satisfying C_{succ}, in L_c should also be in the constrained border. *CAP*, on the other hand, only counts a subset of such itemsets (*strategy II, section 3.2.2*).

To see this, consider the example database of Figure 3.2 and the succinct constraint $S.Type \supseteq \{snack, soda\}$. From the previous discussions it is known that any generated itemset should contain at least one item from the set {B} and at least one item from the set {C, F} and optionally one or more other items from the whole universal set of items. Assume that the set of frequent items was {A, B, C}. According to strategy II of CAP, only the itemsets BC and ABC will be generated and counted. However, *BF, BCD, BCE* and *BCF* should also be counted, since they are by definition in the constrained border (*Lemma 5*).

To work around this problem, CAP should be modified to generate and count all the candidates belonging to the constrained negative border. This can be done as follows:

Assuming C_{succ} is as defined in *strategy II* of *CAP* (*section 3.2.2*) and observing that such definition makes the minimum size of an itemset, satisfying C-C_{freq}, equal one (i.e. $k=1$), strategy II should be as follows:

a- Define $C_1^c = \sigma_{p_1}(Item)$ and $C_1^{\neg c} = \sigma_{p_2}(Item)$. Define corresponding sets of frequent sets of size 1: $L_1^c = \{e \mid e \in C_1^c \ \&\ freq(e)\}$, and $L_1^{\neg c} = \{e \mid e \in C_1^{\neg c} \ \&\ freq(e)\}$.

b- Add $C_1^c - L_1^c$ to $CNBd(L_c)$.

c- Define $C_2 = L_1^c \times L_1^c \cup C_1^{\neg c}$, and L_2 the set of frequent itemsets in C_2.

d- Add $C_2 - L_2$ to $CNBd(L_c)$.

e- Do step 3 of strategy II with no modification.

The process of re-computing the constrained negative border, given a set of constrained frequent-sets L_c and a constraint C, is done by the function *constrained-negativeborder-gen*(L_c, C). Figure 4.2 gives a high-level description of this function.

Function Constrained-negativeborder-gen(L_c, C)

1 Split L_c into L_m, L_{m+1}, … , L_n where L_k is the set of constrained frequent itemsets in L of size k, n and m are the size of the smallest and the largest sets of frequent itemsets in L_c respectively.

2 Use the modified strategy II in section 4.2 to generate all candidates of size m, C_m *(using step a)* and all candidates of size $m+1$, C_{m+1} *(using step c)*, according to the constraint C.

3 $CNBd(L_c) = (L_m - C_m) \cup (L_{m+1} - C_{m+1})$

4 **For** k = m+1 to n-1 **do**

5 Use *CAP* to generate C_{k+1} from L_k

6 $CNBd(L_c) = CNBd(L_c) \cup (C_{k+1} - L_{k+1})$

7 **Return** $CNBd(L_c)$

Figure 4.2: A high level description of the function *constrained-negativeborder-gen*

4.3. Correctness proof

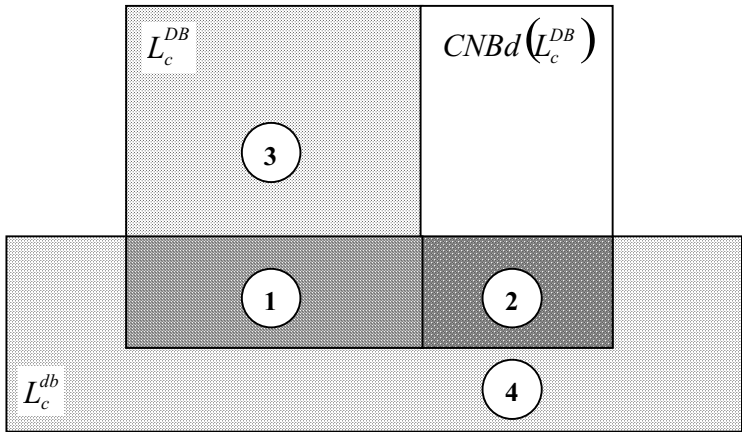

Figure 4.3: A Venn diagram of the possible intersections of L_c^{DB}, $CNBd(L_c^{DB})$, and L_c^{db}

In this section, we will prove that *ICAP* is sound and complete; it discovers all and only those itemsets that are frequent in *DB+* and satisfying the *CFQ*. We rely on the correctness of *CAP* proved in [35].

Theorem 3

Algorithm *ICAP* is sound and complete with respect to counting all constrained frequent itemsets in the updated database *DB*+.

Proof

In Figure 4.3 - which represents all the possible intersections of L_c^{DB}, $CNBd\left(L_c^{DB}\right)$, and L_c^{db} - we have four shaded regions, representing all the possible candidates for inclusion in L_c^{DB+} (*Lemma 1*, *section 3.3.1*). Regarding the steps of algorithm *ICAP* shown in Figure 4.1, step 1 computes L_c^{db} using *CAP* and should be correct by the correctness of *CAP*. Steps 4-6 add itemsets in *region 1* of Figure 4.3 to L_c^{DB+} after updating their support count. By *Lemma 2* we know that all those itemsets are in L_c^{DB+}. Steps 7-10 check candidates in *region 2*, those are candidates of the negative border. If an itemset is found to be frequent in *DB*+, it is directly considered frequent. Steps 11-13 check the candidates in *region 3* by scanning the increment database. Now we have only *region 4* remaining to check. By *Theorem 2* we know that there is no need to do that if the constrained border does not expand. Steps 14-17 check for the negative border expansion by observing the change in $L_c^{DB} \cup CNBd\left(L_c^{DB}\right)$. If an expansion occurs, then we need to scan *DB* for candidates in region 4. Steps 18-21 generate the constrained negative border closure of items in L_c^{DB+} discovered so far. Steps 22-25 count the candidates against *DB* to discover winners. Finally, step 26 generates the constrained negative border of the complete L_c^{DB+}. Since the candidates counted by the steps of *ICAP* are all and only those candidates that need counting, it follows that *ICAP* is sound and complete.

❑

Example:

Consider a transactional database as depicted in Figure 3.2. Let the first 7 transactions be the original database *DB* and the last 3 transactions be the increment database *db*. Consider the following *CFQ*:

$$\left\{X \middle| X.Type \supseteq \{soda, snack\} \,\&\, min(X.price) \geq 20 \,\&\, minsup = 0.2\right\}$$

The algorithm *ICAP* is now applied on the database to find the updated constrained frequent sets. It is assumed that the *CAP* algorithm has been applied on the original database *DB*. Thus

the set of constrained large frequent set $L_c^{DB} = \{BC\}$ and the constrained negative border $CNBd(L_c^{DB}) = \{BF, BCD, BCE\}$. Initially, *CAP* is run on *db* to get the set $L_c^{db} = \{BC, BF, BCF, BCD, BCE, BCDE\}$. Referring to Figure 4.3, the sets of itemsets $\{BC\}$, $\{BF, BCD, BCE\}$, and $\{BCF, BCDE\}$ correspond to those in regions 1, 2, and 4. Region 3 is empty since the only frequent itemset in *DB* is also frequent in *db*. Now, *BC* is frequent in the updated database. Checking items in the negative border, we have *BCE* only moving from the negative border to the updated set of frequent itemsets. Before checking for the itemsets in region 4, we check the constrained negative border expansion. Since, moving *BCE* into the frequent sets does not expand the negative border, we conclude that there is no need to count the candidates in region 4 in *DB*. Thus, the final updated constrained frequent itemsets $L_c^{DB+} = \{BC, BCE\}$ and the new constrained negative border $CNBd(L_c^{DB+}) = \{BF, BCD\}$. ❑

As can be seen from the previous example, *ICAP* did not need a scanning of the original database. This is a great improvement over rerunning the *CAP* algorithm on the original database in terms of the number of transactions read. It would be natural to assume that a more drastic improvement gain will be obtained for typical sizes of databases and database increments or updates.

4.4. When to use *ICAP*?

At this point, it is useful to stop to answer the question of whether it is always useful to run the incremental algorithm. This has been the subject of some previous research, e.g. [30], and it is still valid in the context of this work to ask the same question.

Using the negative border can be considered itself as a measure of the necessity to update the set of frequent itemsets. However, a couple of *winner* itemsets will trigger the expansion of the border. In practical cases, such couple of winners may not represent a great significance to the user that warrant the time spent in the algorithm. More methods are needed to measure such significance. Another aspect of this issue is that sometimes the incremental algorithm may do an unnecessary effort as when it does when many itemsets are large in the increment but there are no winners. This is best demonstrated if we assume that the incremental algorithm is used with every transaction update and the next added transaction has 20 items in it. For any support threshold whatsoever, all subsets of the set of itemsets in this transaction will be frequent in the increment which results in the tremendous 2^{20} frequent itemsets in the increment!! Since adding only one transaction to the original database is not

likely to modify the frequent itemsets and hence expand the border, the effort done by the algorithm will be unnecessarily wasted.

Some alternatives exist to measure the significance of the change in the frequent itemsets (if one exists) and when to use the incremental algorithm:

a- Use the sampling technique reported in [30] to *approximately* measure the difference between the new and the old set of frequent itemsets. If the difference exceeds a user defined threshold, it would warrant using the incremental algorithm. The disadvantage of using this technique is that drawing the sample would require a pass on the original database and this pass is the maximum price that the incremental algorithm pays to get the *exact* set of new frequent itemsets. Although one could argue that the computational effort in the pass to compute the exact set is far more than that needed to draw a random sample.

b- It might be more useful to get some insight about the difference in frequent itemsets by looking more closely at the number of winners among the border itemsets and the relation of those winners with the number of frequent itemsets in the increment. A great number of winning border itemsets might indicate a great change in the number of new frequent itemsets. Also, a border itemset that has many supersets among the frequent itemsets in the increment might also indicate a great change in the number of new frequent itemsets.

c- As for avoiding the unnecessary effort done by the incremental algorithm, it might be useful to start incrementally mining when the size of the increment exceeds a certain percentage of the original database (or if the dimension of *data span* [24] is used, when a new *block* of data is added). This, however, has the disadvantage that if the data in the increment has the same characteristics as the data in the original database, the discovered patterns will remain almost unchanged regardless of the size of the increment.

4.5. Implementation details

This section discusses some of the implementation details that are worth noting. In subsection 4.5.1, the choice of the main data structures used in the implementation of *ICAP* is justified and in subsection 4.5.2, some of the important notes on the implementation of the *CAP* algorithm is mentioned.

4.5.1 Choice of data structures

An important decision in implementing *ICAP* was the choice of the data structure used to represent the set of itemsets. It should be well noted that tremendous performance gains are achieved by a well choice of this data structure [8, 18, 33, 27, 2]. Two different classical structures are suggested in the literature to represent the set of itemsets; the *hash tree* [8] and the *prefix tree* [18, 33] that are suitable for the level-wise method of the *Apriori* and *CAP* algorithms. The later (*prefix tree*) was chosen. The prefix tree is a *trie* that holds the itemsets in an implicit form. An itemset is merely a traversal along the edges of the tree. The node of the tree that is reached after this traversal itemset holds the needed auxiliary information about the itemset (e.g. the frequency count). Figure 4.4 Represents a prefix tree for the powerset of the set of itemsets {A, B, C, D}. The prefix tree has the same advantage of the *hash tree* data structure in counting the candidate itemsets found in a transaction. The prefix tree has, however, the advantage of being able to hold both the candidate and frequent itemsets in one structure without the need to resort to other data structures. Moreover, it has an obvious performance gain when performing the candidate generation phase (*prefix join* [8, 33]) of the level-wise method which is one of the most demanding operations in the whole mining process.

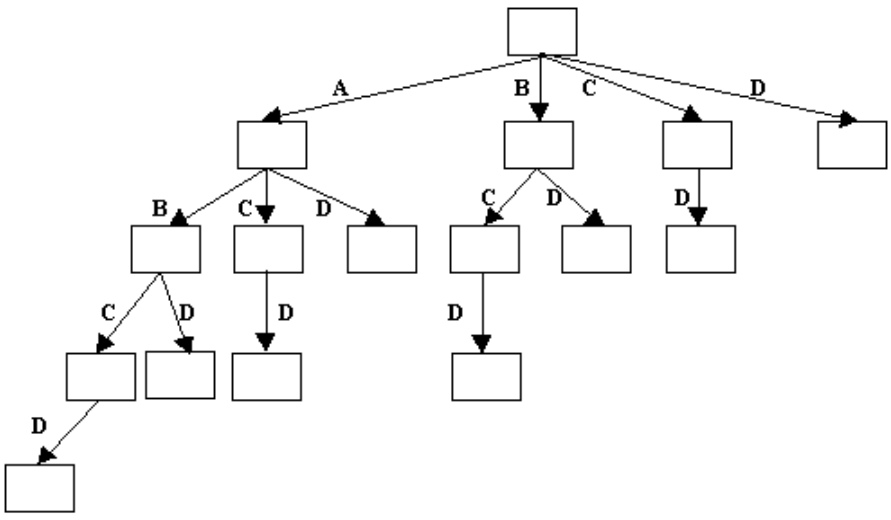

Figure 4.4: A prefix tree

The implementation used for the prefix tree structure is following the one reported in [33].

Figure 4.5 represents this implementation. A hash table is used to facilitate access to the edges emanating from a node in the tree.

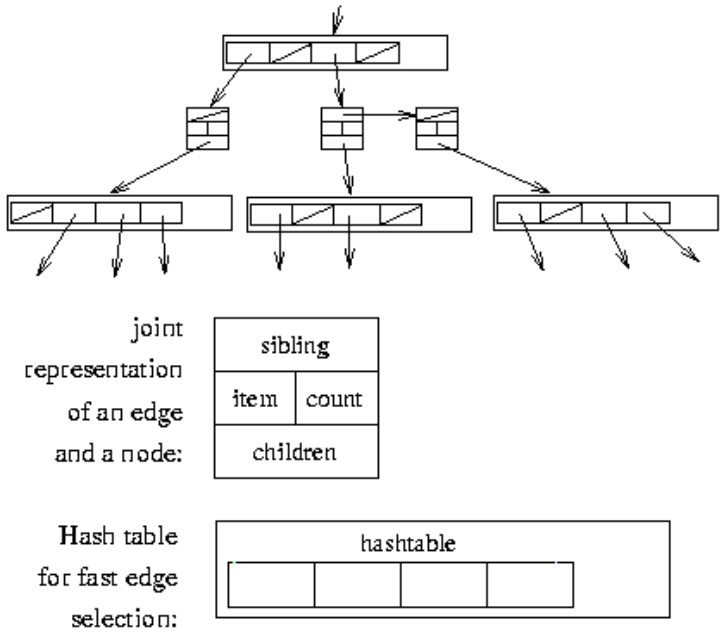

Figure 4.5: The implementation of prefix tree [33]

4.5.2 Notes on the implementation of *CAP*

In the course of implementing *ICAP*, it was mandatory to implement the *CAP* algorithm in its original form reported in [35] and with the modified strategy to generate the negative border (see section 4.2).

An important note was found during the implementation of *CAP* that is worthy of mentioning:

Following strategy II of *CAP* (section 3.2.2) will destroy the natural ordering of the itemsets that is based on the item identifier. This ordering is very important with regard to both the construction of the tree structure holding the itemsets and the subset counting function. This is best illustrated by the following example:

Example:

Assume the succinct constraint in [35], $S.Type \supseteq \{soda, snack\}$, which is the same constraint described after *Lemma 4*. Then the sets X_1, X_2, and X_3 will be as defined in the example.

Assume the set items are {A, B, C, D, E, F} and assume:

$C_1^{X_1}$ = {A, C, E} (soda), $C_1^{X_2}$ = {B, F} (snack) and $C_1^{X_3}$ = {D} (not snack or soda)

Furthermore, assume all items are frequent, meaning that:

$$L_1^{X_1} = C_1^{X_1},\ L_1^{X_2} = C_1^{X_2} \text{ and } L_1^{X_3} = C_1^{X_3}$$

Now, $C_2 = L_1^{X_1} \times L_1^{X_1}$ = {AB, AF, CB, CF, EB, EF}. Assume all of them are also frequent for the sake of demonstration, meaning that $L_2 = C_2$.

Now, generating $C_3 = L_2 \times (L_1^{X_1} \cup L_1^{X_2} \cup L_1^{X_3})$ as reported in [35] will yield several *spurious* itemsets as follows:

C_3 = {ABC, ABD, ABE, ABF, AFB, AFC, AFD, AFE, CBA, CBE, CBF, CFA, CFB, CFD, CFE, EBA, EBC, EBD, EBF, EFA, EFB, EFC, EFD}

As can be seen, all the double underlined itemsets are merely repetitions, so there must be a method to check for them (besides checking for repeating an item in an itemset). This mandates using a separate structure for storing the candidate itemsets upon generation in their natural order (according to their IDs) or in any other fixed ordering that is used as a reference for pruning these spurious itemsets.

Furthermore, when checking to see if the itemset *ABE* for example is frequent in the regular method, it can be seen that its 2-sized subset *BE* satisfies the constraint but cannot be found in L_2 since it is stored as *EB*.

If it is assumed that order will be preserved, another severe problem rises. Such assumption will conflict with the candidate generation phase of the *Apriori* algorithm which is the proposed method for candidate generation in the *CAP* algorithm for later steps. To see that, suppose that the two itemsets in C_3, {EBF, EBD} turned out to be frequent. In generating C_4 both itemsets should be joined to get the itemset EBFD since they both have the same prefix; *EB*. Storing itemsets ordered will prevent this since the two generating itemsets will become {BEF, BDE} and they do not have the same 2-prefix, so the candidate generation phase will not generate them. ❑

To avoid this problem in the proposed algorithm, itemsets were generated non-ordered (as implied by the strategies of *CAP*) and a separate structure of ordered itemsets was used to prune the spurious candidates.

Chapter 5

PERFORMANCE STUDY

This chapter is devoted for the discussion of the different experiments conducted to measure the relative performance of the proposed incremental algorithm *ICAP* for mining constrained association rules compared to the single alternative approach available so far, which is rerunning the *CAP* algorithm from scratch on the updated database.

Section 5.1 starts by describing the environment in which the performance experiments were conducted including the testing machine and the data set. It also discusses the performance measures and parameters and justifies their usage. Section 5.2 describes the basic set of experiments which measure the performance of proposed the algorithm relative to *CAP* for different increment sizes and for different types of queries. Section 5.3 presents the results of the experiments measuring the sensitivity of the algorithm to the change in the database size. Finally, section 5.4 comments on the experiments measuring the sensitivity of the algorithm upon changing the item selectivity.

5.1. The experimental environment and performance parameters

The algorithm was implemented and tested on an IBM compatible PC with a Pentium II® 300 processor and 96 MB of main memory running the Microsoft Windows 95® operating system. The program was the only major job running on the machine throughout all the experiments to achieve a fair environment for comparison.

As for the test data, the program developed in IBM Almaden research center was used to generate the test data. This is available from the IBM QUEST web site (http://www.almaden.ibm.com/cs/quest). The program generates a synthetic transactional database using several parameters (see [8] for an explanation of every parameter and the rationale behind the method of generating the database). Since its introduction, this source has been used in virtually every publication on mining association rules.

Number of Transactions in the updated database	400,000 (in the basic set of experiments)
Number of Items	1000
Average Transaction Size	10
Average size of maximal potentially large itemsets	6

Table 5.1: Parameter Settings

The parameter settings used for the experimental database are listed in Table 5.1. Such parameters are assumed to mimic a reasonable retailing environment. They also give the chance to explore the relative merits of the incremental algorithm. The standard encoding used to name a certain database that is used in [8] is also used here. In this encoding, the name T*x*.I*y*.D*z*K is used to name a transactional database for which its average transaction size is x, its average size of frequent itemset in a transaction is y, and the database size is z thousands of transactions. As for the descriptive data about the items (the *ItemInfo* table), an integer identifier ranging from 0 to 999 is used to represent the items. The item *Type* was chosen among four different types (soda, snack, juice, dairy) by tossing a weighted 4-sided coin. Each side represents an item type and the weight associated with each side represents the percentage of items having this type. The item price was uniformly chosen between 10 and 100. When generating the increment database, the method in [20, 39, 44] was used. In this method, the increment database is taken as a contiguous block from a generated set representing the whole updated database *DB*+. The increment size x% means that (|*DB*+|-x%) of the database is considered as *DB* and the remaining x% is considered as *db*. The increment database was taken from the first x% transactions of the database.

The performance measure that is used throughout all the experiments is the *speedup*. Speedup is measured as the ratio between the running time of the *CAP* algorithm to that of the *ICAP* algorithm for the same setting of parameters. The use of this measure has been the standard in the literature of incremental mining algorithms [20, 21, 39, 44]. Speedup is more significant than reporting the running time of the algorithms since such times are dependent on the different implementations and test beds.

For the system parameters, the main observed parameters are the different query types, the increment size as a ratio of the original database size and the support threshold. The first set of experiments measure the relative performance of the algorithm for different increment sizes and its scalability when decreasing the support threshold. Such measurement is done for every type of constraint. The experiments also test the scalability of the algorithm with the

size of the updated database. Finally, it is expected that the generation of the negative border incurs an overhead when succinct constraints exist, especially when the selectivity of the succinct constraint increases. Therefore, another set of experiments is performed to test the sensitivity of *ICAP* to the increase in item selectivity when a succinct constraint is used.

5.2. Measuring performance for different increment sizes and different query types

As discussed in [35], there are four different categories of constraints; namely succinct only, anti-monotone only, succinct anti-monotone and non-succinct non-anti-monotone constraints. Representative queries of each of the four categories were used in the first set of experiments. The following set of figures (Figure 5.1 to Figure 5.4) represent the speedup of *ICAP* for these different types of constraints, the speedup is measured for different increment sizes (measured as a percentage from the size of the original database). Figure 5.1 shows the results when using the succinct only constraint $S.Type \supseteq \{soda\}$. Figure 5.2 shows the results when using the succinct anti-monotone constraint $min(S.Price) \geq 60$. Figure 5.3 uses the anti-monotone constraint $sum(S.Price) \leq 50$. To test the non-succinct non-anti-monotone category of constraints, the constraint $avg(S.Price) \leq 15$ was used. Finally, Figure 5.4 tests the speedup when using a hybrid constraint combining both the succinct constraint $S.Type \supseteq \{soda, snack\}$ and the succinct anti-monotone constraint $min(S.Price) \geq 20$. The selectivity of the *soda* items and *snack* items were 5% each throughout all experiments. The previous constraints mimic those reported in [35] for which the *CAP* algorithm performed well.

The range of minimum support used is from 1% to 0.2%, which is the standard significant range for this type of database. Below this range, the number of frequent itemsets is very small to give any meaningful comparison and above this range the number of frequent itemsets increases rapidly.

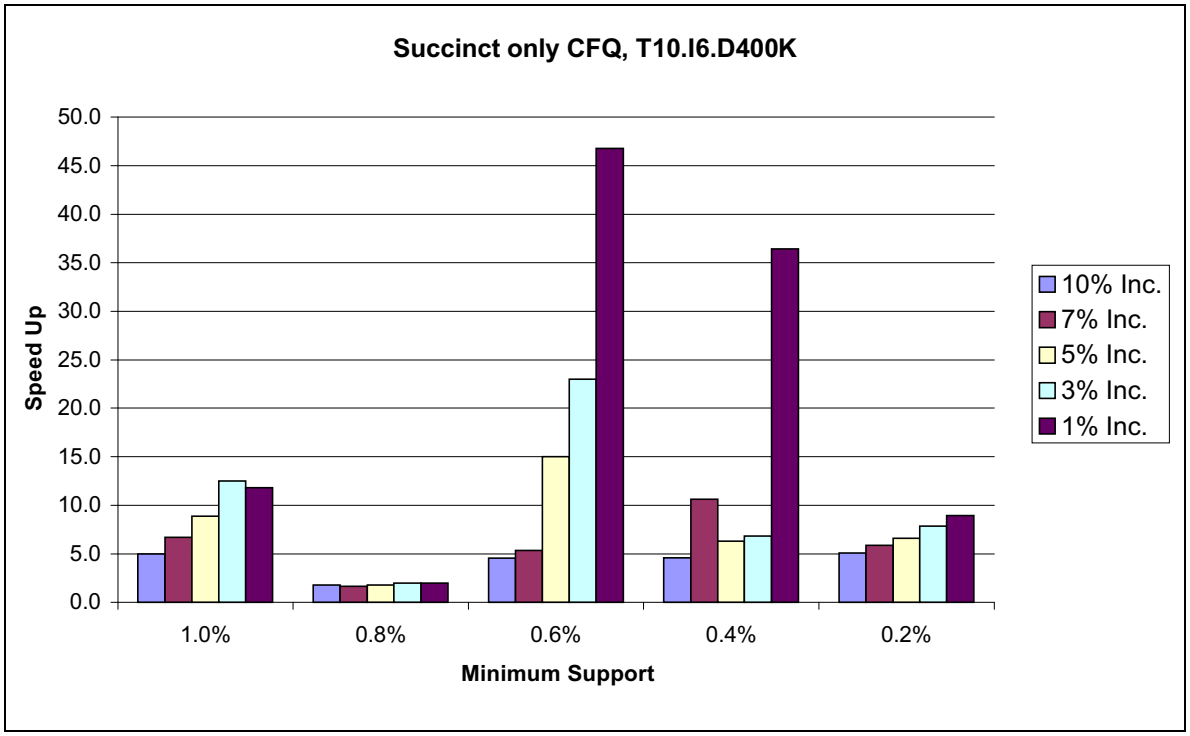

Figure 5.1: Speed up for succinct only and non-succinct non-anti-monotone constraints

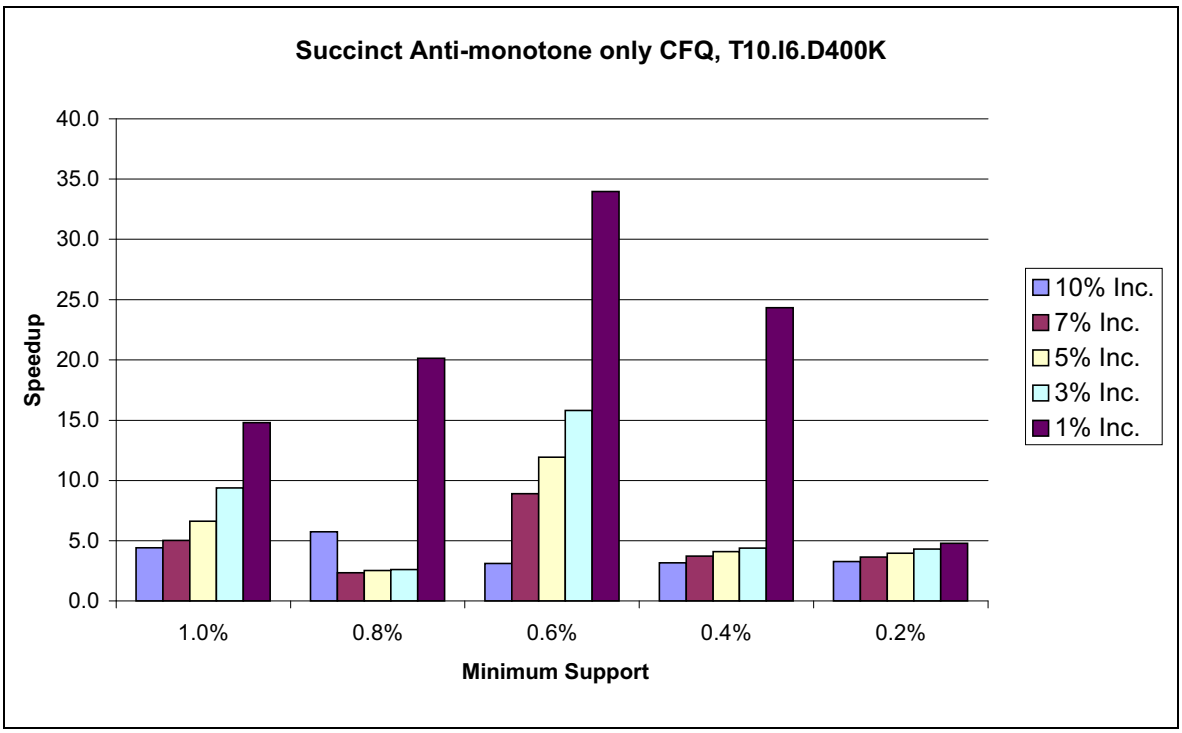

Figure 5.2: Speedup for succinct anti-monotone constraint

Several observations apply generally to all the results of this set of experiments. These observations are:

a- The speedup exhibits a global maximum around the support value of 0.6%. This is similar to the results reported in [44] for the performance gain of the *BORDERS* algorithms over to the *Apriori* algorithm. A justification for this, similar to that in [44], is applicable here, when the support threshold increases, the number of frequent itemsets decrease together with its size. This reduces the time taken by the non-incremental algorithm *CAP*. When decreasing the support threshold the time taken by *CAP* increases with a faster rate than *ICAP* for which the execution time increases with a much slower rate. Hence the speedup increases until we reach the maximum value. Beyond this, the number of frequent itemsets increases rapidly which increases the burden of *ICAP* trying to maintain them and the probability of the border expansion increases. *CAP*, however, sustains its rate of increase in time. This lowers the speedup again. The exact value of *minsup* for which this maximum occurs is for sure dependent on the characteristics of the database. This means we should expect a trend similar to the one in the figures but shifted either to the left or to the right on the x-axis according to the data characteristics.

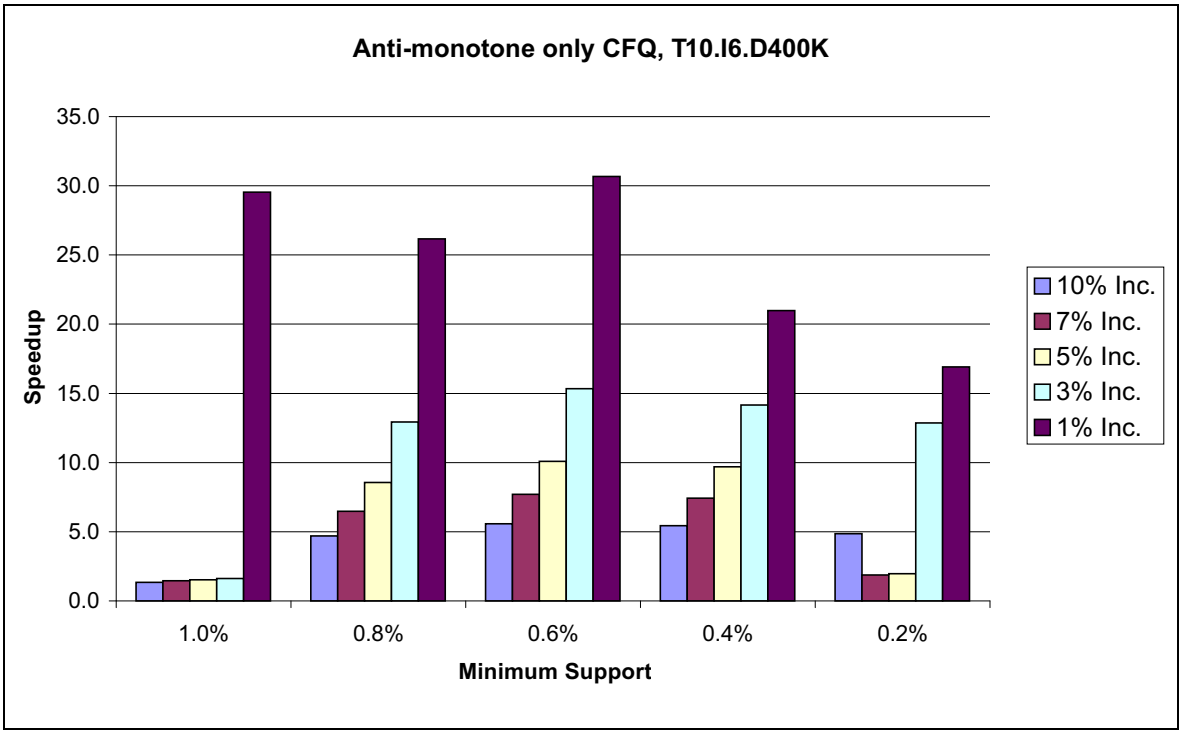

Figure 5.3: Speedup for anti-monotone constraint

b- The second observation, which is very much expected, is that, generally, the speedup increases with the decrease in the increment size to reach its maximum for 1% increment size. This is easily justified by noticing that the main problem of *ICAP* is to discover the set of constrained frequent itemsets in the increment. Decreasing the increment size would relief it of some of its burdens (keeping in mind that *CAP* takes the same time to produce the set of constrained frequent itemsets for the whole database).

c- Several glitches appear in the figures that seem to disobey the general trend discussed above (e.g. the case of 0.8% *minsup* in Figure 5.1). The reason for these abnormalities can be explained knowing that the border expanded in all these cases. Such abnormalities are due to the local characteristics of the transactions in the increment. This means that if the increment database was taken from another part of the whole database, the border might not expand. To eliminate such glitches, it is essential to run the same experiment several times for the same increment sizes and support values but this time changing the place from which the increment database is taken and then taken the mean speedup for all the cases. This would smooth the curves and give a much better trend. It is worthy to mention, however, that the speedup is always over 1 (its minimum value is 1.3 in the case of 10% increment size and 1% minimum support threshold in Figure 5.3) indicating a performance gain in all cases, even those cases when performance drops due to border expansion. It can be seen also that, assuming the border expands for all the cases, the performance gain rises with the decrease in minimum support. This is illustrated in Figure 5.1 for the case of 0.8% and 0.2% minimum support. In both these cases, the border expanded for all increment sizes, however, speedup reached up to 5-9 in the latter case (0.2% *minsup*) although it was confined around 2 for the former case (0.8% *minsup*).

d- To eliminate the effect of the local characteristics of the data in the increment database, the experiment for the hybrid constraint was repeated for 10% and 5% increment sizes. This time, the increment database was taken as a contiguous block from 10 different places not just from the beginning block. The speedup is tested for each case and then its average is taken. This has the effect of *smoothing* the glitches. The results are shown in Figure 5.5. From the figure, it can be noticed that speedup still obeys the same general trend.

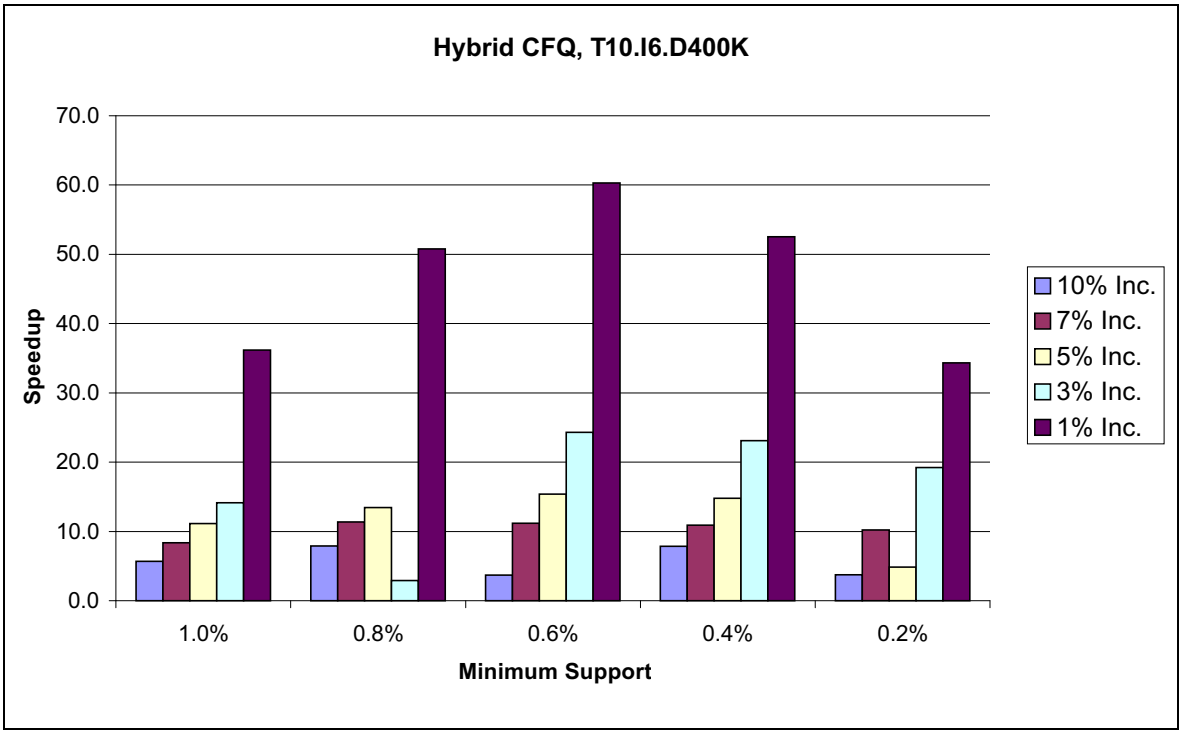

Figure 5.4: Speedup for a hybrid constraint

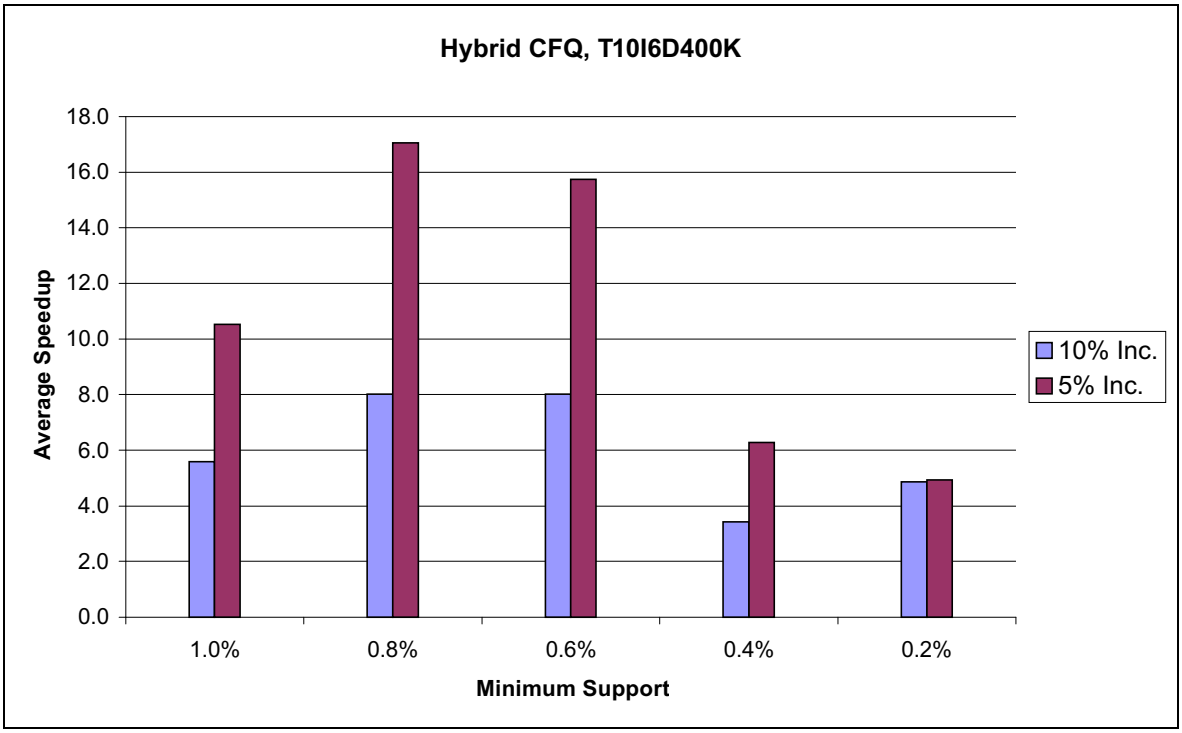

Figure 5.5: Average Speedup for the hybrid constraint

Some particular observations can be deduced for every case:

a- *For the hybrid constraint*, we can comment that the performance gain is more drastic than in the other cases (Figure 5.4). This is attributed to two things, first, combining both the succinct and succinct anti-monotone constraints increased the selective power of the hybrid constraint and therefore lowered the number of frequent itemsets satisfying it. This increases the performance of *ICAP* since the number of frequent patterns it has to generate in the increment database is lowered and the probability of border expansion is also lowered. The second reason is that the succinct constraint component included in the constraint is more selective than that of the succinct only constraint of Figure 5.1 for example. This is a good sign for *ICAP* since it indicates better performance gains when constraints tend to be more constraining. This will normally be the case in a practical environment.

b- *For the anti-monotone constraint and the succinct-anti-monotone constraint*, it should be noted that in the later case, *CAP* degenerates to *Apriori* but on a smaller or a restricted subset of the items. Similarly, for the former case, the only difference between *CAP* and *Apriori* is that *CAP* prunes some itemsets (those not satisfying the anti-monotone constraint) early before counting them. It should be expected then to have similar results of trends of speedup between *CAP* and *ICAP* as those between *BORDERS* and *Apriori* reported in [44].

c- *For the non-succinct non-anti-monotone constraint*, it should be noted that the constraint used will induce the weaker constraint $min(S.Price) \leq 15$ which has exactly the same constraining capability as the succinct only constraint of Figure 5.1 since both constraints have the same selectivity. The extra pass required to filter out the extra itemsets not satisfying the original constraint accounted for a negligible amount of time with respect to the other tasks. This made the results of Figure 5.1 apply also to this case.

5.3. Measuring scalability with database size

The second experiment intended to test how *ICAP* scales-up when changing the database size. The database size was changed in 100,000 transactions increments starting from 100,000 transactions until 400,000 transactions. The experiment was conducted for the 10% increment database size. The reason for this choice is to try to be harsh on the incremental

algorithm, by testing it in its worst cases. The results of the experiment can be seen in Figure 5.6. It was expected that the performance gain should increase drastically with the increase in database size since this increases more the I/O cost of the non-incremental algorithm. The results of the experiment, as seen from the figure, was not quite what was expected. In spite of the increase in the performance gain with the database size, the recorded increase was not that significant. The reason for this is that the size of the database file is small (18 MB in the case of 400,000 transactions) compared to the size of the main memory. This enables the operating system to cache the whole database file into main memory after the first pass. The increase in the database size in this case did not impose a drastic difference on performance. An interesting question arises here which is, *if the whole database can be cached in memory, from where does the incremental algorithm obtain its edge?* The answer is that even if the database is fully accommodated in main memory, the non-incremental algorithm wastes too much time sequentially passing on it. This gives the incremental algorithm its performance gain. More drastic increase in performance should be noted then if the database size exceeded the size of the main memory.

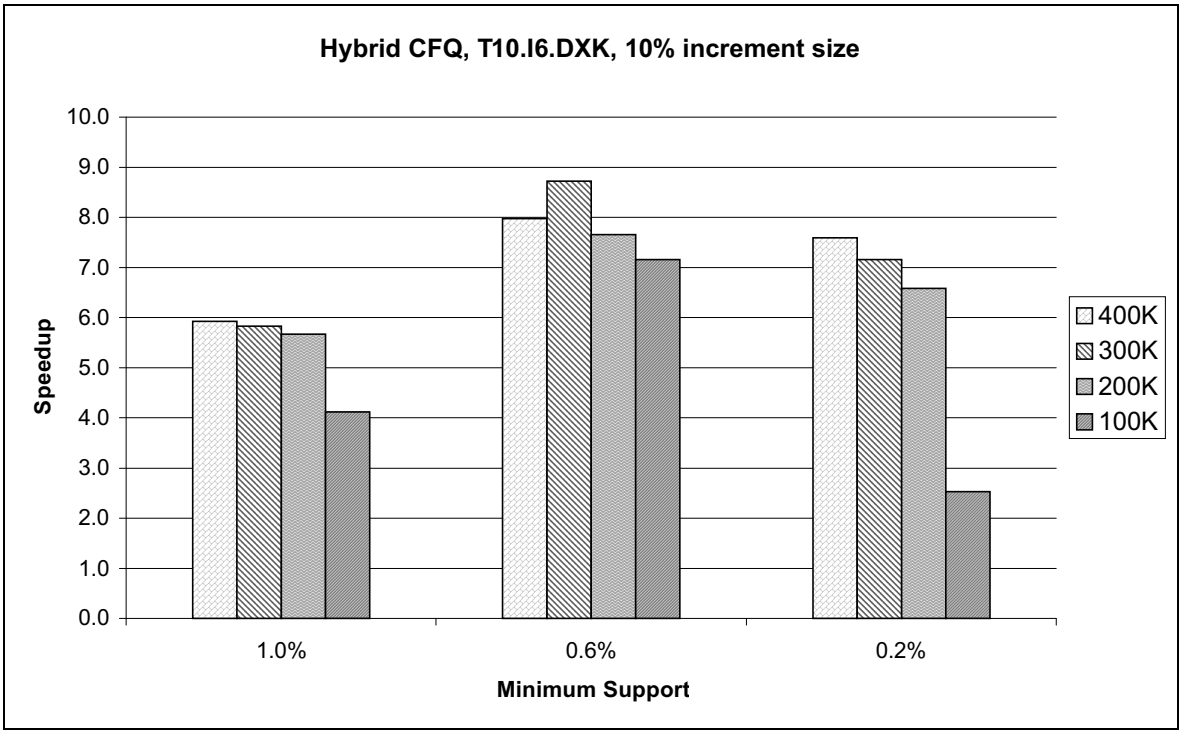

Figure 5.6: Measuring performance for different database sizes

Another observation on Figure 5.6 is that the performance gain increases (though slowly) with decreasing support threshold. This is expected since the non-incremental algorithm passes more on the database in case more itemsets are frequent.

5.4. Measuring sensitivity to item selectivity

In case of succinct constraints, and from Lemma 5, the constrained negative border requires the generation and counting of several itemsets that are known beforehand to be not frequent. From section 4.2, to generate the minimum sized itemsets satisfying he succinct constraint, it is required to generate the cross product of the mandatory sets in the *MGF* of the constraint. The number of itemsets in the border can be very large, especially if the cardinality of the sets involved in the cross product is high. This places a burden on *ICAP* that is not existing for *CAP* and raises a natural question, *does the performance gain disappear for higher number of border items?*

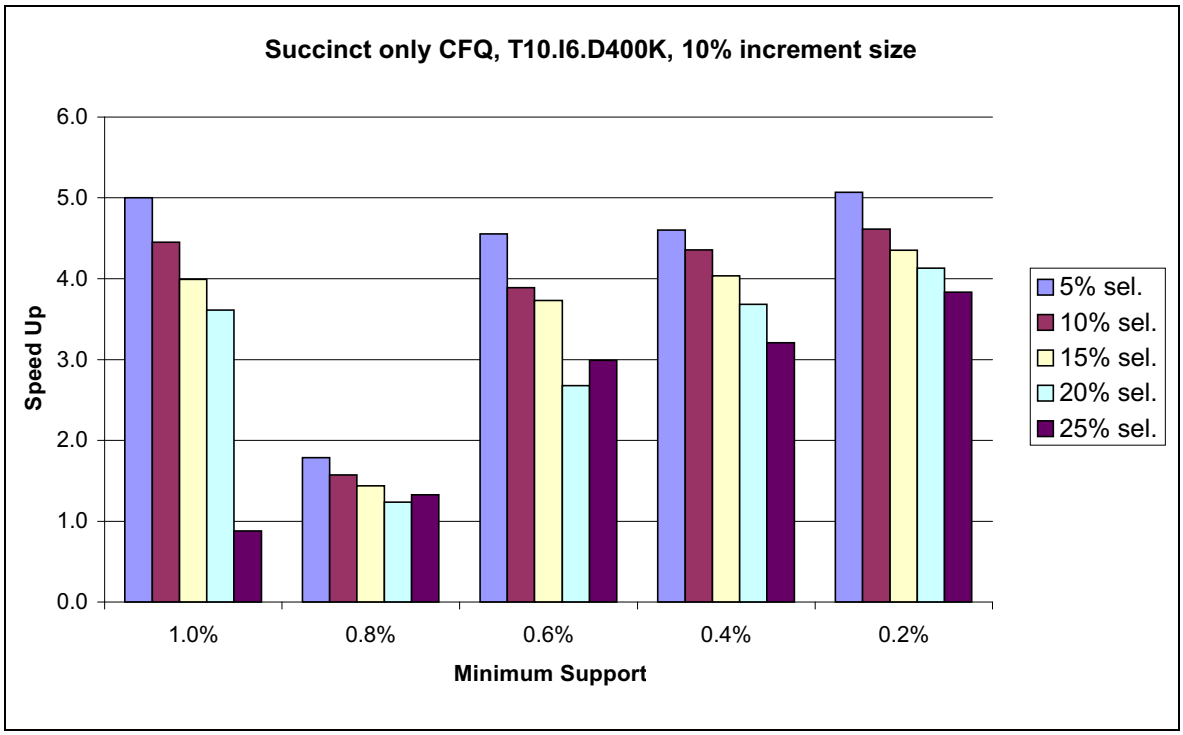

Figure 5.7: Measuring performance with item selectivity

The following experiment was designed to answer this question. The algorithms are compared for the same succinct constraint of Figure 5.1 and for 10% increment size (again to be harsh on *ICAP*) and the different support settings. The item selectivity was increased from

5% to 25% (i.e. the number of soda items in this case was changed from 5% of the whole set of items to 25%). This increased the number of border itemsets.

The results of the experiment can be seen in Figure 5.7. The interesting discovery is that *ICAP* still has its performance edge. This is due to the fact that, as reported in [35], the performance of *CAP* itself also drops with item selectivity because it has to count more itemsets that satisfy the constraint, and more importantly the probability that the length of the frequent itemsets increases is highly costing it extra precious passes over the database. This balances the deterioration in time in *ICAP* needed to maintain the large number of border itemsets.

Chapter 6

CONCLUSION AND SUGGESTIONS FOR FUTURE WORK

6.1. Conclusion

In this thesis, a new algorithm (Incremental Constrained APriori *ICAP*) for incremental mining of constrained association rules is proposed. The algorithm applies the techniques of incremental mining on the constrained mining algorithm *CAP* (Constrained APriori) to produce the new set of updated constrained frequent-sets after the update of the original transactional database by adding new transactions. The proposed algorithm fills the existing gap between the two separate paradigms of incremental and constrained mining of association rules.

In the course of developing the proposed algorithm, the concept of the *constrained negative border* is introduced as the counterpart of the classical concept of the *negative border*. The *constrained border* is proved to have the same property as the normal border, namely being an indicator for the necessity of checking the original database, which is the most costly operation in incremental mining.

The proposed algorithm *ICAP* utilizes the *constrained border* efficiently to maintain the set of constrained frequent-sets. As a direct consequence of this usage, *ICAP* performs at most *one* pass on the original database and only when it is *absolutely* necessary.

Several experiments are conducted to measure the relative performance of the new algorithm compared to rerunning the *CAP* algorithm from scratch on the whole updated database. The test results show that *ICAP* exhibits a speedup gain in virtually all situations. The sensitivity of the algorithm is tested for several increment sizes and support thresholds.

6.2. Suggestions for future work

Several questions remain unanswered and are appropriate directions for future research work:

- In terms of "market basket analysis" when transactional databases represent retail data, it is convenient to expect the data to be of the *add only* type. This justifies that this work deals only with updates in the form of database insertions. However, the general problem of updating the transaction database includes also the deletion of transactions. Thus, an

immediate extension is to study the general problem. Handling deletion of transactions should present no problem using the same techniques in this work. However, several parameters should be studied, the most important of which is when it becomes costly to use the incremental technique. This is because deleting transaction *reduces* the size of the updated database, which might introduce situations in which using an incremental technique is not worthwhile.

- As mentioned in section 4.4, several alternatives exist to measure the appropriate time to run the incremental algorithm. This thesis does not go beyond suggesting such methods. Every alternative should be more investigated to discover the circumstances in which the use of one alternative will be more appropriate than the other. It might be the case that a combination of such alternatives is more appropriate.
- There is an expected surge in the research efforts trying to adapt the new techniques for mining frequent patterns in general to the classical problems of mining association rules. Also a very recent publication formally restated the problem of mining constrained frequent-sets [38] and introduced other properties of constraints (*monotone* and *convertible* constraints). This publication is a starting step in the efforts trying to use the new very efficient techniques to mine constrained frequent sets. Similar efforts should be done to try to combine the benefits of these techniques with that of incremental mining, and needless to say, with incremental mining of constrained rules [27].
- Another question is still open. What will be the case if the database update modified not only the transactional part but also the characteristic data of the items (e.g. changing the price of some items or changing the item type)? It is to be noted that this thesis assumed only modifications in the transactional part. This assumption eases the situation since it guarantees that satisfaction of constraints other than the support constraint will not be affected. However, a modification in the characteristic part of the database would invalidate this fact. Actually, the above question is a tough question to answer. The reason for this relies in that the concept of *winners* and *losers* will not be as simple as it was. The difficulty of this problem is on par with another interesting question in the classical problem of incremental mining of associations, which is *what if we need to discover the new set of frequent itemsets after changing the support threshold?* The literature has no answer to this question, lest the work in [34] which tried to solve an interestingly similar problem.

REFERENCES

[1] Agrawal, C. C.; and Yu, P. H. A New Framework For Itemset Generation. *In Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, June 1-3, 1998, Seattle, Washington. ACM Press 1998.

[2] Agarwal, R.; Aggarwal, C.; and Prasad, V. V. V. A Tree Projection Algorithm for Generation of Frequent Itemsets. In *Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining)*, (to appear), 2000.

[3] Agrawal, R.; Arning, A.; Bollinger, T.; Mehta, M.; Shafer, J. and Srikant, R. The Quest Data Mining System. *Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, August 1996.

[4] Agrawal, R.; Imielinski, T.; and Swami, A. Database Mining: A Performance Perspective. *IEEE Transactions on Knowledge and Data Engineering, Special issue on Learning and Discovery in Knowledge-Based Databases*, 5(6): 914-925, December 1993.

[5] Agrawal, R.; Imielinski, T.; and Swami, A. Mining Associations between Sets of Items in Massive Databases. *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, pp. 207-216, Washington D.C., May 1993.

[6] Agrawal, R. and Psaila, G. Active Data Mining. *Proc. of the 1st Int'l Conference on Knowledge Discovery and Data Mining*, Montreal, August 1995.

[7] Agrawal, R.; Shafer, J.C. Parallel Mining of Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No. 6, December 1996.

[8] Agrawal, R.; and Srikant, R. Fast Algorithms for Mining Association Rules. *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, Sept. 1994.

[9] Agrawal, R.; and Srikant, R. Mining sequential patterns. *In Proc. 1995 Int. Conf. Data Engineering*, Taipei, Taiwan, March 1995.

[10] Ahmed, Khalil M.; El-Makki, Nagwa M.; and Taha, Yousry. A note on "Beyond Market Baskets, Generalizing association rules to correlations". In *SIGKDD explorations: Newsletter* of *The Special Interest Group (SIG) on Knowledge Discovery & Data Mining,* Vol. 1 Issue 2, January 2000.

[11] Ali, K.; Manganaris, S.; and Srikant, R. Partial Classification using Association Rules. In Proc. of the 3rd Int'l Conference on Knowledge Discovery in Databases and Data Mining, August 1997.

[12] Ayan, N. F.; Tansel, A. U.; and Arkun, E. An efficient algorithm to update large itemsets with early pruning. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'99)* August 1999, San Diego, CA USA.

[13] Bayardo, R. J. Efficiently Mining Long Patterns from Databases. *In Proc. of the 1998 ACM-SIGMOD Int'l Conf. on Management of Data*, pp. 85-93.

[14] Bayardo, R. J. and Agrawal, R. Mining the Most Interesting Rules. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'99)* August 1999, San Diego, CA USA.

[15] Bayardo, R. J.; Agrawal, R.; and Gunopulos, D. Constraint-Based Rule Mining in Large, Dense Databases. In *Proc. of the 15th Int'l Conf. on Data Engineering* , pp. 188-197, 1999.

[16] Bradley, P.; Fayyad, U.; and Mangasarian, O. Data Mining: Overview and Optimization Opportunities. *Microsoft Research Report MSR-TR-98-04*, January 1998

[17] Brin, S.; Motwani, R.; and Silverstein, C. Beyond Market Baskets: Generalizing Association Rules to Correlations. In *Proc. of the 1997 SIGMOD Conf. on the Management of Data*, pp. 265-276.

[18] Brin, S.; Motwani, R.; Ullman, J.; and Tsur, S. Dynamic Itemset Counting and Implication Rules for Market Basket Data. In *Proc. of the 1997 SIGMOD Conf. on the Management of Data*, pp. 255-264.

[19] Chen, M. S.; Han, J.; and Yu, P.S. Data Mining: An Overview from a Database Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6): 866-883, 1996.

[20] Cheung, D.; Han, J.; Ng, V.; and Wong, C.Y. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. *Proc. of 1996 Int'l Conf. on Data Engineering (ICDE'96)*, New Orleans, Louisiana, USA, Feb. 1996.

[21] Cheung, D. W. L.; Lee, S.D.; and Kao, B. A general incremental technique for maintaining discovered association rules. *In Proceedings of the Fifth International Conference On Database Systems For Advanced Applications*, pp. 185-194, Melbourne, Australia, March 1997.

[22] Christian, H. Online Association Rule Mining. *Proc. 1999 ACM-SIGMOD Conf. on Management of Data (SIGMOD'99)*, Philadelphia, PA, June 1999.

[23] Feldman, R.; Aumann, Y.; Amir, A.; and Mannila, H. Efficient Algorithms for Discovering Frequent Sets in Incremental Databases. In *Proceedings of the 1997 SIGMOD Workshop on DMKD, Tucson, Arizona*, May 1997.

[24] Ganti, V.; Gehrke, J. E.; and Rmakrishanan, R. DEMON: Mining and monitoring evolving data. In *Proceedings of the 16th International Conference on Data Engineering*, San Diego, 2000.

[25] Han, J.; and Fu, Y. Discovery of Multiple-Level Association Rules from Large Databases. *Proc. of 1995 Int'l Conf. on Very Large Data Bases (VLDB'95)*, Zürich, Switzerland, September 1995.

[26] Han, J.; Lakshmanan, L. V. S.; and Ng, R. Constraint-based, Multidimensional data mining. *IEEE Computer, Special issue on data mining, August 1999.*

[27] Han, J.; Pei, J.; and Yin, Y. Mining Frequent Patterns without Candidate Generation. *Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00)*, Dallas, TX, May 2000.

[28] Klemettinen, M.; Mannila, P.; Ronkainen, P.; and Verkamo, A. I. Finding Interesting Rules from Large Sets of Discovered Association Rules. In *Proc. of the Third Int'l Conf. on Information and Knowledge Management*, pp. 401-407, 1994.

[29] Lakshmanan, L. V. S.; Ng, R.; Han, J.; and Pang, A. Optimization of Constrained Frequent Set Queries with 2-Variable Constraints. *Proc. 1999 ACM-SIGMOD Conf. on Management of Data (SIGMOD'99)*, Philadelphia, PA, June 1999, pp. 157-168.

[30] Lee, S.D.; and Cheung, D. W. L. Maintenance of Discovered Association Rules: When to update?. In *Proceedings of the 1997 ACM-SIGMOD Workshop on Data Mining and Knowledge Discovery (DMKD-97)*, Tucson, Arizona, May 1997.

[31] Lin, D.; and Kedem, Z. M. Pincer-Search: A New Algorithm for Discovering the Maximum Frequent Set. In *Proc. of the Sixth European Conf. on Extending Database Technology*.

[32] Liu, B.; Hsu, W.; and Ma, Y. Mining Association Rules with Multiple Minimum Supports. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'99)*, San Diego, CA USA, August 1999.

[33] Mueller, A. Fast sequential and parallel algorithms for association rule mining: A comparison. Technical Report CS-TR-3515, University of Maryland, College Park, August 1995.

[34] Nag, B.; Deshpande, P. M.; and DeWitt, D. J. Using a Knowledge Cache for Interactive Discovery of Association Rules. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'99)*, San Diego, CA USA, August 1999.

[35] Ng, R. T.; Lakshmanan, V. S.; Han, J.; and Pang, A. Exploratory Mining and Pruning Optimizations of Constrained Association Rules. In *Proc of the 1998 ACM-SIG-MOD Int'l Conf. on the Management of Data*, pp. 13-24, 1998.

[36] Omiecinski, E.; and Savasere, A. Efficient mining of association rules in large dynamic databases. *In Proc. BNCOD'98*, pages 49-63, 1998.

[37] Park, J-S.; Chen, M-S.; and Yu, P. S. An Effective Hash Based Algorithm for Mining Association Rules. *Proceedings of ACM SIGMOD*, pp. 175-186, May 1995.

[38] Pei, J. and Han, J. *Can We Push More Constraints into Frequent Pattern Mining?* To appear in *Proc. 2000 Int. Conf. on Knowledge Discovery and Data Mining (KDD'00), Boston, MA, August 2000.*

[39] Sarda, N. L.; and Srinivas, N. V. An adaptive algorithm for incremental mining of association rules. *In Proceedings of DEXA Workshop'98*, pp. 240-245, 1998.

[40] Savasere, A.; Omiecinski, E.; and Navathe, S. An Efficient Algorithm for Mining Association Rules in Large Data-bases. In *Proc. of the 21st Conf. on Very Large Databases*, Zurich, Switzerland, September 1995.

[41] Savasere, A.; Omiecinski, E.; and Navathe, S. B. Mining for Strong Negative Associations in a Large Database of Customer Transactions. *Proceedings of the International Conference on Data Engineering*, Orlando, Florida, February 1998.

[42] Srikant, R.; and Agrawal, R. Mining Generalized Association Rules. *Proc. of the 21st Int'l Conference on Very Large Databases*, Zurich, Switzerland, September 1995.

[43] Srikant, R.; Vu, Q.; and Agrawal, R. Mining Association Rules with Item Constraints. In *Proc. of the Third Int'l Conf. on Knowledge Discovery in Databases and Data Mining*, August 1997 pp. 67-73.

[44] Thomas, S.; Bodagala, S.; Alsabti, K.; and Ranka, S. An efficient algorithm for the incremental updation of association rules in large databases. *In Proceedings of the 3rd International conference on Knowledge Discovery and Data Mining (KDD 97)*, New Port Beach, California, August 1997.

[45] Toivonen, H. Sampling large databases for association rules. *In 22nd International Conference on Very Large Databases (VLDB'96)*,pp. 134-145, Mumbay, India, September 1996.

[46] Toivonen, H.; Klemettinen, M.; Ronkainen, P.; Hätönen, K.; and Mannila H. Pruning and grouping discovered association rules. In *MLnet Workshop on Statistics, Machine Learning, and Discovery in Databases*, pp. 47-52, Heraklion, Crete, Greece, April 1995.

[47] Tung, A. K. H.; Lu, H.; Han, J.; and Feng, L. Breaking the Barrier of Transactions: Mining Inter-Transaction Association Rules. *Proc. 1999 Int. Conf. on Knowledge Discovery and Data Mining (KDD'99)* , San Diego, CA, Aug. 1999.

[48] Zaki M. J. Parallel and Distributed Association Mining: A Survey**,** in *IEEE Concurrency*, special issue on Parallel Mechanisms for Data Mining, Vol. 7, No. 4, pp14-25, December, 1999

[49] Zaki, M. J. and Ogihara, M. Theoretical Foundations of Association Rules. *3rd SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, pp 7:1-7:8, Seattle, WA, June 1998.

[50] Zaki, M. J.; Parthasarathy, S.; Ogihara, M.; and Li, W. New Algorithms for Fast Discovery of Association Rules. In *Proc. of the Third Int'l Conf. on Knowledge Discovery in Databases and Data Mining*, pp. 283-286, 1997.

ملخص الرسالة

إن مشكلة التنقيب عن قواعد الارتباط قد جذبت كثيراً انتباه المجتمع البحثى فى الآونة الأخيرة. و قد ظهرت عدة أساليب لاكتشاف قواعد الارتباط. على الرغم من ذلك، فإنه ليس من السهل القيام بالتنقيب المتزايد أو التنقيب المقيد عن قواعد الارتباط. لذا، فإن المجتمع البحثى قد ركز على تقديم حلول منفصلة لهاتين المشكلتين.

بما أن الاتجاه السائد هو اعتبار التنقيب المقيد سيصبح هو الأساس، لذا يلزم إيجاد طرق جديدة للتنقيب المتزايد عن قواعد الارتباط المقيدة. هذه الرسالة تتقدم بخوارزم جديد للتنقيب المتزايد عن تلك القواعد.

على طريق تطوير الخوارزم الجديد، تقدم الرسالة مبدأ "الحد السالب المقيد" و تقترح استخدامه لصيانة قواعد الارتباط المقيدة عندما يتم إضافة بيانات لعمليات جديدة إلى قاعدة بيانات العمليات الأصلية.

أحد الخواص الأساسية للخوارزم المقترح هو عدم احتياجه للمرور على قاعدة البيانات الأصلية إلا مرة واحدة على الأكثر عندما يؤدى إضافة العمليات الجديدة إلى تمدد الحد السالب. من هنا، فإن سرعة التنقيب المتزايد يتم دمجها مع المرونة التى يوفرها التنقيب المقيد.

فى هذه الرسالة، يتم دراسة الخوارزم الجديد و إثبات صحته، كما تقدم الرسالة مجموعة من التجارب لدراسة أدائه فى مقابل استخدام الخوارزم الأصلى للتنقيب المقيد الغير متزايد على قاعدة البيانات النهائية بعد إضافة العمليات الجديدة. تشير النتائج إلى حدوث تحسن كبير فى زمن التشغيل عند استخدام الخوارزم الجديد.

تحتوى الرسالة على ستة فصول و يتضمن الجزء الخاص بالمراجع 50 مرجعاً.

الفصل الأول: عبارة عن مقدمة للرسالة تتضمن الدافع إليها و أهم أهدافها.

الفصل الثاني: يحتوى على الخلفية اللازمة للبحث فهو يعرض لأهم الأبحاث المتعلقة بـــالتنقيب عـــن قواعد الارتباط بصفة عامة بالإضافة إلى التنويعات المختلفة للمشكلة. كما يقوم بعرض أهم الأبحـــاث المقدمة فى مجالى التنقيب المقيد و المتزايد.

الفصل الثالث: يعرض الأساس النظرى الذى بنى عليه الخوارزم المقترح. فى هذا الفصل يتم تقديـــم و شرح مبدأ الحد السالب المقيد و إثبات إمكانية استخدامه كعلامة على ضرورة المرور على قـــاعدة البيانات الأصلية.

الفصل الرابع: يقوم بتقديم و شرح الخوارزم الجديد. فى هذا الفصل أيضاً يتم العرض لطرق حســـاب الحد السالب المقيد ثم يعطى الفصل مثالاً عملياً على تحسين الأداء.

الفصل الخامس: يقدم نتائج التجارب التى تمت لمقارنة أداء الخوارزم المقترح بنظيره الغير مـــتزايد. فى هذا الفصل يتم التعليق على هذه التجارب و شرح النتائج.

الفصل السادس: هو خاتمة الرسالة، و فيه أهم الاستنتاجات و بعض الاتجاهات البحثية المستقبلية فـــى هذا المجال.

جامعة الإسكندرية

كلية الهندسة

خوارزم جديد للتنقيب المتزايد عن قواعد الارتباط المقيدة

رسالة مقدمة لقسم الآلات الحاسبة و التحكم الآلى

كمتطلب جزئى للحصول على درجة الماجستير فى الآلات الحاسبة

مقدمة من:

م/ أحمد مدحت عياد

جامعة الإسكندرية 2000