A CONSTRAINT-BASED INCREMENTAL APPROACH FOR UPDATE OF LARGE ITEMSETS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Engin Demir

August, 2001

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

Prof. Dr. Erol Arkun (Advisor)

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

Prof. Dr. H. Altay Güvenir

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

Assoc. Prof. Dr. Özgür Ulusoy

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

A CONSTRAINT-BASED INCREMENTAL APPROACH FOR UPDATE OF LARGE ITEMSETS

Engin Demir
M.S. in Computer Engineering
Supervisor: Prof. Dr. Erol Arkun
August, 2001

The main focus of data mining, subject of numerous studies in recent years, is to extract and analyze hidden knowledge from the collected and stored massive amounts of data. In order to manage this process, many automated knowledge discovery techniques have been developed. Discovery of association rules, which is the process of mining interesting and frequent patterns from data, is an important class of data mining. Most of the researchers proposed techniques for static databases but in real applications dynamic approaches are necessary to maintain the frequent patterns in terms of association rule mining. Additionally, from the standpoint of users, previously developed systems have lack of user exploration and control, lack of focus, and rigid notion of relationships. In this thesis, we propose a solution to both of the shortcomings of the previous systems. We integrate the anti-monotonicity and succinctness constraints to the previously developed update with early pruning (UWEP) algorithm.

Keywords: Data mining, constraint based, association rules, update of large itemsets, pruning.

ÖZET

YOĞUN NESNE KÜMELERİNİN GÜNCELLENMESİ İÇİN SINIRLAMAYA DAYALI ARTIMLI BİR YAKLAŞIM

Engin Demir
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Erol Arkun
Ağustos, 2001

Son yıllarda birçok çalışmaya konu olan veri araştırmasının temel amacı, biriktirilen ve depolanan yüksek hacimli verilerin içinde saklı olan bilgileri çıkartmak ve çözümlemektir. Bu işlemi yürütmek için çok sayıda bilgi bulma teknikleri geliştirilmiştir. Verilerdeki ilginç ve yoğun oluşumları bulmaya yarayan ilişki kurallarının belirlenmesi veri araştırmasında önemli bir yaklaşımdır. Durağan veritabanları için kimi teknikler geliştirilmiş olmasına karşın; gerçek uygulamalarda ilişki kurallarının araştırılmasında yoğun oluşumları güncellemek için dinamik yaklaşımlara da gereksinim vardır. Ayrıca, kullanıcılar açısından bakıldığında, mevcut sistemlerde, arama ve denetim eksiklikleri, odaklanma eksikliği, ve ilişkilendirmelerde katı bir yalınlık vardır. Bu tezde, mevcut sistemlerdeki iki eksikliğe de çözüm önerilmektedir. Erken eliminasyon yöntemi ile yoğun nesne kümelerinin güncellenmesini gerçekleştiren UWEP algoritmasına, monoton olmayan ve özlü sınırlamalara uyum yetenekleri de uyarlanmıştır.

Anahtar sözcükler: Veri araştırması, sınırlamaya dayalı, ilişki kuralları, yoğun nesne kümelerinin güncellenmesi.

Acknowledgement

I would like to express my special thanks and gratitude to Prof. Erol Arkun, from whom I have learned a lot, due to his supervision, suggestions, and support during this research.

I am also indebted to Prof. Dr. H. Altay Güvenir and Assoc. Prof. Dr. Özgür Ulusoy for showing keen interest to the subject matter and accepting to read and review this thesis.

I would like to thank to my parents and my beloved, NAZLI for their morale support and for many things.

Acknowledging all the honorable faculty members of the department and my dear friends and colleagues for their help, my special thanks go to Necip Fazıl Ayan for the supply of his UWEP implementation.

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

X	: a set of itemsets
DB	: original database
db	: set of inserted transaction to the database
db^-	: set of deleted transactions from the database
DB'	: updated database $(DB + db - db^-)$
DB^-	: set of unchanged transactions $(DB - db^-)$
$\|D\|$	: the number of transactions in database D
$support_D(X)$	: support of itemset X in database D
$tidlist_D(X)$	: transaction list of X in database D
$minsup$	: minimum support threshold
$minconf$	: minimum confidence threshold
C_D^k	: the set of candidate k-itemsets in database D
C_D	: the set of all candidate itemsets in database D
L_D^k	: the set of large k-itemsets in database D
L_D	: the set of all large itemsets in database D
P^k	: set of large k-itemsets in C^k
Q^k	: set of candidate k-itemsets that are not large in C^k
$NBd(S)$	: negative border of S
$CNBd(S)$	: constrained negative border of S
C_m	: constraints that are monotone
C_{sam}	: constraints that are both anti-monotone and succinct
C_{am}	: constraints that are anti-monotone but not succinct
C_{suc}	: constraints that are succinct but not anti-monotone
C_{none}	: constraints that are neither anti-monotone nor succinct

Chapter 1

Introduction

Knowledge management is the most time consuming and expensive part of our daily lives. While the storage of data is getting even bigger, the strategies for processing all these data become much more complicated. The automatic knowledge discovery tools have emerged in order to overcome this difficulty, and have taken the great attention of researchers in the database literature.

Results of these automated tools can be too complicated that in some cases we need another automated tool to simplify the representation of these results. Currently there are a few methodologies to identify the knowledge within huge amounts of data such as association, classification, clustering, and sequence mining. All these methods attack the same problem in different representation schemas. Association has the goal to discover the re-occurrence of records in a dataset to identify some types of implications between these records. Classification targets finding out the characteristics of the underlying data, and this is used in the estimation of a specific attribute in the domain. Clustering divides the dataset into disjoint groups with a similarity measure defined by the user. Sequence mining uses the order of the records in the dataset and determines the characteristics of the dataset according to the sequence of any subset of data.

An association rule, $X \Rightarrow Y$, is a statement of the form "for a specified fraction of the total transactions, a particular value of the attribute set X determines the

value of an attributes set Y with a certain confidence". The problem of association rule mining is divided into two consequent sub-problems [3]:

1. All large itemsets (frequent patterns) are discovered in the transaction database, where large itemsets represent the itemsets which have a number of occurrence greater than or equal to a user specified minimum threshold in a database.

2. Association rules are generated using the large itemsets found in the previous step.

Frequent pattern analysis and the simplicity in the representation of associations made association rule mining popular in the research community, and an extensive amount of research has been done on this area. Customer transactions, web logs, event logs, biological information databases are just a few types of databases where association rule mining is applicable.

In this thesis, we attack the problem of maintenance of large itemsets in dynamic databases where the user can control the process by determining his focus in terms of constraints. The previously proposed incremental algorithms can be considered as a black-box where the user has no control over the process until the end of the algorithm. In our approach, we used anti-monotonicity and succinctness constraints, which will be defined in the next chapter, to set the focus of the user in the mining step. As an incremental approach we carried out the methodology presented in UWEP [10], and proposed an efficient algorithm called **C**onstraint-**B**ased **U**pdate of **L**arge **I**temsets (CBULI). Our algorithm fills the gap between constraint based mining and association rule mining in an incremental environment. CBULI scans the incremental database, *db* once and initial database, DB at most once with the advantage of generating and counting the minimal number of candidates in the large itemset generation framework by using user specified constraints.

1.1 Outline of the Thesis

This thesis is organized as follows. We introduce association rule mining with an analysis of the previous algorithms in Chapter 2. Especially a broad survey on constraint based mining and incremental update techniques is included to give a detailed understanding of the concepts underlying the approach we present in the following chapter. In Chapter 3, we present an incremental mining algorithm called $CBULI$, which achieves a maximized degree of pruning for all categories of constraints, and experimental results are reported in Chapter 4. We conclude our thesis and give suggestions for future work in Chapter 5.

Chapter 2

A Survey in Association Rule Mining

2.1 Background

The research area of Knowledge Discovery in Databases and Data Mining is a composition of several disciplines from statistics, databases, pattern recognition, artificial intelligence, optimization, visualization, and parallel computing. Data Mining is defined as a sub-process in Knowledge Discovery in Databases in database terminology. We repeat the definitions of these terms as given in [13] to make clear the bounds of this research area.

"**Knowledge Discovery in Databases (KDD)** is the process of identifying valid, novel, potentially useful, and ultimately understandable structure in data. This process involves selecting or sampling data from a data warehouse, cleaning or preprocessing it, transforming or reducing it (if needed), applying a data mining component to produce structure, and evaluating the derived structure."

"**Data Mining** is a step in the KDD process concerned with the algorithmic means by which patterns or models (structures) are enumerated from the data under acceptable computational efficiency limitations."

Data mining techniques can be divided into five classes:

- Predictive Modeling: An attribute in the database is predicted according to the properties of other data.
- Clustering: Data is divided into specific groups where each data corresponding to a group is more "similar" to each other than the other groups. The similarity measurement is a user specified relationship between the records.
- Dependency Modeling: Probabilistic density functions for the causal data structure underlying the database is estimated.
- Data Summarization: Interesting summaries of subsets of the data is computed such as similarities between a few attributes in the data.
- Change and Deviation Detection: The order of data in the database is considered as a starting point and sequential information is mined.

2.2 Association Rule Mining

Association rule mining is a common method used in data summarization. Associations are rules that state certain combinations of values occur with other combinations of values with a specified frequency and certainty. In this sense, association rules aim to explain the presence of some attributes according to the presence or absence of some other attributes. The problem was studied first by Agrawal et al. [3] in 1993 on a supermarket basket data, and has been widely explored to date.

2.2.1 Problem Description

Formally, let $I = \{x_1, x_2, ..., x_n\}$ be a set of distinct literals, called **items**. A set $X \subseteq I$ with $|X| = k$ is called a **k-itemset** or simply an itemset. Let a database DB be a multi-set of subsets of I. Each $T \in DB$ is called a **transaction**. We

say that a transaction $T \in DB$ supports an itemset $X \subseteq I$ if $X \subseteq T$ holds. An association rule is an expression $X \Rightarrow Y$, where X, Y are itemsets and $X \cap Y = \emptyset$ holds. The fraction of transactions T supporting an itemset X with respect to database DB is called the **support** of X, $support(X) = |\{T \in DB | X \subseteq T\}|/|DB|$ and the itemsets which have $support(X) \geq minsup$ are called **large**. The **support of a rule** $X \Rightarrow Y$ is defined as $support(X \Rightarrow Y) = support(X \cup Y)$. The **confidence of this rule** is defined as $confidence(X \Rightarrow Y) = support(X \cup Y)/support(X)$ [6].

Given a set of transactions DB, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user specified $minsup$ and $minconf$, respectively. Formally, the problem is generating all association rules $X \Rightarrow Y$, where $support(X \cup Y) \geq minsup \times |DB|$ (frequency constraint) and $support(X \cup Y)/supp(X) \geq minconf$.

2.2.2 Analysis of Algorithms

The first algorithm in this area is Apriori [6]. It is based on an iterative level-wise approach to generate and compute all large itemsets. Each level corresponds to the length of the itemsets. At the first step, all items in the transaction database are considered as candidates C^1 for that level. These candidates are checked in the transaction database to have a support greater than the minimum support threshold, $minsup$. The itemsets passing the validation belong to the large itemsets, L^1. In following levels, C^k is generated by performing a $(k-2)$ join on large itemsets of the L^{k-1} and then satisfaction of frequency constraint for these itemsets is validated in the transaction database. This process continues in this manner until $C^k = \emptyset$. Apriori uses the following fact to prune the itemsets:

> "All subsets of a large itemset must also be large."

Apriori style algorithms are decomposed into two steps [3]. At the first step all large itemsets are computed and then at the second step all these large itemset are used to generate the association rules in the transaction database.

The latter problem is simpler, since given an itemset $I = \{x_1, x_2, \ldots, x_k\}$ we can generate at most k rules of the form $\{I - \{x_a\}\} \Rightarrow \{x_a\}$ and validate them to have a confidence greater than the user specified confidence threshold $minconf$. An efficient algorithm for the generation of association rules from the set of large itemset is presented in [4].

Computation of large itemsets is a more difficult task than rule generation especially for large databases. Therefore, most of the previous research was done on the computation of large itemsets.

After the introduction of Apriori algorithm [6], researchers concentrated on eliminating the bottlenecks of this approach and carried out their studies on the following aspects:

- Reducing number of passes over the transaction database to improve I/O cost [42, 15]
- Improving large itemset generation procedure to maximize computational efficiency [36, 32, 11, 56, 31, 23, 44]
- Finding efficient distributed and parallel algorithms for association rule generation [5, 16, 57]
- Improving I/O and computational costs by using sampling techniques [58, 52]
- Applications of the large itemset generation method to different types of associations such as quantitative, generalized, sequential, cyclic and negative association rules. [47, 45, 46, 43]
- Online methods for the application of preprocess-once-query-many approach in online analytical processing (OLAP). [24, 2]
- Incremental approaches for the maintenance of discovered association rules. [17, 30, 19, 18, 49, 7, 41, 35, 50, 10, 39]
- Approaches for multi-level, hierarchical databases. [21, 28]

- Constraint-based association rules. [48, 34, 12, 33, 29, 22, 20, 37, 55, 38]
- SQL-based association rule mining. [27, 51, 40]
- Different relationships between set of itemsets in the transaction database [14, 54, 53]

For a detailed description and comparison of the previous approaches please refer to [9, 1, 26]. In the following sections, we give a broad survey on both constraint-based association rule mining and incremental association rule mining.

2.3 Constraint-based Association Rule Mining

Association rule mining is a batch process where the user sets minimum support and minimum confidence thresholds and then performs the mining step. Nothing in between is interacted with the user. However, these thresholds do not always lead to interesting and reasonably understandable results. The user should clarify his focus in the mining step. A simple solution is to perform a pre-processing on the database by cleaning the part out of focus and after the mining operation post-processing the results according to needs of the user.

Constraint-based association rule mining is proposed to make the mining process more human-oriented. This leads to ad hoc mining while increasing the performance of current approaches.

This problem is first addressed in [34] and the main problems of the present-day model of association rule mining are identified as:

- Lack of user exploration and control: Lack of user interaction with the system in the internal steps of the algorithm is a drawback of the current systems.
- Lack of focus: Ad hoc mining is a must.

- Rigid notion of relationships: Different notions of relationships other than the support and confidence metrics can also be used.

2.3.1 Problem Description

As a special case of association rule mining, constrained association is defined as a set of itemsets $\{X|X \subseteq I\&C(X)\}$, where C denotes one or more Boolean constraints. This is based on large itemset generation framework and additionally, constraints on antecedent or consequent of association rules are defined to construct rules from previously computed constrained associations.

C is a conjunction of SQL like constraints where

1. *Single Variable (1-var) Constraints*:
   (a) *Class Constraints*: $S.A \subset A$, where $S.A$ is a set of values from the domain of attribute A.
   (b) *Domain Constraints*:
      - $\forall x \in S.A, x\ \theta\ v$, where θ is one of the operators $=, \neq, <, \leq, >, \geq$ and v is a constant value.
      - $v\ \theta\ S.A$, where θ is one of the operators $\in, \notin$.
      - $V\ \theta\ S.A$, where θ is one of the operators $\in, \notin$ and V is a set of constants from the domain of attribute A.
   (c) *Aggregate Constraints*: $agg(S.A)\ \theta\ v$, where agg is one of the aggregate functions $Min, Max, Sum, Count, Avg$ and θ is one of the operators $=, \neq, <, \leq, >, \geq$.
2. *Two Variable (2-var) Constraints*:
   - $S_1.A\ \theta\ S_2$.A, where θ is one of $\subset, \not\subset, \subseteq, \not\subseteq$.
   - $(S_1.A\ o\ S_2.A)\ \theta\ V$, where θ is one of $=, \neq, \subset, \not\subset, \subseteq, \not\subseteq$, o is one of $\cup, \cap$ and V is a set of constants (including $\emptyset$).

- $agg_1(S_1.A)\ \theta\ agg_2(S_2.A)$, where agg_1, agg_2 are aggregate functions and θ is one of the operators $=, \neq, <, \leq, >, \geq$.

Assume that we store information - specific attributes such as price and type of items - about items in a transaction database. This information lets us specify some sort of constraints that are mentioned above. For instance, $S.Price \leq 1000$ selects all items in S with price less than or equal to 1000 where S is a set variable on the *Item* domain. The constraint $\{snacks, beers\} \subseteq S.Type$ returns a set on *Item* domain where each itemset in S should include some items whose type is *snacks* and some items whose type is *beers*, however $S.Type \cap \{snacks, beers\} = \emptyset$ excludes such items. These constraints are examples of 1-var constraints. 2-var constraints include two distinct sets of items to compare. For instance, $Max(S_1.Price) \leq Avg(S_2.Price)$. Complex queries can be performed by joining different constraints, i.e. $\{sodas\} \subseteq S_1.Type\ \&\ \{beers\} \subseteq S_2.Type\ \&\ Max(S_1.Price) \leq Min(S_2.Price)$ that finds pairs of sets of cheaper soda items and sets of more expensive beer items.

2.3.2 Basic Terminology

Definition 2.3.1 (Anti-monotone and monotone constraints) *A constraint C_{am} is anti-monotone if and only if for any pattern X not satisfying C_{am}, none of the super-patterns of X can satisfy C_{am}, where a super-pattern $X' \supset X$. A constraint C_m is monotone if and only if for any pattern X satisfying the constraint, C_m, every super-pattern of X also satisfies C_m.*

Lemma 2.3.1 *All subsets of an itemset X satisfying anti-monotone constraints (C_{am}) in a database D, also satisfy these anti-monotone constraints in D.*

Definition 2.3.2 (Succinct constraints)

1. *A subset of items I is a succinct set, if it can be expressed as $\sigma_p(Item)$ for some selection predicate p, in which $Item$ is the set of items, and σ is the selection operator.*

Constraint	Anti-monotone	Succinct
$\forall x \in S.A, x\ \theta\ v, (\theta \in \{=, \leq, \geq\})$	Yes	Yes
$v \in S.A$	No	Yes
$S.A \supseteq V$	No	Yes
$S.A \subseteq V$	Yes	Yes
$S.A = V$	Partly	Yes
$Min(S.A) \leq v$	No	Yes
$Min(S.A) \geq v$	Yes	Yes
$Min(S.A) = v$	Partly	Yes
$Max(S.A) \leq v$	Yes	Yes
$Max(S.A) \geq v$	No	Yes
$Max(S.A) = v$	Partly	Yes
$Count(S.A) \leq v$	Yes	No
$Count(S.A) \geq v$	No	No
$Count(S.A) = v$	Partly	No
$Sum(S.A) \leq v$	Yes	No
$Sum(S.A) \geq v$	No	No
$Sum(S.A) = v$	Partly	No
$Avg(S.A)\theta v\ (\theta \in \{=, \leq, \geq\})$	No	No

Table 2.1: Characterization of commonly used 1-var constraints [34]

2. *$SP \subseteq 2^{Item}$ is a succinct powerset, if there is a fixed number of succinct sets $Item_1, ..., Item_k \subseteq Item$, such that SP can be expressed in terms of the strict powersets of $Item_1, ..., Item_k$ using union and minus.*

3. *A constraint C is succinct provided $SAT_C(Item)$ is a succinct powerset, where $SAT_C(Item)$ is the set of itemsets that satisfy C.*

Suppose that $C_{suc} \equiv S.Price \geq 1000$ and $Item_1 = \sigma_{Price \geq 1000}(Item)$, then C_{suc} is succinct since $SAT_{C_{suc}}(Item)$ is simply 2^{Item_1}. Suppose $C_{suc} \equiv \{snacks, sodas\} \subseteq S.Type$ and $Item_1 = \sigma_{Type="snacks"}(Item)$, $Item_2 = \sigma_{Type="sodas"}(Item)$, $Item_3 = \sigma_{Type \neq "snacks" \wedge Type \neq "sodas"}(Item)$, then C_{suc} is succinct since $SAT_{C_{suc}}$ can be expressed as $2^{Item} - 2^{Item_1} - 2^{Item_2} - 2^{Item_3} - 2^{Item_1 \cup Item_3} - 2^{Item_2 \cup Item_3}$.

A representative subset of anti-monotone and succinct constraints excluding negative and strict ones are shown in Table 2.1. In the table, $S.A$ refers to a set of values from the domain of attribute A, x refers to element of $S.A$, and v is a constant value over the domain attribute values. In Table 2.1, *Partly*

is used to describe the cases of equality where greater than or equal to and less than or equal to operators are in opposite categories of constraints such as $Min(S.A) = v$ since it is equivalent to $Min(S.A) \leq v$ & $Min(S.A) \geq v$. In that case $Min(S.A) \leq v$ and $Min(S.A) \geq v$ are categorized as non anti-monotone and anti-monotone, respectively. Attribute values of itemsets in computation of the sum function must be positive, otherwise none of the sum functions are anti-monotone. Detailed discussion on 1-var constraints can be found in [34].

Definition 2.3.3 (Member Generating Functions)

1. *We say that* $SP \subseteq 2^{Item}$ *has a member generating function (MGF) provided that there is a function that can enumerate all and only elements of* SP*, and that can be expressed in the form* $\{X_1 \cup \ldots \cup X_n | X_i \subseteq \sigma_{p_i}(Item), 1 \leq i \leq n \ \& \ \exists k \leq n : X_j \neq \emptyset, 1 \leq j \leq k\}$*, for some* $n \geq 1$ *and some selection predicates* $p_1, \ldots, p_n$.

2. *A 1-var constraint* C *is pre-counting prunable provided that* $SAT_C(Item)$ *has an MGF.*

As an example, MGF is simply $\{X | X \subseteq Item_1 \ \& \ X \neq \emptyset\}$ for the constraints $S.Price \geq 1000$, where $Item_1 = \sigma_{Price \geq 1000}(Item)$ and for the constraint $\{snacks, sodas\} \subseteq S.Type$ MGF is $\{X_1 \cup X_2 \cup X_3 | X_1 \subseteq Item_1 \ \& \ X_1 \neq \emptyset \ \& \ X_2 \subseteq Item_2 \ \& \ X_2 \neq \emptyset \ \& \ X_3 \subseteq Item_3\}$, where $Item_1 = \sigma_{Type = \text{“}snacks\text{”}}(Item)$, $Item_2 = \sigma_{Type = \text{“}sodas\text{”}}(Item)$, $Item_3 = \sigma_{Type \neq \text{“}snacks\text{”} \wedge Type \neq \text{“}sodas\text{”}}(Item)$.

Lemma 2.3.2 *If* C *is succinct, then* C *is pre-counting prunable.*

Lemma 2.3.3 *Let* C_1, C_2 *be two succinct constraints with respective MGFs:* $\{S_1 \cup \ldots \cup S_m | S_i \subseteq \sigma_{p_i}(Item), 1 \leq i \leq m \ \& \ \exists m' \leq m : S_i \neq \emptyset, 1 \leq i \leq m'\}$*, and* $\{T_1 \cup \ldots \cup T_n | T_i \subseteq \sigma_{q_i}(Item), 1 \leq i \leq n \ \& \ \exists n' \leq n : T_i \neq \emptyset, 1 \leq i \leq n'\}$ *Then:* $\{R_{11} \cup \ldots \cup R_{mn} | R_{ij} \subseteq \sigma_{p_i \wedge q_j}(Item), 1 \leq i \leq m, 1 \leq j \leq n \ \& \ R_{kl} \neq \emptyset, 1 \leq k \leq m', 1 \leq l \leq n'\}$ *is an MGF for* $C_1 \& C_2$

As a construction, combined MGF for both of the constraints given above is $\{X_1 \cup X_2 \cup X_3 | X_1 \subseteq \sigma_{Type=\text{"snacks"} \wedge Price \geq 1000}(Item)$ & $X_1 \neq \emptyset$ & $X_2 \subseteq \sigma_{Type=\text{"sodas"} \wedge Price \geq 1000}(Item)$ & $X_2 \neq \emptyset$ & $X_3 \subseteq \sigma_{Type \neq \text{"snacks"} \wedge Type \neq \text{"sodas"} \wedge Price \geq 1000}(Item)\}$.

Definition 2.3.4 (Convertible constraints) *A constraint C is convertible anti-monotone if and only if (i) C is neither monotone, nor anti-monotone, nor succinct, and (ii) there exists an order R over the set of items I such that any pattern X satisfying the constraint implies that each suffix of X with respect to order R also satisfies the constraint. A constraint C is convertible-monotone if and only if (i) C is neither monotone, nor anti-monotone, nor succinct, and (ii) there exists an order R over the set of items I such that any pattern X satisfying the constraint implies that each pattern of which X is a suffix with respect to order R also satisfies the constraint. A constraint C is convertible if and only if C is either convertible anti-monotone or convertible-monotone.*

Definition 2.3.5 (Strongly convertible constraint) *A constraint C is called strongly convertible constraint, provided that there exists an order R over the set of items such that C is convertible anti-monotone with respect to R and convertible monotone with respect to inverse order R^{-1}.*

Definition 2.3.6 (Quasi-succinctness) *A constraint C is quasi-succinct if it can be reduced to two 1-var constraints C_1, C_2 such that: (i) C_1, involving only the variable S, is succinct, and is a sound and tight pruning condition for candidate S-sets; and (ii) C_2, involving only T, is succinct, and is a sound and tight pruning condition for candidate T-sets.*

2.3.3 Previous Algorithms

One of the initial studies on the concept of constraint based association rule mining is [48]. In this paper, optimization of frequent itemset generation for a set of Boolean constraints is discussed. Boolean constraints that are expressed

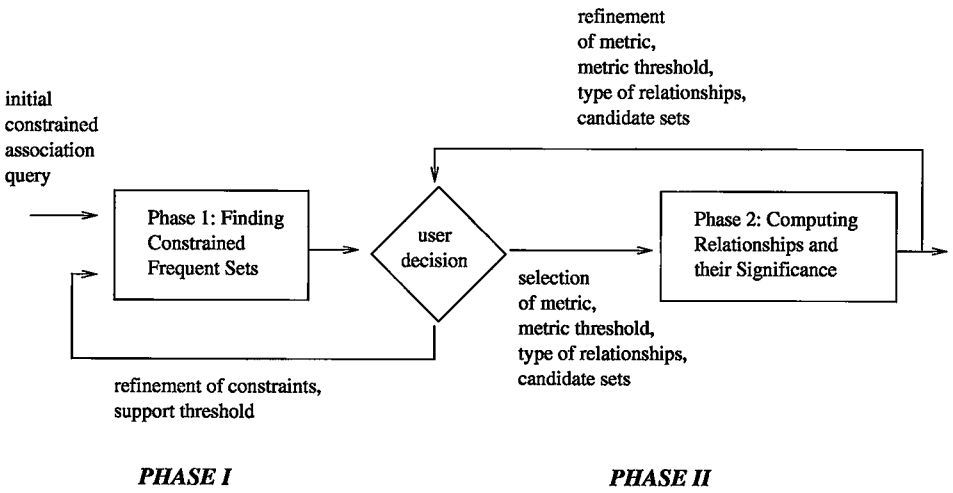

Figure 2.1: Architecture for exploratory association rule mining [34]

over a taxonomy of items and three approaches that exploit these constraints as a part of frequent itemset generation process are discussed. According to the experimental results presented in the paper, the integrated approach provides significant performance gain as compared to post-pruning by using constraints, although choice of the approach is likely to depend upon data characteristics.

Anti-monotone and succinct constraints for the purpose of interactive mining to facilitate user exploration and goal directed mining are introduced in [34, 22]. These constraints are examined to determine whether constraint verification can be pushed into the computation of frequent itemsets. The authors proposed three different algorithms for this purpose and the CAP algorithm, which exploits constraints in the mining process as early as possible, achieved better performance. An architecture (in Figure 2.1) for supporting constraint based, human centered, exploratory mining of various kinds of rules including associations is proposed and the notion of constrained frequent set queries (CFQ) is introduced. Effective pruning strategies for CFQs with 1-var constraints are developed and discussed in the same paper.

Bayardo et al. proposed Dense-Miner [12] to mine constraint based association rules in large and dense databases. This approach directly exploits user specified constraints including minimum support, minimum confidence and a new constraint that ensures every mined rule offers a predictive advantage over any

of its simplifications. The algorithm has three important search space pruning strategies:

- Functions that allow the algorithm to flexibly compute bounds on confidence, improvement and support of any rule derivable from a given node in the search tree
- Approaches for reusing support information gathered during previous database passes within these functions to allow pruning of nodes before they are processed
- An item-ordering heuristic that ensures there are plenty of pruning opportunities.

Query flocks introduced in [53] consists of a parameterized query and a filter that selects certain assignments of values for parameters by applying a condition (filter) to the result of the query for that value assignment. The techniques described in the paper apply to any monotone filter condition. In [50], this approach is generalized to compute safe sub-queries of a query flock incrementally.

Convertible constraints are identified in [37]. A constraint based frequent pattern mining method, called *constrained frequent pattern growth* (CFG), which integrates constraint pushing with a recently developed frequent pattern growth method [23] is proposed. The constraints are categorized into five classes: succinct, anti-monotone, monotone, convertible, and inconvertible.

In [38], Pei et al. classify convertible constraints into three classes: convertible anti-monotone, convertible monotone and strongly convertible (Table 2.2, where * means it depends on the specific constraint). These constraints are characterized by using the notion of prefix monotone functions and according to their arithmetical closure properties. i.e. $Avg(S.A)\ \theta\ v, Median(S.A)\ \theta\ v$, and $Sum(S.A)\ \theta\ v$, where $S.A$ can contain arbitrary values and v refers to a constant value and $\theta \in \{\leq, \geq\}$. It is shown that these types of constraints cannot be pushed deep into the basic Apriori framework, but it is possible for the

Constraint	Convertible anti-monotone	Convertible monotone	Strongly convertible
$Avg(S.A)\theta v$ $(\theta \in \{\leq, \geq\})$	Yes	Yes	Yes
$Median(S.A)\theta v$ $(\theta \in \{\leq, \geq\})$	Yes	Yes	Yes
$Sum(S.A) \leq v$ $(v \geq 0, \forall x \in S, x\theta 0, \theta \in \{\leq, \geq\})$	Yes	No	No
$Sum(S.A) \leq v$ $(v \leq 0, \forall x \in S, x\theta 0, \theta \in \{\leq, \geq\})$	No	Yes	No
$Sum(S.A) \geq v$ $(v \geq 0, \forall x \in S, x\theta 0, \theta \in \{\leq, \geq\})$	No	Yes	No
$Sum(S.A) \geq v$ $(v \leq 0, \forall x \in S, x\theta 0, \theta \in \{\leq, \geq\})$	Yes	No	No
$f(S.A) \geq v$ (f is a prefix decreasing function)	Yes	*	*
$f(S.A) \geq v$ (f is a prefix increasing function)	*	Yes	*
$f(S.A) \leq v$ (f is a prefix decreasing function)	*	Yes	*
$f(S.A) \leq v$ (f is a prefix increasing function)	Yes	*	*

Table 2.2: Characterization of convertible constraints [38]

frequent pattern growth mining approach [23]. For the case of strongly convertible constraints, when the constraint has a high selectivity (fewer itemsets satisfy it), converting it into an anti-monotone constraint will yield maximum benefits by search space pruning; when the constraint selectivity is low (and checking is reasonably expensive), then converting it into a monotone constraint will save considerable effort in constraint checking.

Lakshmanan et al. studied pruning strategies for CFQs with 2-var constraints in [29]. It is shown that few of the 2-var constraints are monotone (or anti-monotone). Therefore, a notion of quasi-succinctness is introduced and characterized as a pruning optimization. A representative subset of 2-var constraints are listed in Table 2.3 to identify anti-monotone and quasi-succinct ones. Heuristic pruning techniques are discussed for non-quasi-succinct constraints in the paper.

Efficient mining of constrained correlated sets is studied in [20]. Grahne et al. extended Brin et al.'s [14] principle of finding minimal sets to two useful semantics of answering constrained correlation queries: all minimal answers that are valid versus all minimal valid answers. Monotone, anti-monotone and succinct constraints are exploited and their interactions with the correlated sets are handled.

2-var Constraint	Anti-monotone	Quasi-succinct
$S.A \cap T.B = \emptyset$	Yes	Yes
$S.A \cap T.B \neq \emptyset$	No	Yes
$S.A \subseteq T.B$	No	Yes
$S.A \not\subseteq T.B$	No	Yes
$S.A = T.B$	No	Yes
$Max(S.A) \leq Min(T.B)$	Yes	Yes
$Min(S.A) \leq Min(T.B)$	No	Yes
$Max(S.A) \leq Max(T.B)$	No	Yes
$Min(S.A) \leq Max(T.B)$	No	Yes
$Sum(S.A) \leq Max(T.B)$	No	No
$Sum(S.A) \leq Sum(T.B)$	No	No
$Avg(S.A) \leq Avg(T.B)$	No	No

Table 2.3: Characterization of 2-var constraints [29]

2.4 Incremental Association Rule Mining

Suppose that association rules of an existing database has already been mined, and after the mining operation some updates have been performed on the database. To find out new frequent itemsets $L_{DB'}$ in the updated database DB', the naive approach is to apply a previously proposed association rule mining algorithm to updated database DB'. However, this method is not efficient, because it fails to reuse the results of the previous mining. If the size of the previous database DB is larger than the update, $DB' - DB$, then an incremental approach can perform much better than the naive approach.

2.4.1 Problem Description

We know the large itemsets in DB, L_{DB}, as a result of the previous mining operations. While we are keeping these results, some update operations can be performed on the database including insertions, deletions and modifications. We can treat the modification of existing transactions as deletion followed by insertion. If the updated patterns are not a sample of the original database then it is expected that the previous results are not valid anymore.

Thus, the update problem is to find $L_{DB'}$ efficiently, given the knowledge of $DB, db, db^-, L_{DB}, minsup$ and to compute association rules from these large itemsets.

However, modification and deletion operations are not as frequent as insertions and generally it is assumed that $db^- = \emptyset$

2.4.2 Previous Algorithms

The first study in the area of incremental mining is FUP (Fast UPdate) algorithm [17] which supports only insertions to the original database. It is a level-wise approach and makes k-passes over the whole database, where k is the cardinality of the longest large itemset. In each scan, the increment is processed initially and then the results obtained are used to guide the mining of the original database DB.

The strategy presented in FUP is as follows: All 1-itemsets are considered as candidates in the first pass over the increment. Support values of candidates that are large in DB are computed in this pass. Large itemsets having support greater than the minimum threshold $minsup$ are put in L_{DB+db}. Remaining candidates that are large only in db have chance to become large in $DB + db$. Thus, they are identified and verified to test the frequency constraint in $DB + db$ with an additional scan of original database DB. At the end of this pass, all large 1-itemsets are obtained. Candidates for the next pass are calculated using the *AprioriGen* [6] function, and the process repeats in this manner until all the large itemsets have been identified.

An incremental mining algorithm, called MLUp (Multi-Level association rules Update), for updating multi-level association rules over a taxonomy hierarchy was presented in [19]. It follows the approach presented in FUP algorithm [17] with an improvement in the candidate set generation procedure and is called FUP*, and mainly the same reasoning is proposed in [21]. MLUp restricts its attention to deriving intra-level rules, i.e. rules within each level. Different minimum

support thresholds are used for each level of the hierarchy. Both FUP* and MLUp are applicable only to a database which allows frequent or occasional updates restricted to insertions of new transactions.

FUP_2 algorithm [18] was designed to address the maintenance of association rules in a dynamic database where insertion, deletion and modification of transactions are supported. FUP_2 algorithm works as follows. Like its ancestor, it is a level-wise approach and generates large itemsets after k-passes where k denotes the maximum length of large itemsets. In the first iteration, $C^1_{DB'}$ is the set of all 1-itemsets, otherwise $C^k_{DB'}$ is generated as in the Apriori case, using *AprioriGen* function by $L^{k-1}_{DB'}$. Next, $C^k_{DB'}$ is divided into two partitions: $P^k_{DB'} = C^k_{DB'} \cap L^k_{DB'}$ and $Q^k_{DB'} = C^k_{DB'} - P^k_{DB'}$.

The candidates in the two partitions of $C^k_{DB'}$ are handled differently. $support_{DB}(X)$ is available for $X \in P^k_{DB}$ where $P^k_{DB'} \subseteq L^k_{DB'}$ from the previous mining results. db and db^- are scanned to find out $support_{db-db^-}(X)$. If $support_{DB'}(X) \geq |DB'| \times minsup$, then X is large, and hence it is added to the set $L^k_{DB'}$. For each candidate $X \in Q^k_{DB'}$, $support_{DB}(X)$ is not known, but $support_{DB}(X) < |DB| \times minsup$ can be concluded since X is a candidate. So, such a candidate can be large only if $support_{db-db^-}(X) > (|db| - |db^-|) \times minsup$. In the scan of db and db^- , $support_{db-db^-}(X)$ for the candidates $X \in Q^k_{DB'}$ is counted and the candidates with $support_{db-db^-}(X) \leq (|db| - |db^-|) \times minsup$ pruned away from $Q^k_{DB'}$. DB^- is scanned to find out the support values of the remaining candidates and these values are added to their $support_{db}(X)$. Itemsets that are satisfying $support_{DB'} \geq |DB'| \times minsup$ are added to $L^k_{DB'}$. As a result, all candidates in $C^k_{DB'}$ are processed and the algorithm proceeds to the next level unless $L^k_{DB'}$ is empty. As an improvement, DHP technique [36] is introduced into FUP_2 algorithm to hash counts of the itemsets in DB'.

DELI (Difference Estimation for Large Itemsets) algorithm [30] which uses a sampling technique to estimate the difference between the old and new association rules is based on Apriori [6] and FUP_2 [18] algorithms. This estimate is used as an indicator for whether the FUP_2 algorithm should be applied to the database to find out the new association rules accurately. If the estimated

difference is large enough (with respect to some user specified thresholds), the algorithm signals the need of an update operation, which can be accomplished by using FUP_2 algorithm. If the estimated difference is small, then we do not run FUP_2 immediately. In this case, since the difference between the old and new association rules is small, old rules can be taken as an approximation of the new rules. Experimental results given in the paper show that this approach is applicable and can bring savings on machine resources. It only scans the original database DB once, in order to pick a sample S. The size of the sample should be small enough so that it fits in main memory. Thereafter, the algorithm only examines the updates db and db^-, whose sizes are not very large.

Definition 2.4.1 (Negative Border) *It consists of all itemsets that were candidates of the level-wise method which did not have enough support. That is,* $NBd(L^k_{DB}) = C^k_{DB} - L^k_{DB}$, *where* C^k_{DB} *is the set of candidate k-itemsets and* $NBd(L^k_{DB})$ *is the set of k-itemsets in* $NBd(L_{DB})$. *Therefore,* $L^k_{DB} \cup NBd(L^k_{DB}) = C^k_{DB}$.

There are basically three approaches to generate the itemsets in NBd and find the supports of these itemsets:

- **Complete Closure:** While generating all supersets of the itemsets in NBd with unknown counts, itemsets that are known to be small are removed and none of their supersets are generated. At the end of the generation process, a final scan over $DB + db$ is performed to count the support values of the generated itemsets.

- **Layered Closure:** Closure is generated in a layered manner and supersets of k-itemsets in NBd is generated at the k-th iteration. An additional pass over $DB + db$ is performed to update the support values of these supersets and to prune the small ones in that layer before the candidate generation process.

- **Hybrid Closure:** It is a combination of the above two approaches where layered closure approach is applied initially and then, after a certain layer

the complete closure is computed. The number of layers up to which the layered closure is computed is a design parameter.

An incremental technique based on *negative border*, for maintenance of association rules when new transaction data is added to or deleted from a transaction database is proposed in [49]. It maintains the negative border that is used to decide when to scan the whole database, along with the large itemsets. Initially, it completely mines the increment *db*, that is, $L_{db} \cup NBd(L_{db})$ is computed by applying a level-wise algorithm on the increment. It can be used in conjunction with any level-wise algorithm like Apriori [6] or Partition [42]. The algorithm adopts the complete closure approach, however, it is computed only after having mined the increment. If there are itemsets in L_{DB} that are small in DB' or if there are itemsets in $NBd()$ that are large in DB', negative border is re-computed until there is no change in it. The authors mentioned that the algorithm can be easily parallelized with minimal communication and synchronization between processing nodes. In [7] Aumann et al. proposed a similar algorithm, called BORDERS, based on negative border and studied three approaches for the cases including only additions, additions and deletions, and changing support threshold. These two algorithms use the following fact:

Theorem 2.4.1 *Let X be an itemset such that $X \notin L_{DB} \cup NBd(L_{DB})$ and $X \in L_{DB'}$. Then there exists an itemset I such that $I \subset X, I \in NBd(L_{DB})$ and $I \in L_{DB'}$. That is, some subset of X has moved from $NBd(L_{DB})$ to $L_{DB'}$.*

By Theorem 2.4.1, if none of the itemsets move from negative border to the set of large itemsets, there is a need to scan the whole database. Even if an itemset moves from negative border to the set of large itemsets, there is no need to scan the database until negative border expands, since final support counts of itemsets in negative border can be easily computed. Thus, both of the algorithms require at most one scan of the original database, DB and k-passes over the increment *db*, where k is the maximum length of large itemsets.

An adaptive incremental algorithm for association rule mining is proposed in [41]. In general, it scans the incremental database twice and the original database

exactly once with the assumption that the incremental database is similar to a sample of the original database and no significant difference between the patterns in the original and the incremental databases is expected. The algorithm scans *db*, counts supports of candidates including only $L_{DB} \cup NBd(L_{BD})$ and computes L_{db}. If $L_{db} \subseteq L_{DB}$ then no new large itemset is found and the algorithm stops. If there is an itemset which exists in both L_{db} and $NBd(L_{DB})$, then it scans the increment, *db* for validation of missed candidates in *db*. If the maximum length of missed candidates in *db* is not small, it turns out to be a level-wise algorithm. Output of the algorithm is the sets of the large itemsets in $DB + db$ and only in *db*.

The framework presented in [35] is similar to the non-incremental Partition algorithm [42]. Initially, *db* is scanned with Partition algorithm. Support counts of large itemsets in DB are updated by *db* and support counts of large itemsets in *db* are updated by DB in order to find out global large itemsets. Thus, this approach requires one scan over the whole database.

An approach based on the negative border concept with anti-monotone constraints is studied in [50] in order to maintain constrained associations. Constraints satisfying anti-monotone property is categorized into four classes:

- Frequency constraint: Minimum support threshold in association rule mining.
- Item constraints: Boolean constraints defined on the items in the itemsets.
- Aggregation constraints: Aggregate functions including min,max,sum,count and avg.
- External constraints: Constraints defined on the attributes of the items while these attributes do not exist in the result set.

The algorithm presented in [49] is extended to handle these types of constraints. Frequency constraint is applied during the support counting step. Anti-monotone item constraints and anti-monotone aggregation constraints are used in the candidate generation phase. External constraints and remaining aggregation and

item constraints are applied at the data filtering stage. Integration of all these constraints in the mining algorithm is just in a generate and test manner. Additionally, SQL formulations of incremental mining based on the SQL-based mining framework discussed in [40] is presented in this paper. Applicability of this approach to problems such as mining closed sets and query flocks is discussed at the end of the paper.

An algorithm, called Carma (Continuous Association Rule Mining Algorithm), is proposed in [24] to compute large itemsets online. Transaction database is scanned at most twice to compute all large itemsets. During the first scan, Carma continuously constructs an adjacency lattice of large itemsets with respect to the scanned part of the database. Deterministic lower and upper bounds for the support values are computed for these large itemsets. User can adjust the support and confidence thresholds anytime in this scan. According to the comparison of the user's support threshold and the computed bounds for the support threshold of itemsets, global large itemsets are determined. If all large itemsets are not determined in the previous pass, then in the second scan over the database precise support values of these itemsets are determined and all small itemsets are pruned. It is possible to use this algorithm in order to continuously process a stream of transactions and to generate the resulting association rules online. However, it will need to access the original database DB if there are any locally large itemsets in the increment, even though these itemsets may not be globally large.

Ayan et al. proposed an algorithm, called UWEP (Update With Early Pruning) [10] that follows the approaches of FUP_2 [18] and Partition Update [35] for maintenance of association rules in a dynamic database. UWEP is a level-wise algorithm. Initially, 1-itemsets in db are counted as a candidate set and corresponding *tidlists* (transaction id lists) are generated for each item in db. Large itemsets in DB, that has no item in db, and their supersets are checked for largeness in $DB + db$. In each iteration, large itemsets in db and large itemsets in DB that are not counted over db are checked for largeness in $DB + db$. At the end of each iteration, candidate set is generated from the set of large itemsets obtained in that iteration. The basic advantage of this algorithm is that it prunes

the itemsets that are large in DB but small in $DB + db$ by look-ahead pruning at the beginning of the process. By promoting the itemsets that are large both in db and $DB + db$ to C^k_{db}, it reduces size of candidate set. It is shown that the algorithm generates and counts the minimum number of candidates in order to determine the set of new large itemsets. UWEP scans the original database at most once and the inserted database exactly once. According to experimental results, it is shown to be efficient in case of insertion to the transaction database. However, it is discussed that the algorithm does not perform better than the previous algorithms in case of deletion and modification.

In [39], DELTA (Differential Evaluation of Large iTemset Algorithm) is proposed for efficient incremental mining of association rules including single/multi-support and flat/hierarchical mining. It performs at most three passes over the increment and either at most one pass over the previous database for single-support or at most two scans over the previous database for multi-support cases. Hybrid closure method is used to compute the closure of negative border. Number of layers that the layered closure approach is applied before computing complete closure is chosen as two. It provides complete mining results for both the updated database and the increment.

Recently there is a study [8] on constraint-based incremental mining of association rules, called ICAP (Incremental Constrained APriori) for the case of insertions to the original transaction database. This algorithm combines the methodologies presented in incremental mining of BORDERS algorithm and constraint based mining of CAP algorithm. Negative Border concept is extended to handle anti-monotone and succinct constraints.

Definition 2.4.2 (Constrained Negative Border) *The constrained negative border of a set of frequent itemsets L, satisfying a constraint C, henceforth referred to as $CNBd(L)$ is defined as follows:*
$CNBd(L) = \{\ S | S$ is an itemset satisfying only $C - C_{freq}$ & $S' \subset S \Rightarrow S' \in L$ or S' does not satisfy $C_{suc}\ \}$, where C_{freq} refers to frequency constraint.

ICAP requires at most one pass over the original database if constrained

negative border expands. Initially, CAP is used to mine the increment with some modification to perform the computation of constrained negative border, thus it scans the increment as the length of the maximum large itemset in *db*. After mining the increment, frequencies of old large itemsets and frequencies of the itemsets in the constrained negative border of the original database is computed in *db*. According to the largeness of these itemsets in $DB + db$, the constrained negative border may or may not be recomputed.

2.5 Discussion on Association Rules

Previously proposed association rules mining approaches still have some drawbacks:

- Effect of data skewness: Effect of temporal changes (i.e. skew) in the distribution of database values between DB and *db* has not been explored in detail [32].
- Size of database: Previously proposed algorithms are experimented with synthetically generated datasets, which are large enough to fit in main memory of the systems.
- Incomplete results: Computational constraints and the number of rules generated is used to decide the value of *minsup*. Thus, we cannot be sure whether we lost important rules or not.
- Spuriousness in itemset generation: No combination of support and confidence can generate purely correct associations [1].
- Changing user requirements: Interactive approaches are necessary to set the focus of user at run time. Therefore, different interestingness measures are proposed. For a detailed survey on interestingness measures, refer to [25].
- Flat versus. hierarchical databases: Different levels of abstraction can be used to simplify the representation of associations.

- Dealing with dense data sets: Extensive number of candidates can be generated with current approaches.

In constraint based association rule mining, dominant problems other than the general problems of association rule mining is as follows:

- Item selectivity: The power of a constraint can be measured with the notion of item selectivity. It is defined as the percentage of items that satisfy the given constraint. We cannot measure the item selectivity of a constraint without examining the characteristics of items in the dataset. Thus, a constraint with low selectivity will lead to a superior pruning while a constraint with high selectivity can decrease performance of the approach.
- Item ordering: In terms of the optimizations presented for convertible constraints in previous algorithms except the frequent pattern growth approach, the order of items can play an important role in pruning steps. Different constraints may require different and even conflicting item orderings.
- Order of constraints: For an effective pruning strategy, we have to check the constraints in some order where the order is from the ones that have low selectivity to the ones that have high selectivity. Additionally, for constraints that are neither monotone, nor anti-monotone, nor succinct we have to decide on whether to preserve an ordering of items or to induce any weaker constraint where we do not have any automated approach to induce any weaker constraint.

In a maintenance system, all operations performed in a database including insertions, deletions and modifications must be considered for sake of generality. Nevertheless, most of the research done in this area has focused on the case of insertions made to a database, since it is the general case for a retail market analysis.

Optimizations used in incremental algorithms that support only insertions to a database have made use of the fact that new winners generated in the updating

process must appear and have enough support counts in the increment. However in general, this optimization does not hold in cases of deletion and modification.

Incremental approaches are generally used to decrease computation time to find the set of large itemsets in the updated database by using old mining results. These approaches perform well with respect to running a non-incremental approach over the whole updated database. However, application of these approaches is another research area that, we can not force to run the incremental algorithm after an addition of one record to a large dataset. For instance, addition of a transaction record with 20 items leads to a huge number of candidates such as 2^{20}, since any itemset in the increment will be large in *db* for any support threshold. Additionally, one transaction is not likely to change the frequent itemsets in the whole database and an unnecessary effort will be spent for the computations. Therefore, some strategies must be considered to decide on when to apply incremental approaches. Currently, there are some methodologies to measure the significance of change in frequent itemsets, and to decide on when to update, i.e.,

- Sampling techniques are proposed to measure the difference between the set of new and old frequent itemsets [30]. If the difference exceeds a user defined threshold, then the incremental approach can be performed. This strategy requires an additional pass over the original database DB to mimic the behavior of the dataset. However, this is a costly approximation.

- Incremental mining can be performed when the size of the increment exceeds a certain size or percentage of the original database. However, this approach is not applicable if the original data and the increment has the same characteristics and discovered frequent patterns remain almost unchanged.

Chapter 3

Constraint-Based Update of Large Itemsets

In this thesis, we incorporated the methodologies presented in constraint-based mining into an incremental association rule mining algorithm. Association rule mining is decomposed into two subproblems: computation of frequent patterns (itemsets) and rule generation from those frequent patterns. The former step is computationally more expensive than the latter one, thus we studied on the first problem. We mainly focus on the technical challenges in guaranteeing a level of performance that is highly related with the selectivities of the constraints in frequent itemset generation. In the previous chapter, we addressed two properties of constraints that are critical to pruning: anti-monotonicity and succinctness. Different pruning strategies for each combination of these constraints are introduced according to their properties. Finally, we describe a constraint-based incremental algorithm, called CBULI, which achieves a maximized degree of pruning for all categories of constraints. We show how CBULI algorithm incorporates these ideas by pushing constraints as deep inside the computation as possible.

3.1 Pruning Strategies for Different Types of Constraints

For each type of constraint, we use the strategy presented in [34] that exploits those constraints to the maximum extent possible. We restrict our attention to 1-var constraints in the rest of the thesis. We begin by categorizing four combinations of anti-monotone and succinct constraints as separate cases:

1. Constraints that are both anti-monotone and succinct (C_{sam})
2. Constraints that are succinct but not anti-monotone (C_{suc})
3. Constraints that are anti-monotone but not succinct (C_{am})
4. Constraints that are neither anti-monotone nor succinct (C_{none})

The following strategies are introduced for each class of constraints in order to guarantee a level of performance.

3.1.1 Constraints that are both Anti-monotone and Succinct

In Definition 2.3.3, MGF for $SAT_C(Item)$ when C is succinct is defined as $\{X_1 \cup \ldots \cup X_n | X_i \subseteq \sigma_{p_i}(Item), 1 \leq i \leq n \ \& \ \exists k \leq n : X_j \neq \emptyset, 1 \leq j \leq k\}$, for some $n \geq 1$ and some selection predicates $p_1, \ldots, p_n$. If C is also anti-monotone, then the MGF is in the form $\{X | X \subseteq \sigma_p(Item), X \neq \emptyset\}$. Therefore we can set $C_{db}^{1,C} = \{X | X \in C_{db}^1 \ \& \ X \in \sigma_p(Item)\}$ where C_{db}^1 is the set of candidate sets of size 1. According to the anti-monotone property of these constraints, $C_{db}^{1,C}$ is used to generate the itemsets in the following iterations of the algorithm.

Strategy 1
Replace C_{db}^1 in the UWEP Algorithm by $C_{db}^{1,C}$ defined above.

3.1.2 Constraints that are Succinct but not Anti-monotone

We only describe the case where the MGFs are of the form $\{X_1 \cup X_2 | X_1 \subseteq \sigma_{p_1}(Item)\ \&\ X_1 \neq \emptyset\ \&\ X_2 \subseteq \sigma_{p_2}(Item)\}$.

Strategy 2
Define $C_{db}^{1,X_1} = \sigma_1(Item)$ and $C_{db}^{1,X_2} = \sigma_2(Item)$. Define corresponding sets of frequent sets of size 1: $L_{db}^{1,X_1} = \{X | X \in C_{db}^{1,X_1}\ \&\ support_{db}(X) \geq minsup\}$, and $L_{db}^{1,X_2} = \{X | X \in C_{db}^{1,X_2}\ \&\ support_{db}(X) \geq minsup\}$.

Define $C_{db}^2 = \{\{X,Y\} | X \in L_{db}^{1,X_1}\ \&\ Y \in (L_{db}^{1,X_1} \cup L_{db}^{1,X_2})\}$, and L_{db}^2 , as usual, the set of frequent sets in C_{db}^2 , i.e., $L_{db}^2 = \{X | X \in C_{db}^2\ \&\ support_{db}(X) \geq minsup\}$.

In general, C^{k+1} can be obtained from L^k in exactly the same way as in the Apriori algorithm - with a single modification. In the classical case, a set X is in C^{k+1}, if all its subsets of size k are in L^k, i.e., $\forall X' : X' \subset X$ and $|X'| = k \Rightarrow X' \in L^k$.

We have the following modification in candidate generation. A set X is in C^{k+1}, if for all subsets:

$\forall X' : X' \subset X$ and $|X'| = k$ and $X' \cap L_1^{X_1} \neq \emptyset \Rightarrow X' \in L^k$.

As usual, $L^{k+1} = \{X | X \in C^{k+1}\ \&\ support_{db}(X) \geq minsup\}$.

3.1.3 Constraints that are Anti-monotone but not Succinct

Strategy 3
C_{db}^k is defined as in the UWEP Algorithm. Constraint satisfaction is tested before counting is done and a set $X \in C_{db}^k$ is dropped from counting if X fails C.

3.1.4 Constraints that are neither Anti-monotone nor Succinct

Strategy 4
Any weaker constraint C' can be induced from C and depending on category of C' one of the Strategies 1–3 above for the generation of frequent sets can be used. After generating the frequent sets, they are tested for satisfaction of C.

3.1.5 Multiple Constraints

Multiple constraints can be easily handled with the given strategies above. MGFs of all constraints in C_{sam} can be combined using Lemma 2.3.3 and can be applied as in Strategy 1. Also MGFs of all constraints in C_{suc} can be combined and applied as in Strategy 2. For all constraints in C_{am}, Strategy 3 can be applied. Finally new anti-monotone or succinct constraints can be induced from the constraints in C_{none} and applied as in Strategy 4.

3.2 The Proposed Algorithm CBULI

We assumed that all large itemsets in the original database, DB, are given. By using a pre-defined minimum support threshold, *minsup*, and constraints that are specified by the user, $C_{sam} \cup C_{suc} \cup C_{am} \cup C_{none}$, CBULI algorithm computes all the large itemsets in the updated database, $DB + db$, satisfying the given constraints in $C_{sam} \cup C_{suc} \cup C_{am} \cup C_{none}$.

CBULI which is presented in Figure 3.1 is iterative in itself and can be divided into five steps as mentioned in [10] with some modifications:

- Counting 1-itemsets in *db* and creating a *tidlist* for each item in *db*. Itemsets satisfying $C_{sam} \cup C_{am}$ are determined to be used in computation of large itemsets in *db*.

```
    CBULI(DB; db; L_DB; |DB|; |db|; minsup, C_sam, C_suc, C_am, C_none)
1   C^1_db = 1-itemsets in db satisfying C_sam ∪ C_am with support > 0
2   PruneSet = L^1_DB − C^1_db
3   initial_pruning(Prune_set)          % Figure 3.2
4   prepare C^1_db as indicated in Strategy 2
5   k = 1
6   while C^k_db ≠ ∅ and L^k_DB ≠ ∅ do
7         Unchecked = L^k_DB
8         for all X ∈ C^k_db do
9             if X is small in db and X is large in DB then
10                remove X from Unchecked
11                if X is small in DB+db or X does not satisfy C_am then
12                    remove all supersets of X from L_DB
13                else
14                    add X to L_DB+db
15            else if X is large both in db and DB then
16                remove X from Unchecked
17                if X satisfies C_am then
18                    add X to L_DB+db and L^k_db
19                else
20                    remove all supersets of X from L_DB
21            else if X is large in db but small in DB and X satisfies C_am then
22                find support_DB(X) using tidlists
23                if X is large in DB+db then
24                    add X to L_DB+db and L^k_db
25        for all X ∈ Unchecked do
26            if X satisfies C_sam ∪ C_am then
27                if X satisfies C_suc then
28                    find support_db(X) using tidlists
29                    if X is small in DB+db then
30                        remove all supersets of X from L_DB
31                    else
32                        add X to L_DB+db
33            else
34                remove all supersets of X from L_DB
35        k = k +1
36        C^k_db = generate_candidate(L^{k-1}_db) based on Strategy 2
37  if C_none ≠ ∅
38        L_DB+db = L'_DB+db where L'_DB+db is the subset L_DB+db which satisfies C_none
```

Figure 3.1: Pseudocode for CBULI algorithm

- Checking large itemsets in DB except the ones that are common in db and satisfying all anti-monotone constraints. These itemsets and their supersets are checked for largeness in $DB+db$ and for constraints in $C_{sam} \cup C_{am} \cup C_{suc}$.
- Checking the large itemsets in db for largeness in $DB+db$ and for constraints in C_{am}.
- Checking the large itemsets in DB that are not counted over db. These itemsets and their supersets are checked for largeness in $DB + db$ and for constraints in $C_{sam} \cup C_{am} \cup C_{suc}$.
- Generating the candidate set from the set of large itemsets obtained in the previous step using MGF of C_{suc}.

Finally, there is a pass over the large itemsets to check the satisfaction of constraints in C_{none}.

In the first step of the algorithm, support values of 1-itemsets are counted and associated *tidlists* for each itemset are generated, where a *tidlist* for an itemset is an ordered list of transaction identifiers of the transactions in which the items are present. Application of *tidlists* was introduced in [42] to compute the support of candidate k-itemsets efficiently without re-scanning the database. It is assumed that transactions are sorted according to transaction identifiers (TID) and corresponding *tidlists* are also sorted according to the order of TIDs to improve further computation on *tidlists*.

Before the second step, while counting supports of itemsets in db, these itemsets are also checked for satisfaction of constraints in $C_{sam} \cup C_{am}$ in order to eliminate the candidates that are not satisfying $C_{sam} \cup C_{am}$ both in C^1_{db} and L_{DB}. In $PruneSet$, we store old large 1-itemsets which do not exist in db or the ones which do not satisfy $C_{sam} \cup C_{am}$ to perform an initial pruning on old large itemsets. This pruning is done by the *initial_pruning* procedure (Figure 3.2) at step 3. All itemsets in $PruneSet$ is verified for largeness in $DB + db$. If there exists an itemset X whose support is 0 in db, it is true by definition that $support_{DB+db}(X) = support_{DB}(X)$. If X was small in DB, then it is also small

```
    initial_pruning(PruneSet)
1   while PruneSet ≠ ∅ do
2         X = first element of the PruneSet
3         if X is small in DB + db or X does not satisfy C_sam ∪ C_am then
4           remove X and all its supersets from L_DB and PruneSet
5         else
6           add the supersets of X in L_DB to the PruneSet
7           remove X from L_DB
8           if X satisfies C_suc then
9             add X to L_{DB+db}
10        remove X from PruneSet
```

Figure 3.2: Initial pruning algorithm

in $DB + db$, since while the number of transactions is increasing, the support of X has not changed. Otherwise, if X is large in DB, we have to check for largeness in $DB + db$.

Lemma 3.2.1 *All supersets of a small itemset X in a database D are also small in D.*

Frequency constraint (largeness) is an anti-monotone constraint defined on occurrence of itemsets. Therefore we can generalize Lemma 3.2.1 for all types of anti-monotone constraints according to Definition 2.3.1.

Lemma 3.2.2 *If an itemset X does not satisfy an anti-monotone constraint, none of its supersets satisfy this anti-monotone constraint.*

If an itemset X in DB does not satisfy any of the anti-monotone constraints, by Lemma 3.2.2 any superset of X does not satisfy this constraint in the updated database, $DB + db$. The advantage of CBULI algorithm is pruning such itemsets and their supersets from the set of large itemsets in DB as early as possible. This strategy was introduced in UWEP algorithm just for frequency constraint,

however our approach incorporates all types of anti-monotone constraints in case of early pruning.

Definition 3.2.1 *Let X be a k-itemset which contains items $I_1, \ldots, I_k$. An immediate superset of X is a (k+1)-itemset which contains the k items in X and an additional item I_{k+1}.*

In the initial pruning step, if an itemset X satisfies all anti-monotone constraints, then all immediate supersets of X in L_{DB} are added to the *PruneSet*, otherwise X and all its immediate supersets are removed from L_{DB}. The itemsets satisfying all anti-monotone constraints are checked for satisfaction of constraints in C_{suc} to be included in L_{DB+db}. This process is repeated for each itemset in *PruneSet* until it is empty. It guarantees that all itemsets in L_{DB} with a non-existing item in *db* is validated at the third step of the algorithm. This early pruning strategy improves computational efficiency in case of existence of data skewness between the old set of transactions and the incremental part of transactions.

After all the itemsets in L_{DB} that have distinct items from itemsets in *db* are counted and itemsets that do not satisfy $C_{sam} \cup C_{am}$ are eliminated from L_{DB}, C^1_{db} is recomputed using Strategy 2. Thus, itemsets that are not satisfying any of the constraints in $C_{sam} \cup C_{am} \cup C_{suc}$ are not included in the candidate set.

It should be noted that for sake of simplicity, we showed the algorithm in a compact form and did not mention multiple sets that can be generated by using MGF of C_{suc}. As given in Definition 2.3.3, the general form of an MGF is $\{X_1 \cup \ldots \cup X_n | X_i \subseteq \sigma_{p_i}(Item), 1 \leq i \leq n \ \& \ \exists k \leq n : X_j \neq \emptyset, 1 \leq j \leq k\}$, for some $n \geq 1$ and some selection predicates $p_1, \ldots, p_n$. Initially, we have to generate n sets $C^{1,X_i}_{db}, i \leq n$ where $C^{1,X_i}_{db} = \sigma_{p_i}(C^1_{db})$ and n is the number given in MGF. We have to validate each set C^{1,X_i}_{db} to compute corresponding L^{1,X_i}_{db}. These sets of large itemsets are used in the candidate generation step of the algorithm in the following iterations. One key observation here is the value of constant k given in MGF. This number defines the minimum length of itemsets that satisfy

constraints in C_{suc}. Thus, we only add large k–itemsets to L_{DB+db} where k is the number given in MGF.

The while loop in the algorithm (lines 6–36) performs the validation on the set of candidate itemsets in *db* and the remaining itemsets in L_{DB}.

We have chosen frequency constraint as a starting point to categorize the behavior of candidates if we know the supports of these itemsets at that step. Otherwise, before computing supports of itemsets, anti-monotone constraints are used to prune candidates in order to gain computational efficiency by decreasing the size of candidate set.

There are 4 cases to consider for validation of candidates which are discussed below:

For an itemset X, it can be

1. small both in *db* and in DB
2. small in *db* but large in DB
3. large both in *db* and in DB
4. large in *db* but small in DB

Lemma 3.2.3 *Let X be an itemset. If $X \notin L_{DB}$, then $X \in L_{DB+db}$ only if $X \in L_{db}$.*

Corollary 3.2.1 *Let X be an itemset. If X is small both in DB and db, then X can not be large in $DB + db$.*

Let X be a candidate k-itemset in *db*. If X is small in *db* and also small in DB, then X can not be a large itemset in $DB + db$ by Corollary 3.2.1. Therefore we just eliminate those candidates. On the other hand, if X is small in *db* but large in DB, we have to check for largeness in $DB + db$ (lines 9–15). If X does

not satisfy the frequency constraint, then X and all its supersets are found to be small in $DB + db$ by Lemma 3.2.1. Also at that step, we extended frequency constraint checking for all kinds of anti-monotone constraints to preserve the downward closure property of anti-monotone constraints (Lemma 3.2.2). Thus, we pruned X that does not satisfy any of the anti-monotone constraints and all of its supersets from L_{DB}. Otherwise, if X satisfies all anti-monotone constraints, we add it to L_{DB+db}.

Lemma 3.2.4 *Let X be an itemset. If $X \in L_{DB}$ and $X \in L_{db}$, then $X \in L_{DB+db}$.*

If X is large both in *db* and DB (lines 15–21), then by Lemma 3.2.4 X is also large in $DB + db$. Additionally we check whether X satisfies other anti-monotone constraints or not. If it satisfies all anti-monotone constraints, we add X to L_{DB+db} and L^k_{db}, otherwise we remove X and all its supersets from L_{DB}. On the other hand, if X is large in *db* but small in DB (lines 21–25), we have to check for largeness in $DB + db$. However in this case, we do not know the support of X in DB and we have to compute it using *tidlists*. Computation of support values using *tidlists* can be costlier than simply constraint checking. As we mentioned before, we apply the strategy to check for anti-monotone constraints other than frequency constraint to decrease the size of the candidate set and then perform the computation of supports by using *tislists*. If X satisfies all anti-monotone constraints including frequency constraint then we add it to L_{DB+db} and L^k_{db}.

The most important advantage of CBULI algorithm w.r.t. previously proposed constraint-based update algorithms is the decision on the selection of candidates to be included in L^k_{db}. CBULI places the candidates that are large both in *db* and $DB + db$ to L^k_{db}, while previous approaches either put all itemsets that are large in $DB + db$ in the k-th iteration or put all candidates that are large in *db* without considering if they are small or large in DB. We used the strategy presented in UWEP for selecting the candidates, and while computing the large k-itemsets in *db* we do not place the candidates that are large in *db* and small in $DB + db$. This strategy leads to a significant reduction of the number of

```
1   generate_candidate(L_db^{k-1})
2   C_db^k = ∅
3   for all itemsets X ∈ L_db^{k-1} and Y ∈ L_db^{k-1} do
4       if X_1 = Y_1 ∧ ... ∧ X_{k-2} = Y_{k-2} ∧ X_{k-1} ≤ Y_{k-1} then
5           C = X_1 X_2 ... X_{k-1} Y_{k-1}
6           if all subsets S of C is an element of L_db^{k-1} then
7               tidlist_db(C) = tidlist_db(X) ∩ tidlist_db(Y)
8               support_db(C) = |tidlist_dbC|
9               add C to C_db^k
```

Figure 3.3: Candidate generation algorithm of UWEP

candidates in *db*. However, as mentioned in UWEP, there is a possibility that a large k-itemset in DB may not be generated in C_{db}^k because of this strategy. The solution of this is also given in UWEP that a set of itemsets, called *Unchecked*, is stored to keep the large k-itemsets in DB that are not generated in *db* for further verification. Initially at step 7, all large itemsets are assigned to *Unchecked* and in the following steps while processing the itemsets in the candidate set matching ones are eliminated. At the end of step 25, all large k-itemsets which are common in both of the candidate set and L_{DB} are removed and only the ones which are not validated in DB remain. These itemsets are verified with the final for loop in lines 25–35. Since we do not know supports of these itemsets in *db*, we must compute them by using *tidlists*. Therefore, as we discussed before, we check these itemsets for satisfaction of anti-monotone constraints other than frequency constraint, $(C_{sam} \cup C_{am})$. If any of these itemsets does not satisfy any of the anti-monotone constraints, we remove all of its supersets from L_{DB}. Otherwise, we have to check them for satisfaction of constraints in C_{suc}, since these itemsets are not generated in the candidate generation part of the algorithm in the previous iterations and satisfaction of constraints in C_{suc} is not verified. After the constraint checking step, support of these itemsets are computed using *tidlists*. If they are found to be small in $DB + db$, all supersets of them are pruned from L_{DB}, else they are placed in L_{DB+db}.

In CBULI, we incorporated the candidate generation algorithm of UWEP, which is presented in Figure 3.3, based on the Strategy 2. We have used the

MGF defined in Definition 2.3.3 to prune the candidate sets in a generate-only manner. So far, we just examined the simple case where MGF is represented in the form: $\{X_1 \cup X_2 | X_1 \subseteq \sigma_{p_1}(Item)\ \&\ X_1 \neq \emptyset\ \&\ X_2 \subseteq \sigma_{p_2}(Item)\}$. This approach can be extended to handle the general form of MGF. Complete details can be found in [34]. However, we give the following example which is the same as the one discussed in [34] to clarify the methodology used in general form of MGF.

Example 3.2.1 *Suppose that the given constraint is* $\{snacks, sodas\} \subseteq S.Type$ *and the corresponding MGF is* $\{X_1 \cup X_2 \cup X_3 | X_1 \subseteq I_1\ \&\ X_1 \neq \emptyset\ \&\ X_2 \subseteq I_2\ \&\ X_2 \neq \emptyset\ \&\ X_3 \subseteq I_3\}$, *where* $I_1 = \sigma_{Type="snacks"}(Item)$, $I_2 = \sigma_{Type="sodas"}(Item)$, $I_3 = \sigma_{Type\neq"snacks" \wedge Type\neq"sodas"}(Item)$. *The strategy is to first form the sets* L_{db}^{1,X_i}, *where* $L_{db}^{1,X_i} = \{X | X \in I_i\ \&\ support_{db}(X) \geq minsup\}, 1 \leq i \leq 3$. *Then* L_{db}^2 *is computed from* $C_{db}^2 = L_{db}^{1,X_1} \times L_{db}^{1,X_2}$. *In the next step* L_{db}^3 *is computed from* $C_{db}^3 = L_{db}^2 \times (L_{db}^{1,X_1} \cup L_{db}^{1,X_2} \cup L_{db}^{1,X_3})$. *Later on computation of* L_{db}^k *and* C_{db}^{k+1} *are done as usual, with the modification in the candidate generation that* X *is in* C_{db}^{k+1}, *if and only if*
$\forall X' \subset X$ *and* $|X'| = k$ *and* $X' \cap L_{db}^{1,X_1} \neq \emptyset$ *and* $X' \cap L_{db}^{1,X_2} \neq \emptyset \Rightarrow X' \in L_{db}^k$.

As a final step in the algorithm, if there are constraints that are neither anti-monotone nor succinct then each large itemset in L_{DB+db} is checked for satisfaction of these constraints.

As shown in Figure 3.1, algorithm CBULI incorporates the strategy for handling multiple constraints and shows where the various strategies are integrated within the UWEP framework.

3.3 Implementation Details

General structures employed in CBULI are implemented as discussed in UWEP [9]. Items in a transaction database are stored with an associated unique number, therefore itemsets are stored as a list of item numbers. A list of records including

an itemset with a list of transactions where the itemset takes place and its support in this set of transactions is created. C^k_{DB} is created as a queue of this list while L^k_{DB} is created as a hash table where each entry is a queue of this list. Reasoning behind the selection of these data structures is simple; in order to minimize insertion time to C^k_{DB} and L^k_{DB}, a queue structure is implemented and in order to minimize query time for an itemset in L^k_{DB}, a hash table is implemented. It should be noted that a hash structure is not implemented for C^k_{DB}, since it is created and processed sequentially.

Initial pruning is the most important part of the algorithm, since it eliminates most of the candidates in earlier steps of the algorithm. Pruning supersets is a costly operation, since we have to find all supersets of an itemset X that are large in DB. Time and space complexity is an important factor in this case. If we choose not to store and to compute it at run time, then for finding supersets of a k–itemset X we either add one item in L^1_{DB} and check whether it is in L^{k+1}_{DB} , and repeat this process for all items in L^1_{DB} or check for each item in L^{k+1}_{DB} whether it consists of the itemset X. However, both of these approaches require unnecessary computations. Thus, we compute supersets of an itemset X while storing old large itemsets in DB and create a corresponding hash table for finding a specific itemset efficiently. A queue of itemsets with their supports and list of their supersets in DB are stored in each entry of the hash table. Basically, we do not compute supersets of any itemset, but while storing a k–itemset X in the hash table, we also store X in the superset lists of the itemsets that are $k-1$ subsets of X. According to Lemma 2.3.1, all subsets of X also satisfy anti-monotone constraints, therefore we do not make any unnecessary computations. This data structure makes the pruning of supersets easier to handle in an efficient manner. In case of pruning supersets of an itemset X, each itemset that is in the superset list of X is deleted from the hash table and this process is repeated for all sizes of supersets. Shortly, all the itemsets that have a subset X are removed from the hash table.

An inverted file is used to store items with the corresponding transaction identifiers in which the items exist. This file is created initially while large itemsets

are computed in DB and updated in CBULI. Here, we assumed that the transaction identifiers in DB and db are distinct and the ones in db are greater than the ones in DB. The importance of this file is best understood at the computation of supports of itemsets that are large in db but small in DB. With the usage of inverted file, we do not have to scan all transactions and identify whether they contain a specific itemset but we simply find the transactions containing a specific item and compute its support in DB. This strategy improves the computational efficiency with an affordable storage space.

It should be noted that usage of MGF in the candidate generation step is an important issue. As we mentioned in the previous sections, the itemsets are sorted according to item identifiers. Application of MGF as discussed in Strategy 2 can destroy the natural ordering of itemsets.

Example 3.3.1 *Assume that the given succinct but not anti-monotone constraint is* $\{soda, juice\} \subseteq S.Type$.

Assume that the itemsets are $\{A, B, C, D, E, F\}$ *and assume* $C^{1,X_1} = \{A, C, D\}$ *(soda),* $C^{1,X_2} = \{F\}$ *(juice), and* $C^{1,X_3} = \{B, E\}$ *(neither soda nor juice). And let* $L^{1,X_1} = C^{1,X_1}$, $L^{1,X_2} = C^{1,X_2}$, $L^{1,X_3} = C^{1,X_3}$. *Then* $C^2 = L^{1,X_1} \times L^{1,X_2} = \{AF, CF, DF\}$. *Assume all of the candidates are also large, meaning* $L^2 = C^2$. *Then* $C^3 = L^2 \times (L^{1,X_1} \cup L^{1,X_2} \cup L^{1,X_3})$ *and it yields several spurious itemsets such as* $C^3 = \{AFB, AFC, AFD, AFE, CFA, CFB, CFD, CFE, DFA, DFB, DFC, DFE\}$

There are two problems to consider in the candidate generation part that we have to prevent the repetition of an item in an itemset and repetition of an itemset in different item orderings. As can be seen in the example, there are some repetitions in C^3 (i.e., $AFC = CFA, AFD = DFA, CFD = DFC$), so we must eliminate them in the candidate generation step.

Preserving the item ordering in the candidate generation or using any fixed ordering can be considered as two alternative solutions. However, both of these

approaches have some problems. If we preserve the natural ordering of the candidate generation process, then in the process of constraint checking part of anti-monotone constraints, we may not find the subsets of the itemset which satisfy anti-monotone constraints. For instance, when checking AFD for constraint satisfaction, the subset FD satisfies the frequency constraint but cannot be found in L^2 since it is stored as DF. Otherwise, if we preserve the fixed ordering that the items in an itemset is sorted, then we can skip some of the candidates in the candidate generation part. For instance, assume that the itemsets $\{AFB, AFE\}$ in C^3 turn out to be frequent. These itemsets will be joined to create a candidate itemset, namely $\{AFBE\}$ in C^4, as shown Figure 3.3. However, storing itemsets in sorted order will prevent the generation of this candidate since the itemsets will become $\{ABF, AEF\}$ and they do not have the same prefix.

We have used the natural ordering of the itemsets in the candidate generation part, and additionally we stored the sorted ordering of these itemsets in a separate structure to be used in the pruning part of the algorithm.

3.4 Correctness and Completeness

In this section, we show that CBULI algorithm presented in Figure 3.1 is sound and complete; it discovers all and only those itemsets that are constrained frequent by satisfying all given constraints. We rely on the correctness of the strategies presented in [34].

Lemma 3.4.1 *Given sets of old (DB) and new (db) transactions with a set of frequent itemsets (L_{DB}) in DB and user specified constraints ($C_{sam} \cup C_{am} \cup C_{suc} \cup C_{none}$), CBULI algorithm is sound and complete with respect to computing all constrained frequent itemsets in $DB + db$.*

Proof. Let X be a k–itemset. In Figure 3.4 , all possible cases for a frequent itemset in $DB + db$ are presented. In order to be frequent in $DB + db$, an itemset X must be large in at least one of the transaction databases by Corollary 3.2.1.

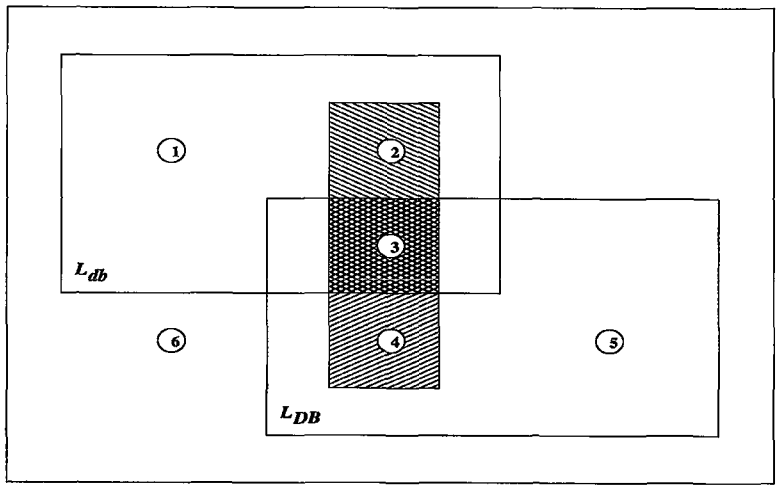

Figure 3.4: Possible cases for an itemset to be a constrained frequent itemset

Therefore, we do not consider the itemsets that are small in both of DB and db (region 6). We have to check frequent itemsets in DB with respect to db, and frequent itemsets in db with respect to DB.

- **If** $X \in L_{DB}$ (region 3, 4 and 5)
  Itemsets that are large in DB but have $support_{db}(X) = 0$ or do not satisfying all constraints in $C_{sam} \cup C_{am}$ are checked for frequency constraint in the initial pruning procedure (Figure 3.2). If an itemsets X does not satisfy any of the anti-monotone constraints ($C_{sam} \cup C_{am} \cup$ frequency constraint), all of its supersets are pruned from further consideration by Lemma 3.2.2. If X satisfies all anti-monotone constraints, we put its immediate supersets into $PruneSet$ and check X for the constraints in C_{suc}. If X satisfies all of those constraints, we put it into L_{DB+db}. This process is repeated until $PruneSet$ is empty. At the end of initial pruning step, L_{DB+db} contains the constrained frequent itemsets including the itemsets whose supports are zero in db, and L_{DB} contains the frequent itemsets in DB with $support_{db}(X) \geq 0$ and the ones that are satisfying all constraints in $C_{sam} \cup C_{am}$.

  $Unchecked$ is initialized to the set of large k–itemsets in DB, L^k_{DB}, in line 7. If an itemset in C^k_{db} is also present in $Unchecked$, it is identified on lines 9 and 15. If any large itemset X in L_{DB} is small in C^k_{db}, X is checked for satisfaction of all anti-monotone constraints. If it satisfies all

these constraints, it is stored in L_{DB+db}, else all supersets of X are removed from L_{DB}. If any large itemset X in L_{DB} is large in C^k_{db}, it is also large in $DB + db$ by Lemma 3.2.4 and we check it for satisfaction of all constraints in $C_{sam} \cup C_{am}$ to be included in L_{DB+db} and L^k_{db}. There is no need to check itemsets that are present in C^k_{db} for satisfaction of constraints in C_{suc} explicitly, since the candidates in C^k_{db} are generated using the MGF of C_{suc}.

Remaining itemsets that do not exist in C^k_{db} are considered in the for loop on line 25. Each itemset is checked for satisfaction of all anti-monotone constraints. If any of them does not satisfy any of these anti-monotone constraints, this itemset and its supersets are pruned from L_{DB}. Itemsets satisfying all anti-monotone constraints are also checked for satisfaction of constraints in C_{suc} to be included in L_{DB+db}.

Thus, all the frequent itemsets in DB are checked for satisfaction of all types of anti-monotone and succinct constraints including the frequency constraint.

- **If** $X \in L_{db}$ (region 1, 2 and 3)
  C^1_{db} is prepared as indicated in Strategies 1, 2 and 3 to satisfy all constraints in $C_{sam} \cup C_{am} \cup C_{suc}$ in lines 1 and 4.

  As discussed in UWEP algorithm [9], we promoted the itemsets that are possibly large both in db and $DB + db$ to C^k_{db}. Itemsets in C^k_{db} that are large in $DB + db$ are stored in L^k_{db} on lines 18 and 24. By Lemma 3.2.1, if a k–itemset X is large in db but small in $DB + db$, in any case, all supersets of X must also be small in $DB + db$. Therefore, it is useless to store these itemsets in L^k_{db}.

  All itemsets in C^k_{db} are identified in the first for loop. Itemsets that are common in both L^k_{DB} and C^k_{db} are identified and processed between lines 9 and 21. Additionally, if any large itemset in C^k_{db} is small in DB (lines 21–25), it is checked for satisfaction of all anti-monotone constraints to be included in L_{DB+db}.

Finally, satisfaction of constraints in C_{none} is validated for all itemsets in L_{DB+db}.

DB	
TID	**Items**
1	A,C,D,E,F
2	B,D,F
3	A,D,E
4	A,B,D,E,F
5	A,B,C,E,F
6	B,F
7	A,D,E,F
8	A,B,D,F
9	A,D,F

db	
TID	**Items**
10	A,F
11	B,C,F
12	A,C
13	B,F
14	A,B,C
15	A,C,D

Item	**Type**	**Price**
A	soda	30
B	snack	25
C	soda	40
D	soda	45
E	dairy	30
F	juice	35

Table 3.1: A database example

Since CBULI algorithm counts all itemsets that need to be considered in $DB + db$ and checks these candidates for satisfaction of the given constraints, it follows that CBULI is sound and complete.

3.5 Running Example

In this section, we will give a representative example to examine the characteristics of CBULI algorithm. In the following example, we will write an itemset including items $X_1, X_2, \ldots, X_n$ as $X_1X_2 \ldots X_n$. A pair (itemset, support) refers to an itemset with its support in the transaction database.

Example 3.5.1 *Two sets of transactions are given in Table 3.1 with the information about the items. Assume that the minimum support threshold is given as* 0.3 *with the constraints* $\{soda\} \subseteq X.Type$ *and* $Min(X.Price) \geq 30$. *Here* $|DB| = 9, |db| = 6, |DB + db| = 15$ *and for an itemset to be large in* DB, db *and* $DB + db$ *it must be listed in at least 3, 2 and 5 transactions in the corresponding datasets respectively.*

We assumed that the frequent itemsets in DB are counted by an association rule mining algorithm and these frequent itemsets with their counts are given as follows:

$L^1_{DB} = \{(A,7),(B,5),(D,7),(E,5),(F,8)\}$
$L^2_{DB} = \{(AB,3),(AD,6),(AE,5),(AF,6),(BD,3),(BF,5),(DE,4),(DF,6),(EF,4)\}$
$L^3_{DB} = \{(ABF,3),(ADE,4),(ADF,5),(AEF,4),(BDF,3),(DEF,3)\}$
$L^4_{DB} = \{(ADEF,3)\}$

Transaction database db is added to old transaction database DB, thus the set of large itemsets in DB is no longer valid through updated database. The problem is to find the set of large itemsets that satisfy the given constraints in DB + db by using the information about large itemsets in DB. While we are computing constrained frequent itemsets in DB + db, some large itemsets in DB can be pruned and some new constrained frequent itemsets can be added to the final result set.

Execution of the algorithm is described below.

Initially *db* is scanned to find the support of 1-itemsets satisfying the constraints in $C_{sam} \cup C_{am}$. While scanning *db*, *tidlists* for each 1-itemset in *db* is generated. In the example dataset, the set of 1-itemsets satisfying the constraints in $C_{sam} \cup C_{am}$ with their counts in *db* are
$C^1_{db} = \{(A,4),(C,4),(D,1),(F,3)\}$

In the second step of the algorithm, the old large itemsets that have items whose support is zero in *db* or the ones that are not satisfying the constraints in $C_{sam} \cup C_{am}$ are assigned to $PruneSet$ in order to perform an initial pruning.
$PruneSet = \{(B,5),(E,5)\}$

B is found to be large in $DB + db$, since it has a support of 5. But it has a price of 25 which is less than 30, thus it does not satisfy C_{sam} and we prune B and all its supersets namely AB, BD, BF, ABF, BDF from L_{DB}. For each element of $PruneSet$, we repeat the same operation. E is also found to be large in $DB + db$, since it has a support of 5. Additionally, it satisfies C_{sam} and we add all its supersets in L_{DB}, namely $AE, DE, EF, ADE, AEF, DEF, ADEF$ to $PruneSet$. However E has a type of *dairy* and does not satisfy C_{suc}, thus we remove it from L_{DB}. We put AE into L_{DB+db}, since it is large in $DB + db$ and it satisfies $C_{sam} \cup C_{suc}$. DE, EF, ADE, AEF, DEF and $ADEF$ fail to qualify the

frequency constraint, thus we remove them from L_{DB}.

After the initial pruning step, remaining large itemsets and new large itemsets are:
$L^1_{DB} = \{(A,7),(D,7),(F,8)\}$
$L^2_{DB} = \{(AD,6),(AF,6),(DF,6)\}$
$L^3_{DB} = \{(ADF,5)\}$
$L_{DB+db} = \{(AE,5)\}$

C^1_{db} is recomputed as in Strategy 2. The corresponding MGF for the given succinct only constraint, $\{soda\} \subseteq X.Type$ is defined as $\{X_1 \cup X_2 | X_1 \subseteq \sigma_{Type="soda"}(Item)\ \&\ X_1 \neq \emptyset\ \&\ X_2 \subseteq \sigma_{Type\neq"soda"}(Item)\}$. Therefore C^1_{db} is divided as $C^{1,X_1}_{db} = \sigma_{Type="soda"}(Item)$ and $C^{1,X_2}_{db} = \sigma_{Type\neq"soda"}(Item)$.
$C^{1,X_1}_{db} = \{(A,4),(C,4),(D,1)\}$
$C^{1,X_2}_{db} = \{(F,3)\}$

Unchecked is initialized to the set of remaining large 1-itemsets in L^1_{DB}.
$Unchecked = \{(A,7),(D,7),(F,8)\}$

Each candidate in both C^{1,X_1}_{db} and C^{1,X_2}_{db} is checked for largeness in db to be included in L^{1,X_1}_{db} and L^{1,X_2}_{db} respectively. Frequent candidates in C^{1,X_1}_{db} and *Unchecked* are promoted to be included in L_{DB+db}, since the minimum length of itemset to be included in the result set according to MGF of C_{suc} is 1. At the end of the first iteration, we have the following information
$L^{1,X_1}_{db} = \{(A,4),(C,4)\}$
$L^{1,X_2}_{db} = \{(F,3)\}$
$L^1_{DB+db} = \{(A,4),(C,6),(D,8)\}$

The candidate generation procedure generates the 2-sized candidates using the MGF given above. ($C^2_{db} = L^{1,X_1}_{db} \times (L^{1,X_1}_{db} \cup L^{1,X_2}_{db})$) and we begin the second iteration with the set of candidates where,
$C^2_{db} = \{(AC,3),(AF,1),(CF,1)\}$
$Unchecked = \{(AD,6),(AF,6),(DF,6)\}$

AC is found to be large in db but small in DB. We have to compute the support of AC in DB by using the *tidlists* of A and C. Intersection of the $tidlist_{DB}(A) = \{1,3,4,5,7,8,9\}$ and $tidlist_{DB}(C) = \{1,5\}$ is $tidlist_{DB}(A) \cap tislist_{DB}(C) = \{1,5\}$ thus the support of AC in DB is 2. Total support of AC in $DB + db$ is 5 and it is found to be large in $DB + db$. Since AC qualifies all the constraints, it is added to both L_{db} and L_{DB+db}. AF is small in db but large in DB. Support of AF is computed using *tidlists* in db ($tidlist_{db}(A) \cap tislist_{db}(F) = \{10\}$) and is found to have a total support of 7 in $DB+db$. Thus, AF is large in $DB+db$ and we put it into L_{DB+db} but not in $\overset{2}{}_{db}$. CF is found to be small in both DB and db, therefore we directly prune it from further consideration by Lemma 3.2.1.

AD and DF are found to be large in DB, but we do not know the support of these itemsets in db. Support of these itemsets are counted using *tidlists* and are found to be 1 and 0 in DB respectively. Therefore the total support of AD is 7 and the total support of DF is 6 and both of them are found to be large in $DB + db$. They also satisfy the given constraints and we put them into L_{DB+db}.

At the end of the second iteration, it is found that
$L^2_{db} = \{(AC,3)\}$
$L^1_{DB+db} = \{(A,6),(C,6),(D,8)\}$
$L^2_{DB+db} = \{(AC,5),(AD,7),(AE,5),(AF,7),(DF,6)\}$

We proceed the third iteration with
$C^3_{db} = \emptyset$
$Unchecked = \{(ADF,5)\}$

Since $C^3_{db} = \emptyset$, we only check the elements in *Unchecked*. Support of ACF is 0 in db, and 5 in DB, thus it is large in $DB + db$ and we put it into L_{DB+db}. Finally, the result of CBULI algorithm is:
$L^1_{DB+db} = \{(A,6),(C,6),(D,8)\}$
$L^2_{DB+db} = \{(AC,5),(AD,7),(AE,5),(AF,7),(DF,6)\}$
$L^3_{DB+db} = \{(ADF,5)\}$

3.6 Theoretical Performance Evaluation

Incremental association rule mining algorithms are proposed to eliminate the repetition of previously performed computations done on the set of old transaction database. Thus, it is always expected that an incremental algorithm performs better than re-running an association rule mining algorithm such as Apriori [6] on the whole set of transaction database.

Recently proposed algorithms try to minimize the number of scans over the transaction databases to decrease I/O time and gain computational efficiency. In general, one pass over the whole transaction database is sufficient to generate all frequent itemsets in the updated database. Thus, CBULI scans *db* once and *DB* at most once, meaning that the algorithm is linear in terms of the database size.

Usage of anti-monotone and succinct constraints in the counting step of the frequent itemsets, let the user specify his focus on the domain. By straightening the focus, size of the transaction database to be considered gets smaller and fewer candidates are generated. These two types of constraints play an important role in terms of pruning the candidates. While anti-monotone constraints are used to test the generated itemsets, succinct constraints are used to prune before counting. Pruning done by anti-monotone constraints are similar to the frequency constraint and each itemset is checked for satisfaction of all anti-monotone constraints by Lemma 2.3.1. However, each succinct constraint has its MGF to be used in the candidate generation part in a generate only manner by Lemma 2.3.2. It should be noted that multiple MGFs can be joined to create a resulting MGF to be used in the candidate generation part by Lemma 2.3.3.

In CBULI, initially old large itemsets that do not satisfy any of the anti-monotone constraints are pruned with their supersets from the set of old large itemsets. This initial pruning eliminates some of the old large itemsets from further verification steps in $DB + db$. However, currently proposed ICAP verifies all old large itemsets in $DB + db$, which is expected to be costlier than the initial pruning of the itemsets in L_{DB}.

As in UWEP, the k–sized candidates that are frequent both in db and $DB+db$ are stored in L_{db}^{k} with the modification that these itemsets are also checked for satisfaction of all anti-monotone constraints. This yields a much smaller number of candidates. However, ICAP algorithm promotes frequent k–itemsets satisfying all anti-monotone constraints in db to L_{db} without using the knowledge of old large itemsets. Thus, it may generate and count many itemsets which will be pruned in the following steps of the algorithm.

Storage and computation of the constrained negative border in ICAP is a costly operation, especially if the constraint negative border expands it must be recomputed. Similarly, this argument is valid for the approach presented in [50] when the negative border expands. On the other hand, the most important property of the (constrained) negative border is that it can be used to indicate if there are changes between the set of old constrained frequent itemsets and the set of new constrained frequent itemsets. Therefore, if negative border does not expand, there is no need to scan DB.

Chapter 4

Experimental Results

4.1 Notes on Experimental Environment and Performance Measure

In the experiments, we evaluate the relative efficiency of UWEP, CAP and CBULI. We conducted the experiments on UWEP, which is an incremental but not constraint-based algorithm, by post-pruning the resulting frequent itemsets according to the given constraints. In case of CAP, which is a non-incremental constraint based algorithm, we run the algorithm over the whole dataset (old + the increment).

We assumed that initially all frequent itemsets in DB are provided to the incremental mining algorithms. Here we decide on supplying the old large itemsets in DB with an association rule mining algorithm, like Apriori. Constraints are only employed in the update step in order to satisfy continuity of the constraints in the whole database.

We have tested all the algorithms on a IBM compatible PC with single 650Mhz Pentium III processor and 128MB main memory running Mandrake-Linux 8.0. In the experimental phase, the program was the major process running on the machine in order to prepare a stable environment and get reliable results.

Parameter	Value
Number of transactions (D)	100000
Number of maximal potentially large itemsets (L)	1000
Number of items (N)	1000
Average size of the transactions (T)	10
Average size of maximal potentially large itemsets (I)	4

Table 4.1: Parameters used in data generation

Databases are constructed by using the synthetic dataset generator program developed by IBM Almaden Research Center. This program uses several parameters to construct a synthetic dataset and details of these parameters can be found in [6]. Our data generation procedure is slightly different from the methodology presented in [6] since we need two synthetic datasets, the original one and the increment. We generated a transaction database with double the size of $|DB|$. The first $|DB|$ transactions are stored as the original transactions, and x% of the remaining transactions are considered as x% increment.

Example 4.1.1 *If we create a dataset of size 200, the first 100 transactions are stored as the original dataset, DB. Transactions between 100 and 110 are stored as 10% increment, transactions between 100 and 120 are stored as 20% increment, and so on.*

Since all the transactions are generated using the same statistical pattern, all subsets of the generated dataset have similar characteristics, such as the original dataset and the increment with any size.

Parameters used in our experiments are listed in Table 4.1. We follow the notation $Tx.Iy.Dm.dn$ used in [17] to denote databases in which $|DB| = m$ thousands, $|db| = n$ thousands, $|T| = x$ and $|I| = y$. We generated two minable views as $trans(TID, Itemset)$ and $itemInfo(Item, Type, Price)$. $trans$ is generated by the synthetic data generator program. In $itemInfo$, each item is represented with an item identifier ranging from 0 to 999, item prices are chosen uniformly between 1 and 1000, and item types are randomly assigned from the domain

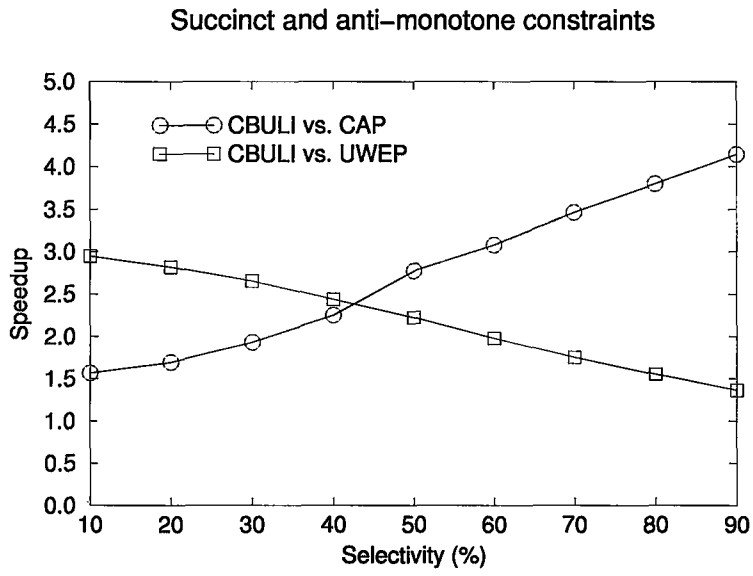

Figure 4.1: Performance comparison for succinct and anti-monotone constraint

attributes of [a–z] where each character represents a distinct type.

As a performance measure, we conducted our experiments on speedup charts using different support values, different increment sizes for scalability and *item selectivity*, where item selectivity means that x% of the items satisfy the given constraint. Speedup is a fair measure than running time of the algorithms since timing highly depends on the performance of the system and the data structures implemented in the algorithms.

4.2 Experiments with Different Query Types

In the first set of experiments, we measured the relative performance of the algorithms for each type of anti-monotone and succinct constraints. While we experimented with various databases, the results discussed below are based on a database of 100,000 transactions with an increment of 10,000 transactions (T10.I4.D100.d10). Transaction databases are generated as discussed in the previous section using the parameters listed in Table 4.1 and minimum support threshold is set to 0.5%.

In these experiments, we expect CBULI to perform better than CAP with high selectivity of the constraints. This is because of the mining framework presented in these algorithms. CAP is a non-incremental approach and perform the mining operation on the whole set of data. In that case, the number of candidates that satisfy the constraints in the first few iterations dominates the whole process. If selectivity of constraints is low, number of candidates to consider is small, otherwise if the selectivity is high, number of candidates can be very large. On the other hand, CBULI is an incremental algorithm and uses the results of previous mining operations to find the new set of constrained associations. Thus, it has to verify all frequent itemsets in the original database and the candidates in the incremental part. If selectivity of constraints is low, verification of all old frequent itemsets and the candidates in the increment can be a costly operation and decreases the performance of this approach over CAP, since most of the old frequent itemsets are not generated in the earlier levels of CAP algorithm by using the constraints. If selectivity of constraints is high, then CBULI is expected to perform much better than CAP.

For the case of CBULI versus UWEP, characteristics of performance chart with item selectivity is opposite of the performance chart of CBULI versus CAP. These two algorithms are based on the same incremental mining framework, but the pruning optimizations employed in CBULI for each type of anti-monotone and succinct constraints determine the relative performance of these algorithms. CBULI is expected to perform better than UWEP with low item selectivity, while performance gain of CBULI over UWEP decreases slowly with an increase in item selectivity.

In the first step, the succinct and anti-monotone constraint $Max(S.Price) \leq v$ is used to find the constrained frequent sets. In Figure 4.1 speedup of CBULI versus CAP and UWEP is plotted as a function of item selectivity. Since the *Price* values of the itemsets are uniformly chosen between 1 and 1000, different values of v corresponds to different selectivities of the constraint.

CBULI runs about 4 times faster than CAP for a 90% selectivity. If the selectivity gets smaller, than the performance of CAP gets closer to CBULI.

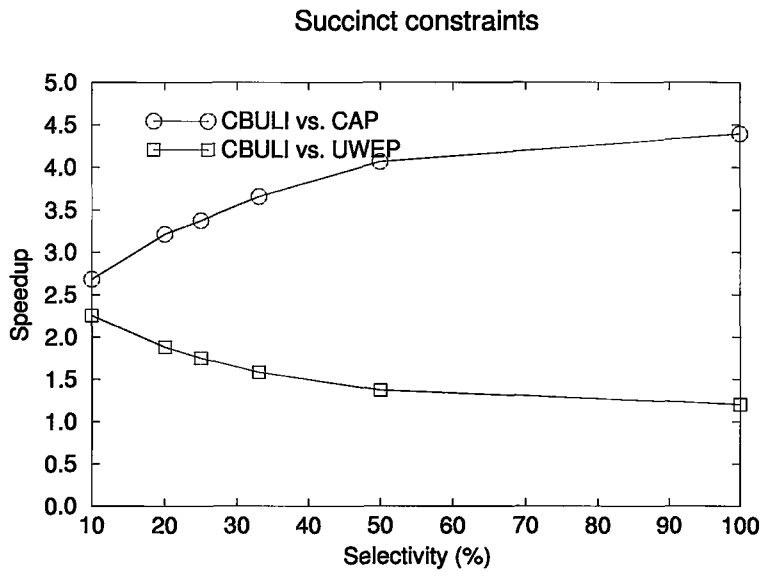

Figure 4.2: Performance comparison for succinct but not anti-monotone constraint

On the other hand, for a 10% selectivity CBULI runs about three times faster than UWEP. Performance of both of CBULI and UWEP gets closer when the selectivity increases to a level of 90%..

In the next step of the experiment, we use the succinct but not anti-monotone constraint $\{x\} \subseteq S.Type$. Figure 4.2 shows the speedup for CBULI over CAP and UWEP. While the degree of pruning performed by succinct constraints is minimized with high selectivity, the speedup of CBULI is about 4 times over CAP algorithm with high selectivity. On the other hand, for 10% selectivity CBULI runs about twice faster than UWEP.

We also tested the performance of the succinct constraints with different support thresholds ranging from 0.2 to 1.0. This range mimics the limits of the dataset, since below 0.2 the number of candidates increases drastically and above the threshold of 1.0, the number of candidates and, thus the large itemsets are small to represent a fair timing comparison. Results of this experiment are shown in Figure 4.3. CBULI performs 6 times better than CAP with a support of 0.2. However, performance of our algorithm decreases with increasing support threshold since the number of candidates to counted plays an important role in the

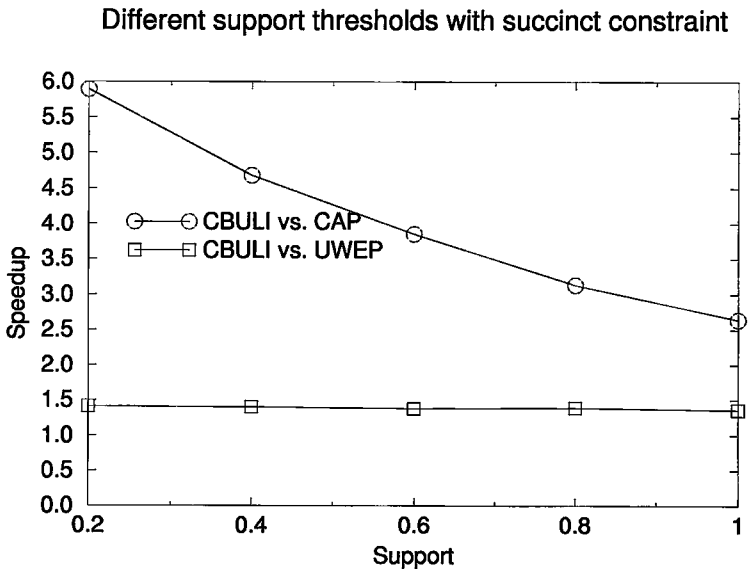

Figure 4.3: Performance comparison for different support thresholds

computation of frequent itemsets and larger minimum support threshold causes smaller number of candidates.

Succinct constraints are used in generate-only manner. The number of candidates in the first few iterations of the frequent itemset generation procedure dominates the whole set of candidates. Thus, pruning done by the succinct constraints in the first few iterations plays an important role on the size of the candidate set leading to an order of performance gain. Additionally, effect of a succinct constraint in the pruning step is much stronger if the constraint also satisfies the anti-monotone property.

Next we consider the category of constraints, that are anti-monotone but not succinct. According to the constraint $Sum(S.Price) \leq MaxSum$ we have chosen, unlike the previous cases, the notion of item selectivity does not directly make sense. We numbered the price values of each item as its item number, and compared with a $MaxSum$ value. Speedup results are shown in Figure 4.4. Similar to the previous cases, performance of CBULI gets better than CAP with the increasing value of $MaxSum$, while the performance of CBULI gets closer to UWEP.

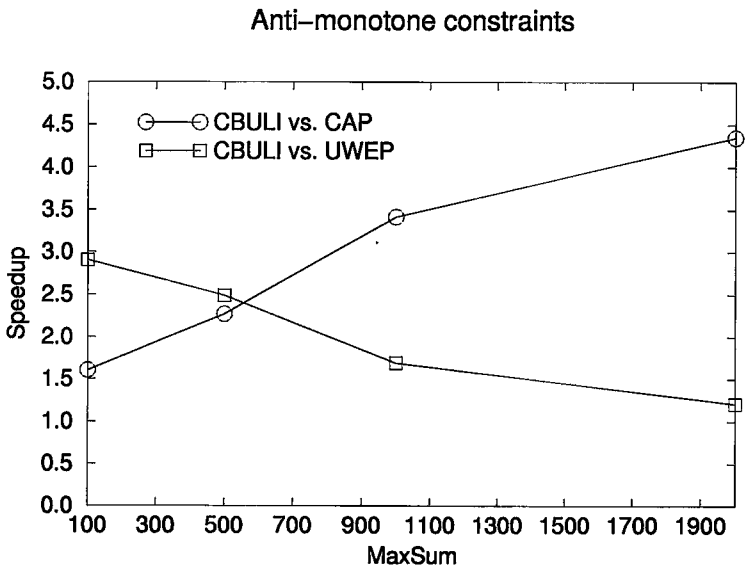

Figure 4.4: Performance comparison for anti-monotone but not succinct constraint

Anti-monotone constraints satisfy the downward closure property of frequency constraint which is the basic pruning strategy used in classical algorithms. Pruning is performed in a generate and test style that each candidate is checked for satisfaction of anti-monotone constraints. Since the number of candidates in the first few iterations (especially in the first two levels) is larger than the following steps, pruning done by the anti-monotone constraints must be significant in these levels.

Finally, we consider the constraint $Avg(S.Price) \leq v$. We can not directly use the presented optimizations for anti-monotone and succinct constraints since it is neither succinct nor anti-monotone. But as shown before, the constraint induces the weaker constraint $Min(S.Price) \leq v$, which is succinct but not anti-monotone. Extra cost is that a final verification step is needed to check whether the frequent sets really satisfy $Avg(S.Price) \leq v$. Therefore, performance of the algorithm with this kind of constraints is similar to succinct constraints as shown in Figure 4.2.

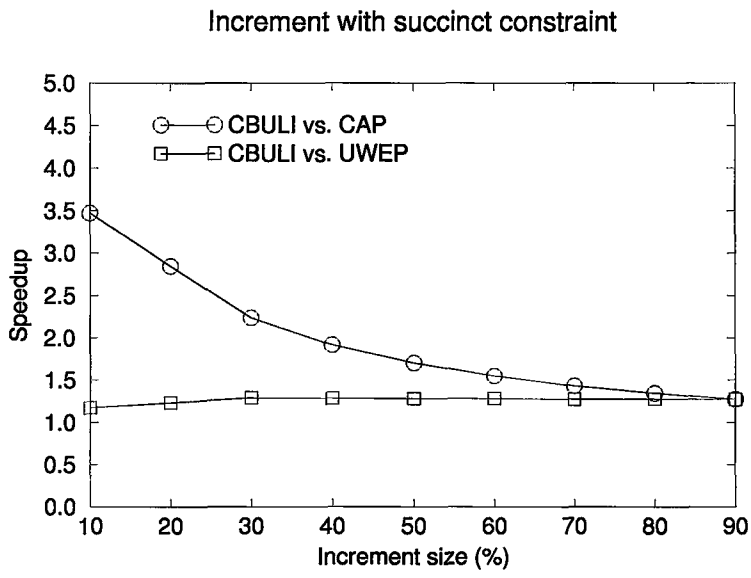

Figure 4.5: Performance comparison for different increment sizes with succinct constraint

4.3 Scalability with Database Size

We conducted the second set of experiments to measure the scalability of our approach with changing database and increment sizes. In the following experiments we applied the succinct constraint, $x \subseteq S.Type$ with an item selectivity of 50%. Minimum support threshold is set to 0.5%.

In the first step, we changed the size of incremental database. A sample database of T10.I4.D100.dx is used where x changes over 10% to 100% of the original database size. We expect our approach to perform well if the increment size is smaller than the size of original database. As shown in Figure 4.5, CBULI performs well if the size of the increment is smaller than the original size.

We measured the scalability of our approach in Figure 4.6 by changing the database size ranging from 100,000 to 500,000 transactions with 10% increment size of the original database. CBULI is expected to perform better when compared to other algorithms with an increase in the size of the original database. According to the experimental results, speedup of CBULI over the CAP algorithm converges to 6 when the database size is 500,000 transactions. However,

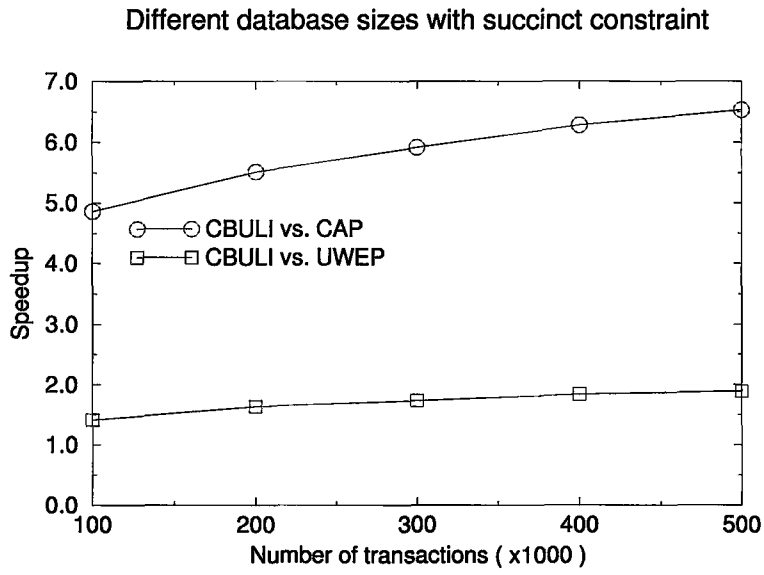

Figure 4.6: Performance comparison for different database sizes

results do not reflect the expectations about performance gain of CBULI versus UWEP, since most of the new frequent itemsets in *db* can satisfy the given constraint and the initial pruning performed by both of the algorithms is effective in terms of eliminating the old frequent itemsets in earlier steps.

Chapter 5

Conclusion and Future Work

Association rules have a wide area of usage. Maintenance of association rules is also an important issue, especially in dynamic databases. Instead of computing all the rules again, efficient algorithms are proposed which uses the previously discovered rules to find the set of new association rules.

Motivated by the problems of the association rule mining systems, such as lack of user exploration and control, lack of focus and rigid notion of relationships, we proposed a constraint-based incremental association rule mining algorithm. It is based on the strategies presented in UWEP and CAP algorithms. The algorithm ensures that anti-monotone and succinct constraints are pushed deep inside the mining process, to achieve a maximized degree of pruning. Thus, it fills the gap between constraint-based association rule mining and incremental association rule mining.

We presented a theoretical performance evaluation of our algorithm where we discussed the time complexity of the strategies presented in the thesis, and compared our approach with the existing approaches. Moreover, several experiments were performed to measure the relative performance of CBULI compared to running CAP over the whole dataset and UWEP with a filtering stage. Experimental results indicate that the optimizations employed in CBULI provides considerable computational efficiency compared to CAP. However, we could not get a large

speedup gain over UWEP algorithm in some of the experiments, since we used constraints with 50% selectivity which can be considered as high.

We believe that ad hoc mining is crucial in database mining systems. There are still some points to consider in order to get an effective approach. The user must have knowledge about the dataset to set the focus by using constraints. The performance of the system highly depends on the type of the constraints, and its selectivity. Another factor that determines the performance of the algorithm is the number of constraints that are pushed into the mining process. If the selectivity is high, and the number of constraints is large then the algorithm spends much time for the constraint checking part.

In this thesis, we only discussed 1-var anti-monotone and succinct constraints. Analysis of other constraints that are discussed in Section 2.3 involving more than one variable is expected to lead to interesting results in our approach.

It is discussed in [10] that the optimization strategies presented in UWEP do not work in case of deleted transactions. Although for a retail market dataset insertions can be considered as the only operation to be performed, in a general maintenance approach deletions and modifications must also be considered. The problem of maintenance can be a challenging problem if the modification is done both on the transactions and on the characteristics of the items in the dataset.

Bibliography

[1] C. C. Aggarwal and P. S. Yu. Mining large itemsets for association rules. In *Data Engineering Bulletin*, pages 23–31, 1998.

[2] C. C. Aggarwal and P. S. Yu. Online generation of association rules. In *Proc. of the Intl. Conf. on Data Engineering (ICDE'98)*, Orlando, Florida, February 1998.

[3] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of ACM SIGMOD Intl. Conference on Management of Data (SIGMOD'93)*, pages 207–216, Washington, May 1993.

[4] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. *Advances in Knowledge Discovery and Data Mining*, chapter Fast discovery of association rules, pages 307–328. AAAI/MIT Press, 1996.

[5] R. Agrawal and J. C. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962–969, 1996.

[6] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. of the 20th Intl. Conf. on Very Large Databases (VLDB'94)*, pages 478–499, 1994.

[7] Y. Aumann, R. Feldman, O. Lipsttat, and H. Manila. Borders: An efficient algorithm for association generation in dynamic databases. *Journal of Intelligent Information Systems*, 12(1):61–73, April 1999.

[8] A. Ayad, N. El-Makky, and Y. Taha. Incremental mining of constrained association rules. In *Proc. of First Intl. SIAM Conf. on Data Mining*, Chicago, April 2001.

[9] N. F. Ayan. Updating large itemsets with early pruning. Master's thesis, Bilkent University, Ankara, Turkey, July 1999.

[10] N. F. Ayan, A. U. Tansel, and E. Arkun. An efficient algorithm to update large itemsets with early pruning. In *Proc. of the Intl. Conf. on Knowledge Discovery in Data and Data Mining (SIGKDD'99)*, San Diago, CA, August 1999.

[11] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD'98)*, pages 85–93, Seattle, Washington, June 1998.

[12] R. J. Bayardo, R. Agrawal, and D. Gunopulos. Constraint-based rule mining in large, dense databases. In *Proc. of the 15th Intl. Conf. on Data Engineering (ICDE'99)*, pages 188–197, Sydney, Australia, March 1999.

[13] P. S. Bradley, U. M. Fayyad, and O. L. Mangasarian. Mathematical programming for data mining: Formulations and challanges. *INFORMS Journal on Computing*, 11:217–238, 1999.

[14] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD'97*, pages 255–264, Tucson, Arizona, May 1997.

[15] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. of ACM SIGMOD Conf. on Management of Data (SIGMOD'97)*, pages 255–264, New York, May 1997.

[16] D. W. Cheung, J. Han, V. T. Ng, A. W. Fu, and J. Fu. A fast distributed algorithm for mining association rules. In *Proc. of the Intl. Conf. on Parallel and Distributed Information Systems (PDIS'96)*, Miami Beach, Florida, December 1996.

[17] D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proc. of 12th Intl. Conf. on Data Engineering (ICDE'96)*, pages 106–114, New Orleans, Louisiana, February 1996.

[18] D. W. Cheung, S. D. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In *Proc. of the 5th Intl. Conf. on Database Systems for Advanced Applications (DASFAA'97)*, pages 185–194, Melbourne, Australia, April 1997.

[19] D. W. Cheung, V. T. Ng, and B. W. Tam. Incremental updates of discovered multi-level association rules. *Intl. Journal of Artificial Intelligence Tools*, 6(2):273–290, 1997.

[20] G. Grahne, L. Lakshmanan, and X. Wang. Efficient mining of constrained correlated sets. In *Proc. of Intl. Conf. Data Engineering (ICDE'00)*, pages 512–521, San Diago, CA, February 2000.

[21] J. Han and Y. Fu. Discovery of multi-level association rules from large databases. *IEEE Transactions on Knowledge and Data Engineering*, 11(5), 1999.

[22] J. Han, L. Lakshmanan, and R. T. Ng. Constraint-based multidimensional data mining. *IEEE COMPUTER (special issues on Data Mining)*, 32(8):46–50, 1999.

[23] J. Han, J. Pei, and Y. Yi. Mining frequent patterns without candidate generation. In *Proc. of ACM-SIGMOD Intl. Conf. on Management of Data (SIGMOD'00)*, pages 1–12, Dallas, TX, May 2000.

[24] C. Hidber. Online association rule mining. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD'99)*, pages 145–156, Philadelphia, PA, May-June 1999.

[25] R. J. Hilderman and H. J. Hamilton. Knowledge discovery and interestingness measures: A survey. Technical report, University of Regina, Saskatchewan, Canada, 1999.

[26] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for association rule mining - a general survey and comparison. In *SIGKDD Explorations*, pages 58–64, July 2000.

[27] M. Houtsma and A. Swami. Set-oriented mining of association rules in relational databases. In *Proc. of 11th Intl. Conf. on Data Engineering (ICDE'95)*, Taipei, Taiwan, March 1995.

[28] M. Kamber, J. Han, and J. Y. Chiang. Using data cubes for metarule-guided mining of multi-dimensional association rules. Technical report, School of Computing Science, Simon Fraser University, May 1997.

[29] L. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *Proc. of ACM-SIGMOD Intl. Conf. Management of Data(SIGMOD'99)*, pages 157–168, Philadelphia, PA, June 1999.

[30] S. D. Lee and D. W. Cheung. Maintenance of discovered association rules: When to update? In *Proc. of ACM SIGMOD Workshop on Data Mining and Knowledge Discovery (DMKD-97)*, Tucson, Arizona, May 1997.

[31] D. Lin and Z. M. Kedem. Pincer-search: A new algorithm for discovering the maximum frequent set. Technical Report TR1997-742, NYU, September 1997.

[32] J. L. Lin and M. H. Dunham. Mining association rules: Anti-skew algorithms. In *Proc. of Intl. Conf. on Data Enginering (ICDE'98)*, Orlando, Florida, February 1998.

[33] R. T. Ng, L. Lakshmanan, J. Han, and T. Mah. Exploratory mining via constrained frequent set queries. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD'99)*, pages 556–558, Philadelphia, PA, June 1999.

[34] R. T. Ng, L. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD'98)*, pages 13–24, Seattle, Washington, June 1998.

[35] E. Omiecinski and A. Savasere. Efficient mining of association rules in large dynamic databases. In *Proc. of 16th British National Conference on Databases (BNCOD'98)*, pages 49–63, Cardiff, Wales, UK, July 1998.

[36] J. S. Park, M. S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. In *Proc. of ACM SIGMOD Conference on Management of Data (SIGMOD'95)*, pages 175–186, San Jose, CA, May 1995.

[37] J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In *Proc. of Intl. Conf. Knowledge Discovery and Data Mining (KDD'00)*, pages 350–354, Boston, MA, August 2000.

[38] J. Pei, J. Han, and L. Lakshmanan. Mining frequent itemsets with convertible constraints. In *Proc. of Intl. Conf. on Data Engineering (ICDE'01)*, Heidelberg, Germany, April 2001.

[39] V. Pudi and J. Haritsa. Quantifying the utility of the past in mining large databases. Technical Report TR-2000-01, DSL, Indian Institute of Science, 2000.

[40] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD'98)*, pages 13–24, Seattle, Washington, June 1998.

[41] N. L. Sarda and N. V. Srinivas. An adaptive algorithm for incremental mining of association rules. In *Proc. of 9th Intl. Conf. on Database and Expert Systems Applications Workshop (DEXA Workshop'98)*, pages 240–245, Vienna, Austria, August 1998.

[42] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. of 21th Intl. Conf. on Very Large Databases (VLDB'95)*, pages 432–444, Zurich, Switzerland, September 1995.

[43] A. Savasere, E. Omiecinski, and S. B. Navathe. Mining for strong negative associations in a large database for customer transactions. In *Proc. of Intl. Conf. on Data Engineering (ICDE'98)*, 1998.

[44] P. Shenoy, G. Bhalotia, J. R. Haritsa, M. Bawa, S. Sudarshan, and D. Shah. Turbo-charging vertical mining of large databases. In *Proc. of ACM-SIGMOD Intl. Conf. on Management of Data (SIGMOD'00)*, pages 22–33, Dallas, TX, May 2000.

[45] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. of 21st Intl. Conf. on Very Large Databases (VLDB'95)*, pages 407–419, Zurich, Switzerland, September 1995.

[46] R. Srikant and R. Agrawal. Mining sequential patterns. In *Proc. of 11th Intl. Conf. on Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, March 1995.

[47] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD'96)*, pages 1–12, Montreal, Canada, June 1996.

[48] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery in Databases and Data Mining (KDD'97)*, pages 67–73, Newport Beach, CA, August 1997.

[49] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An efficient algorithm for the incremental updation of association rules in large databases. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, pages 263–266, Newport Beach, CA, August 1997.

[50] S. Thomas and S. Chakravarthy. Incremental mining of constrained associations. Technical Report TR1998-018, University of Florida, 1998.

[51] S. Thomas and S. Sarawagi. Mining generalized association rules and sequential patterns using sql queries. In *Proc. of the 4th International conference on Knowledge Discovery and Data Mining (KDD'98)*, New York, August 1998.

[52] H. Toivonen. Sampling large databases for association rules. In *Proc. of the 22nd Conf. on Very Large Databases (VLDB'96)*, Bombayi India, September 1996.

[53] D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: a generalization of association rule mining. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD'98)*, pages 1–12, Seattle, Washington, June 1998.

[54] A. K. H. Tung, H. Lu, J. Han, and L. Feng. Breaking the barrier fo transctions: Mining inter-transaction association rules. In *Proc. 1999 Intl. Conf. on Knowledge Discovery and Data Mining (KDD'99)*, San Diago, CA, August 1999.

[55] K. Wang, Y. He, and J. Han. Mining frequent itemsets using support constraints. In *Proc. of Intl. Conf. on Very Large Data Bases (VLDB'00)*, Cairo, Egypt, September 2000.

[56] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, pages 283–286, Newport Beach, CA, August 1997.

[57] M. J. Zaki, S. Parthasarathy, Mitsunori Ogihara, and Wei Li. Parallel algorithms for discovery of association rules. *Data Mining and Knowledge Discovery*, 1(4):343–373, 1997.

[58] M. J. Zaki and S. Parthasarthy. Evaluation of sampling for data mining of association rules. In *IEEE Int'l Workshop on research Issues in Data Engineering*, 1997.