Efficient Data Mining of Constrained Association Rules

by

Chiu Yan (Alex) Pang

Ph.D. (Physics), North Carolina State University, 1995

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

we accept this thesis as conforming
to the required standard

The University of British Columbia

August 1998

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science
The University of British Columbia
Vancouver, Canada
V6T 1Z4

Date: September 3, 1998

Abstract

With the recent advances in information technology, companies are now collecting more and more data related to their business. Companies are very interested in decision support systems that can discover knowledge from data and help them gain insight into their data. Data mining with the goal of discovering non-trivial information or patterns hidden in large databases has, therefore, recently become one of the most active research areas in database technology.

Association rules relate items which tend to occur together in a given event or record. Mining association rules represents one of the most important problems in data mining. However, the current framework suffers seriously from the lack of user interaction and focus. In this thesis, we propose a new paradigm called *Constrained Association Rules* where (i) the mining of the rules is divided into two phases with various breakpoints for user feedback, and (ii) users can associate constraints with their queries. We analyze many SQL-style constraints and introduce the notions of *succinctness* and *anti-monotonicity* for their classification.

We design a new algorithm called CAP for mining association rules that satisfy a set of given constraints. The idea is to check for satisfaction of the constraints as early as possible by exploiting the properties of anti-monotonicity and succinctness of the constraints. Several optimization techniques are developed. Our experimental evaluation indicates that CAP runs much faster and can sometimes outrun several basic algorithms by as much as 80 times.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor Dr. Raymond Ng for his invaluable guidance and financial support. He is nice and brilliant. Without him, this work could not have been completed. Special acknowledgments also go to Dr. Laks Lakshmanan and Dr. Jiawei Han who are the coauthors of the paper [NLHP98] on which this thesis is based.

I would also like to thank my second reader Dr. George Tsiknis for reading this thesis and for his invaluable suggestions.

Finally, I would like to thank all of my friends and other people in the Computer Science department at UBC for having created a supportive and friendly environment which has made my study here very memorable.

CHIU YAN (ALEX) PANG

The University of British Columbia

August 1998

To my family

Chapter 1

Introduction

1.1 Data Mining

What is data mining ?

Over the past 15 to 20 years, computers have been used to capture detailed transaction information in a variety of corporate enterprises. Some of the examples of transaction-intensive industries are retail sales, banking, telecommunications, and credit card operations. With the availability of powerful and affordable computer systems, many corporations have created huge repositories of data related to their business. These collected data represent an invaluable source of information because the data implicitly contain knowledge about the behavior of the customers. The knowledge in turn can play a crucial role in the corporation's survival in today's competitive market. Consequently, new techniques and tools which can intelligently and automatically transform the processed data into useful information and knowledge has become highly relevant.

Data mining, which is also referred to as *knowledge discovery from databases*, can be defined as an automatic process of discovering non-trivial, previously un-

known, and potentially useful information from very large datasets. In the past few years, it has not only become one of the most active research areas in databases, but has also attracted increasing industrial attention. A leading industrial analyst [Meta Group] has projected the data mining market to grow from $3.3 billion in 1996 to $8.4 billion by the year 2000. There are many applications of data mining. For example, discovered knowledge from data mining has been found to be relevant in direct marketing, decision making, fraud identification, and process control.

Data mining models

Discovered knowledge means high-level information such as regularities, rules, patterns, trends, etc. Many interesting models for discovered knowledge have been identified by researchers. These include *associations* [AS94], *sequential patterns* [AS95], *classification* [MAR96], *clustering* [NH94], *outliers* [KN97], and *temporal patterns* [ALSS95].

Association models focus on items that occur together in a given event or record. A typical rule discovered in an association based data mining takes the form: If item X is part of an event, then c% of the time (the so-called confidence factor), item Y is also part of the event. A famous example is the rule "98% of the people who buy diapers also buy beer ". Another example is "85% of customers that purchase tires and auto accessories also get automotive services ". Association rule analysis has gained increasing interest with the widespread use of checkout scanners, which let retailers gather transaction details. This is why market-basket analysis has been the most well-known application of association rule analysis. This whole thesis is, in fact, on the problem of finding association relationships. A more precise definition of the model will be given in the following section.

Sequential patterns are similar to association models, except that the relationships among items are spread over time [AS95]. In fact, association mining

techniques can be applied to mining sequential patterns by treating sequences as associations in which the events are linked by time. The motivation behind sequential patterns is the elapsed time between transactions or the duration of an event in an association can be crucial.

In classification [MAR96], it is assumed that the value of a categorical variable can be assigned to any cases. The categorical variable is used as a classification label. A number of cases for which the classification label is known are employed as training examples. A classification algorithm then attempts to find a predictive pattern that can classify other cases.

Clustering models segment a database into different groups whose members are very similar [NH94]. However, unlike classification, one does not know *a priori* what the clusters will be, or on what attributes the data will be clustered. Consequently, the resulting clusters could reveal previously unknown facts about the data. Interesting applications of both classification and clustering are in direct marketing. For example, consider a company doing promotion with a mailing list. If the company can classify the customers on the list into categories such as "likely response" or "unlikely response" based on historical patterns, the company can then tailor its marketing plan and target only the customers that are more likely to respond. This can significantly reduce the cost and increase the rate of return of the promotion.

Another pattern that is gaining increasing popularity is outlier detection [KN97]. While most of the models mentioned above deal with the trends of the majority, outlier detection attempts to identify the exceptional cases (*i.e.* the minority). For example, in the credit card business, an unusual spending pattern within a short period of time often indicates stolen card usage. Other applications include fraud detection in production processes.

Finally, time is often an important attribute of a dataset. A significant example is in financial time series [ALSS95]. It is certainly a man's dream come

true if he can predict the movement of a stock price. Given a time series database, the problem of finding all the time series which have similar behavior is therefore of great interest. Similarity and clustering are some of the typical questions asked in data mining of temporal patterns.

1.2 Association Rules

Association relationships or association rules represent one of the most popular patterns pursued in data mining processes. The basic idea of an association rule is to capture the sets of items that tend to appear together. An example is the rule "80% of the people who buy butter also buy milk". There can be many applications of the discovered rules. To use the rule in the above example, the manager in a supermarket will make sure that butter and milk will not be on sale at the same time.

The idea of association rules is introduced by Agrawal, Imielinski and Swami in the early 90's [AIS93] as part of the IBM's data mining research program [QUEST]. The motivation is to understand customer behavior by examining transactional databases. Each transaction in the database contains items that are purchased together. By counting the frequency of the items appearing in the database, one can identify the sets of items that tend to be purchased together.

To be more precise, an association rule is defined as an implication of the form $X \Rightarrow Y$, where X and Y are sets of items and the set $X \cup Y$ appears frequently enough in the transactional database in the sense that at least $s\%$ of the transactions contain $X \cup Y$, and at least $c\%$ of the transactions that contain X also contain Y. Here, s and c are called the *support threshold* and the *confidence* respectively.

In an association query, a user simply supplies specific values for the support threshold and the confidence as input. A mining engine then finds *all* the itemsets

X, Y such that $X \Rightarrow Y$ satisfies the above definition of an association rule. The output of the query is a list of all such association rules found. Without prior knowledge of the data, a user usually has no idea what the appropriate values for the support threshold or the confidence should be. Runs with supplied values might discover no rules or thousands of rules. This is one of the problems with the above framework of association rules.

Since its introduction [AIS93], the problem of mining association rules has been the subject of numerous studies. Issues discussed include extending the association rules framework [HF95, SA95, SA96, FMMT96, MY97, TUACMN97, BMS97, SVA97], applying the association framework to solve other problems [AS95, AMS97], improving the efficiency of the mining algorithms [PCY95, BMUT97, CHNW96], and parallel implementation of the Apriori Algorithm [AS94, HKK97]. In particular, Agrawal and Srikant in 1994 [AS94] proposed a very efficient algorithm called *Apriori* for mining association rules. We will explain in detail the Apriori Algorithm and review some of the more important related works in the following chapter.

While all of the above mentioned work has enriched the field of association mining, none of them address adequately the question of the *interestingness* of the discovered rules. There may be thousands of rules found, but only a small portion of them are interesting to the user. Only Srikant *et al* [SVA97] discuss the issue of putting constraints on the frequent sets, which is one of the suggestions of this thesis. However, Srikant *et al* [SVA97] only consider membership constraints in the context of an item taxonomy, which correspond to only a small subclass of the categories of constraints considered in this thesis.

1.3 Motivation of the Thesis

While the notion of association rules with the corresponding Apriori Algorithm discussed in the previous section represents a significant development in data mining, it suffers seriously from the following problems.

Problem 1 – Lack of User Exploration and Control: In a classical association query, a user supplies the support threshold and the confidence. Then, *Apriori* or a similar algorithm returns with all the association rules it finds. The whole mining process behaves like a black-box where the user has very limited control of the process except for supplying the two threshold values. But the supplied threshold values might not be appropriate. It is more desirable if the mining process can support user *exploration* on the data. However, even with the development of many efficient algorithms, state-of-the-art association mining nowaday still requires hours to complete. Therefore, without much control over the mining process and the relatively long turnaround time, the classical model of mining association rules cannot support efficient user exploration. Furthermore, the results returned by the black-box may not contain what the user is looking for.

Problem 2 – Lack of Focus: The second problem is related to the fact that the ultimate goal of a data mining user is to support his or her business decision making. The user probably has some specific questions he or she would like to answer and may therefore be much more interested in only certain types of association patterns. For example, a user may want to find associations between sets of items whose origins are domestic, or associations from itemsets of male products to itemsets of female products. However, the classical model does not support any of these expressions of focus or preference.

Problem 3 – Rigid Notion of Relationship: The third problem is that the classical notion of association relationship is too rigid. Two sets of items are "as-

sociated" only if they appear together frequently enough and the rule confidence exceeds the confidence threshold. While such a notion of associations is useful, there can be other types of association that are relevant as well. For example, correlation is often used in statistics. Brin, Motwani, and Silverstein [BMS97] argue that in many circumstances correlation can be more useful than confidence. For example, the rule "past active duty in military $\Rightarrow$ no service in Vietnam " has a very high confidence of 0.9 in census data. Yet it is quite misleading because having past military service only increases the chances of having served in Vietnam. Furthermore, sometimes it may make more sense to have *different* support threshold for the antecedent sets and consequent sets, especially when they are from different domains. The rule *pepsi* $\Rightarrow$ *snacks* is an example of associations from sets of *items* to sets of *types*. In such situation, the appropriate support threshold values for *pepsi* and *snacks* (*i.e.* any item of type *snacks*) can be very different.

These shortcomings in the current framework of the classic Association Rules form the motivation of our work. To overcome these problems, we suggest several principles which will be given in the following section.

1.4 Contributions of our work

We summarize our contributions in this section. First of all, we suggest several principles of association mining which can be used to address the problems given in the last section. The principles suggested are:

1. The mining process should not behave like a black-box with one-time user supplied input parameters and final results at the end. Instead, there should be breakpoints in the process for accepting user feedback.

2. With the user feedback mechanism, a user should not only be able to guide and control the mining process, but also have the chance to approve any task

that involves a substantial cost.

3. The mining system should have a mechanism to allow a user to express his or her focus or preferences.

4. With the user preferences specified, the system should have the ability to adjust its mining algorithms so that it performs only the necessary computation and nothing more.

5. The system should have a mechanism to allow a user to choose different significance metrics and criteria to be used in defining an association relationship.

To realize these principles, we have designed a two phase architecture (Figure 3.1) for exploratory association mining. We will discuss the architecture in more detail in section 3.2. Here, we emphasize that the architecture is new, but downward compatible in the sense that if a user wants only classical associations and the classical mode of interaction, he or she can do so by setting all the parameters at the beginning and turning off all the breakpoints. Moreover, it is consistent with the above five principles, and is powerful in providing human-centered exploration.

The second major part of our contributions is the introduction of the notion of *constrained association queries* (CAQ) to be discussed in Chapter 3. Along with the proposed architecture, CAQ provides a rich interface for the user to express focus and to control the mining process to best suit the particular interests of the user. In addition, we design and develop an efficient algorithm which we call CAP (Constrained Apriori) to make use of the additional pruning power provided by the constraints. To answer a constrained association query, one can imagine a few algorithms, for example, running *Apriori* followed by constraints checking at the end. We have compared CAP with a few of these simpler algorithms. Our experimental results indicate that CAP can sometimes outperform other algorithms by as much as 80 times !

In our investigation, we discover that one can classify all the constraints according to two properties, namely, *succinctness* and *anti-monotonicity*. These properties turn out to be extremely important in optimizing the mining algorithm. While the anti-monotonicity property may be known in literature (in fact *Apriori* is based on the anti-monotonicity of the frequency constraint), the classification of constraints based on succinctness and anti-monotonicity is fundamentally new and is one of our major contributions.

1.5 Outline of the Thesis

The thesis is organized as follows. In the next chapter, we describe the formulation of the problem of mining association rules and present the Apriori Algorithm. We also review in detail some of the important related works. In Chapter 3, we present constrained association queries as a new paradigm in mining association rules. We then discuss some of the basic algorithms that one can think of for these new type of queries in Chapter 4. Chapter 5 describes an optimized algorithm which we refer to as CAP. Chapters 4 and 5 are the main focus of this thesis. Performance of various algorithms are then compared in Chapter 6. Chapter 7 presents yet another optimization technique that can be used not only in CAP, but also in Apriori. Finally, we draw our conclusions in Chapter 8 along with a discussion on some possible directions for future work.

Chapter 2

Background: Association Rules

2.1 Formulation

In this section, we present a mathematical formulation of the association rule problem. We also establish our notation used for the rest of the thesis.

The starting point is to assume that there is a finite (but very large) set of items which we denote by `Item` . The strict power set of `Item` is denoted by $\mathcal{P}(\texttt{Item})$ or $2^{\texttt{Item}}$. A transaction T is simply any subset of $2^{\texttt{Item}}$. A transaction database, denoted by $\mathcal{DB}$, is a set of transactions.

Definition 2.1 Given any set $S \subseteq \mathcal{P}(\texttt{Item})$ and a transaction database, the *support* or the *frequency* of S is the number of transactions containing S.

We often refer to S as an itemset.

Definition 2.2 An itemset S is said to be *large* or *frequent* if the support of S is larger than a given threshold value.

Whether an itemset is large or not depend on the threshold value. We refer to this

threshold value as the *support threshold.* Moreover, we often refer to itemsets of size k as *k-itemsets.* We also denote the set of all large k-itemsets by L_k. We are now ready to define an association rule.

Definition 2.3 Given a support threshold s and another input parameter c referred to as *confidence*, an *association rule* is a rule of the form $X \Rightarrow Y$ if $X \cup Y$ is a large itemset, and among all the transactions that contain X, at least c % of them also contain Y.

X and Y are usually referred to as the antecedent and the consequent set respectively. The problem of association rules is the problem of finding *all* the possible association rules given a support threshold and a confidence.

The motivation of association mining and some of the examples are given in Section 1.2. Here, we note that finding association rules can be divided into two phases. The first one is to find all the large itemsets, *i.e.* potential candidates for $X \cup Y$. The second phase then constructs all possible association rules of the form $X \Rightarrow Y$ from each large item set $X \cup Y$ by checking whether the support ratio of $X \cup Y$ over X exceeds the confidence threshold. For example, if the set $\{A, B, C\}$ is found to be large, then, in the second phase, we check whether each one of the rules $\{A, B\} \Rightarrow \{C\}$, $\{A, C\} \Rightarrow \{B\}$, $\{B, C\} \Rightarrow \{A\}$, $\{A\} \Rightarrow \{B, C\}$, $\{B\} \Rightarrow \{A, C\}$, and $\{C\} \Rightarrow \{A, B\}$ satisfies the confidence requirement. Many previous studies have shown [AIS93, AS94] that the first phase is the bottleneck of the computation. Therefore, most of the literature on mining association rules focuses on the first phase of finding all frequent itemsets. In the following discussion, our main concern is also on the problem of finding large itemsets.

Association mining, in principle, requires consideration of all possible combination of the items. Figure 2.1 shows all the possible itemsets that can be formed from five items arranged in a lattice. The lines in Figure 2.1 connect a set to all its

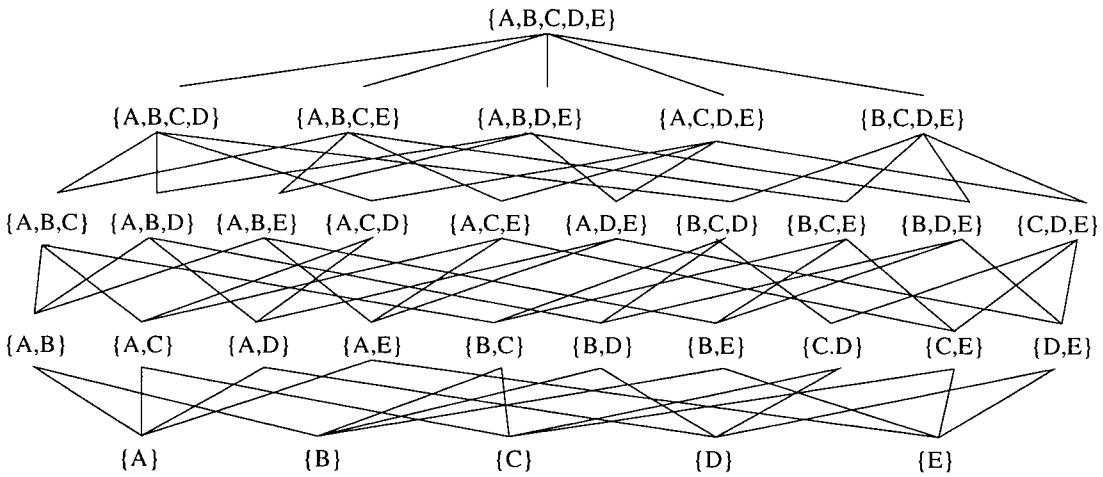

Figure 2.1: The lattice space formed from five items {A,B,C,D,E}

supersets and subsets. The number of sets in this *lattice space* increases exponentially with the number of items. For practical applications, the number of items is at least of the order of several hundreds. Therefore, association mining is a challenging problem and any naive or brute-force approach will certainly fail. To facilitate our discussion, we hereafter refer to the association rules discussed in this section as the *classical association rules* to distinguish from other extensions.

2.2 Apriori Algorithm

Agrawal and Srikant [AS94] proposed an efficient algorithm called *Apriori* listed below for finding all the large itemsets.

Algorithm 2.1 (Apriori)

1 C_1 consists of all itemsets of size 1; $k = 1$; $Ans = \emptyset$;

2 while (C_k not empty) {

2.1 conduct database scan to form L_k from C_k;

2.2 form C_{k+1} from L_k based on C_{freq}; $k++;\}$

3 for each set S in some L_k:

add S to Ans.

Algorithm *Apriori* follows a level-wise generate-and-test framework. In Step 1, it first generates a list of size one candidate sets, denoted by C_1. Then it counts the support of each candidate in the list to find all the size one large itemsets, L_1. From L_1, the algorithm constructs a list of size two candidate sets, C_2. The process is then repeated until there is no more candidates, *i.e.* until C_k is empty (see Step 2). Finally, each large itemset is added to the Ans, the output of the algorithm. The performance of such generate-and-test algorithms is usually very poor because the number of candidates increases exponentially with the number of items. The contribution of Agrawal and Srikant [AS94] is that when they generate C_{k+1} from L_k in Step 2, they make use of an important property of the support of an itemset: the support of an itemset cannot be larger than the support of any of its subset. To prove such a property, one just needs to realize that if a transaction supports (*i.e.* contains) an itemset S, it supports every subset of S. Thus, a size $k+1$ itemset cannot be a large itemset (*i.e.* cannot be in C_{k+1}) unless *all* its size k subsets are large (*i.e.* in L_k). Therefore, Agrawal and Srikant can *a apriori* drop a size $k+1$ candidate set if one of its size k subsets are not in L_k. This greatly reduces the number of size $k+1$ candidate sets, which in turn further reduces the number of size $k+2$ candidate sets, etc. Thus their Apriori Algorithm can be very efficient even for large numbers of items and for large databases.

Example 2.1 We illustrate the Apriori Algorithm in Figure 2.2 with a simple example. We assume that there is only five items A, B, C, D, E and the support threshold is 10. Each set in $C_{k+1}(k = 1, 2)$ is constructed by taking the union of two sets in L_k that differ by only one item. The algorithm then checks for all the

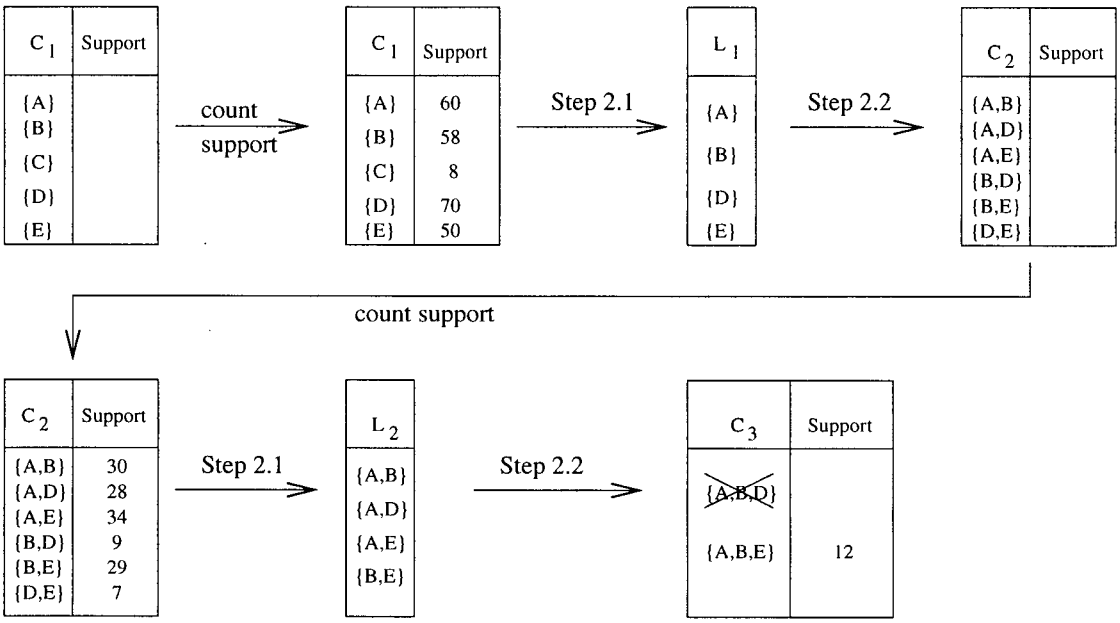

Figure 2.2: An example illustrating Apriori Algorithm with five items

size k subsets. If any one of the subsets is not in L_k, the set is pruned away from C_{k+1}. In our example, the set $\{A, B, D\}$ is pruned away from C_3 because the set $\{B, D\}$ is not in L_2. On the other hand, $\{A, B, E\}$ is not because all the size two subsets, $\{A, B\}, \{A, E\}$, and $\{B, E\}$, are in L_2.

2.3 Related Works

As we mentioned in Chapter 1, there has been numerous studies on mining association rules. In this section, we first review some of the works that are relatively more important in the development of association mining. These works roughly fall into three categories. The first one aims at generalization of the notion of classical association rules. In other words, the focus is on improving the *effectiveness* of the association rules. The goal of the second category is more on improving the *effi-*

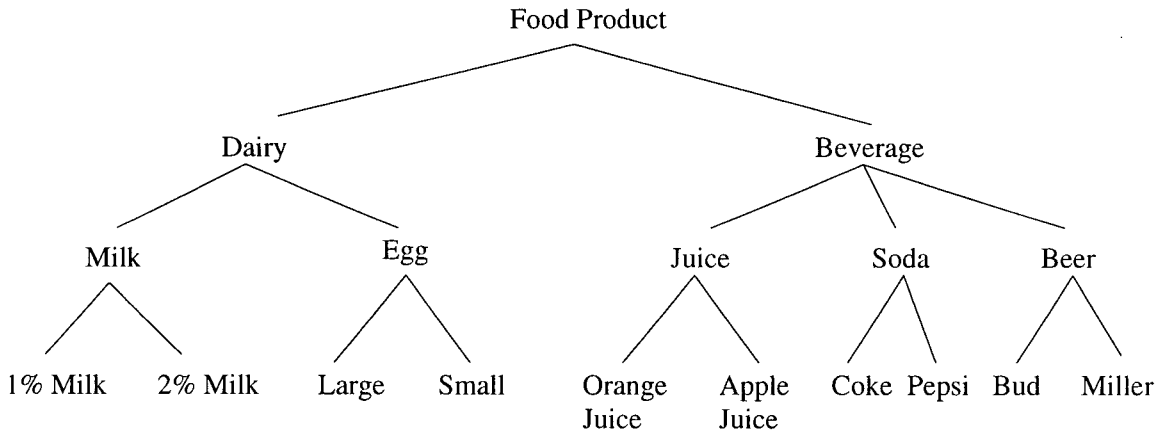

Figure 2.3: An example of a taxonomy (concept hierarchy)

ciency of the mining algorithms. The third category simply covers all the works that do not fall into the first two categories. An example of the third category would be solving other problems with the association concept.

Generalization of Association Framework

Among the first attempt to generalize association rules is the work by Han and Fu [HF95]. Their idea is that while an association rule such as "80% of customers that purchase milk may also purchase beer " is interesting, it could be more informative to also show that "75% of people buy Budweiser if they buy 2% milk ". The association relationship in the latter statement is expressed at a lower concept level where more specific and concrete information is provided. Conversely, sometimes it may be more desirable to have associations at a higher concept level. An example is that the rule "90% of people who live on 123 East 2nd Street fly at least once a year" is obviously less interesting to a travel agent than the rule "70% of people living in Vancouver fly at least once a year". Han and Fu suggested that association mining should be done at multiple concept levels. They call the different concept levels a concept hierarchy. Srikant and Agrawal [SA95] referred to concept hierarchies

Salary	Equi-depth Interval	Distance-based Interval
18K	18K,30K	18K, 18K
30K		30K,31K
31K	31K,80K	
80K		80K,82K
81K	81K, 82K	
82K		

Table 2.1: Equi-depth vs. distance-based partitioning

as taxonomies and had studied essentially the same problem as Han and Fu. A taxonomy (or concept hierarchy) is a *is-a* hierarchy on a set of items with a more general description of the items at the higher levels of the hierarchy. An example of a taxonomy (*i.e.* a concept hierarchy) is shown in Figure 2.3. This taxonomy says that 1% milk *is-a* milk, 2% milk *is-a* milk, milk *is-a* Dairy, Dairy *is-a* Food Product, etc. The problem of multiple-level association rules is to find association rules that span different levels of the taxonomy. Both Han & Fu and Srikant & Agrawal have designed efficient algorithms that can generate association rules at multiple levels.

Another group of studies is on extending association rules to more general type of attributes. So far, association mining had been focused on categorical attributes. Srikant and Agrawal in 1996 [SA96] introduced the problem of mining association rules with quantitative attributes. An example of an association rule containing both quantitative and categorical attributes might be "10% of married people between age 50 and 60 have at least 2 cars". They deal with quantitative attributes by partitioning the values of the attribute and then combining adjacent partitions as necessary. In other words, they converted quantitative attributes to categorical attributes by creating finite number of partitions for the values of the numeric attributes. An important question is what should be the number of intervals and what the criteria in constructing a "good" interval should be. Srikant and Agrawal [SA96] used an equi-depth method where the intervals are determined by

their relative ordering and their support. Here the depth of a partition means the support of the partition. For a depth d, the first d values (in order) are placed in one interval, the next d in a second interval, etc. However, Miller and Yang [MY97] pointed out that equi-depth partitioning may not work well for skewed data, and may not be semantically suitable under certain circumstances. Instead, they proposed a distance-based partitioning where the distance between items are taken into account. As an example, Table 2.3 shows the different partitions obtained by the above two methods on a Salary attribute. Equi-depth partitioning results in three partitions, 18K,30K, 31K,80K, and 81K,82K, while distance-based method gives 18K,18K, 30K,31K, and 80K,82K. Miller and Yang [MY97] argued that distance-based partitions are more consistent with our intuitive understanding of the data and intervals that include close data values (such as 81K,82K) are more meaningful than intervals involving distant values (such as 31K, 80K). In our example, a rule involving the interval 31K, 80K will be of less interest than rules involving the interval 30K,31K. Finally, Fukuda *et al* [FMMT96] considered various performance issues in mining association rules with numeric attributes and designed fast algorithms that borrow techniques from computational geometry.

Other studies that extend the classical association relationship include the work by Brin, Motwani, and Silverstein [BMS97] where the authors suggested correlation as a significant metric of an association, instead of the confidence. Their motivation is to allow associations to generalize beyond market baskets data. This is because under more general settings (*i.e.* , non basket data such as census data), the classical association rules is only one of the many types of recurring patterns that could or should be identified as "associations". Correlation is among the most useful information regarding two variables.

So far, all of the mentioned works assume that the database is fixed and the focus is on mining different types of association rules. However, in practical application of database mining, another problem we need to address is how to update,

maintain and manage the rules discovered. Whenever the database is updated, new association rules may be introduced while some existing ones may be invalidated. Therefore, efficient maintenance of discovered association rules is a non-trivial problem. Cheung *et al* [CHNW96] study this problem and have developed an incremental updating technique.

Improving Efficiency of Mining Algorithms

Apart from extending the association rule framework, many researchers aim to improve the efficiency of the mining algorithm. Park, Chen, and Yu [PCY95] proposed another approach totally different from the Apriori. Their algorithm is referred to as DHP (Direct Hashing and Pruning). Their observation is that the initial candidate set generation, especially for the large size two itemsets, dominates the total execution cost. This can be explained by the reason that unless the support threshold is very selective, L_1 is usually very large, which in turn results in a huge number of candidate sets in C_2 (as $|C_2| = \begin{pmatrix} |L_1| \\ 2 \end{pmatrix}$). The step of determining L_2 from C_2 by scanning the whole database and testing each transaction against C_2 is hence very expensive. The idea of the DHP Algorithm is to generate a much smaller C_2 by using a hashing technique to filter out unnecessary itemsets. When the support of candidate k-itemsets is counted by scanning the database, DHP accumulates information about candidate $(k+1)$-itemsets in advance in such a way that all possible $(k+1)$-itemsets of each transaction are hashed to a hash table. This also allows DHP to trim progressively the size of the transaction database which can reduce the processing time in later iterations. Their experimental evaluation shows that DHP is about four times faster than Apriori.

A major bottleneck in computing frequent sets is in counting the support of candidate sets. At each level k, a database scan is required to find the support of the sets in C_k. Because of the large database size, the I/O cost for each scan can

be huge. Therefore, it will be beneficial if one can reduce the number of database scan. Brin *et al* [BMUT97] proposed a dynamic itemsets counting algorithm (DIC) to reduce the number of database scan. The idea is to start counting $(k+1)$-itemsets before finishing counting k-itemsets. In one of their examples, Brin *et al* show that the number of database scan required by DIC is 1.5 passes while that of Apriori is 3. However, the implementation of the DIC Algorithm requires keeping track of many itemsets (in memory). As the number of all possible itemsets increases exponentially with the number of items, this requirement may pose serious limitations on the algorithm.

To speed up the mining process, another major approach is to use parallel algorithms. Both Agrawal and Shafer [AS96] and Han, Karypis, and Kumar [HKK97] have designed parallel algorithms for mining association rules and have studied various performance issues. Their idea is that the Apriori Algorithm can be divided into several sub-problems such as counting support, candidate generation, and rule generation. While these sub-problems depend on each other, the algorithms that solve these sub-problems may be executed in parallel. For example, rule generation from L_k can be executed independently from counting support of $C_{k'}$ for $k' > k$.

Interesting applications of the association mining framework include the work by Agrawal and Srikant [AS95], where association mining is applied for mining sequential patterns, and the investigation by Ali, Manganaris, and Srikant [AMS97] where it is used for partial classification.

Summary

While all of the above works represent important extensions, improvements and/or applications of the classical association rules [AS94], none of them has solved the problems presented in Section 1.3. In particular, no current framework addresses adequately the question of "interestingness" of the discovered rules. Different users

have different interests and different needs. Support threshold and confidence are two very rough and generic quality (or "interestingness") measures. They may not capture what a user wants. Moreover, they may lead to thousands of rules or no rules at all. As far as we are aware of, no one has talked about putting constraints on the frequent sets to capture the "interestingness" of rules. The only exception is the recent work by Srikant, Vu, and Agrawal [SVA97] which considers only membership constraints on an item taxonomy. Nevertheless, their constraints represent only a small subclass of the types of constraints we study in this thesis, which include domain and SQL-style aggregation constraints. Furthermore, our focus is on the analysis of the pruning properties of the constraints and our Algorithm CAP can handle a much broader class of constraints with significant pruning power than the algorithms given in [SVA97].

Meo, Psaila, and Ceri [MPC96] consider a language for mining association rules with conditions. However, they do not consider pruning optimizations provided by the conditions.

Tsur *et al* [TUACMN97] attempt to generalize association queries to parameterized queries with filters (conditions applied to the result of a query), which they call "query flocks". However, the filter is confined to lower bound constraints on the number of tuples returned by a query. Moreover, they do not have the general notion of constraints nor have they classified the constraints or studied the role of the constraints in optimization.

Chapter 3

Constrained Association Queries

In this chapter, we introduce the notion of *Constrained Association Queries* (CAQ) to address the several problems of the classical association queries discussed in the Chapter 1. Our goal is to let a user specify constraints expressible in SQL-like languages and to enable the user to efficiently carry out *exploratory* and *ad hoc* association data mining activities. Our design is based on the five principles presented in section 1.4

3.1 Definitions

Definition 3.1 A *constrained association query* (CAQ) is a query of the form $\{(S_1, S_2)|C\}$, where S_1, S_2 are set variables and C is a conjunction of a set of constraints on S_1, S_2.

The above definition is fairly general. In particular, we do not have the notion of antecedent and consequent set in the definition, although typically $S_1(S_2)$ represents the antecedent (consequent respectively) set in an association relationship. The reason is that for this generalization such identification is not required in

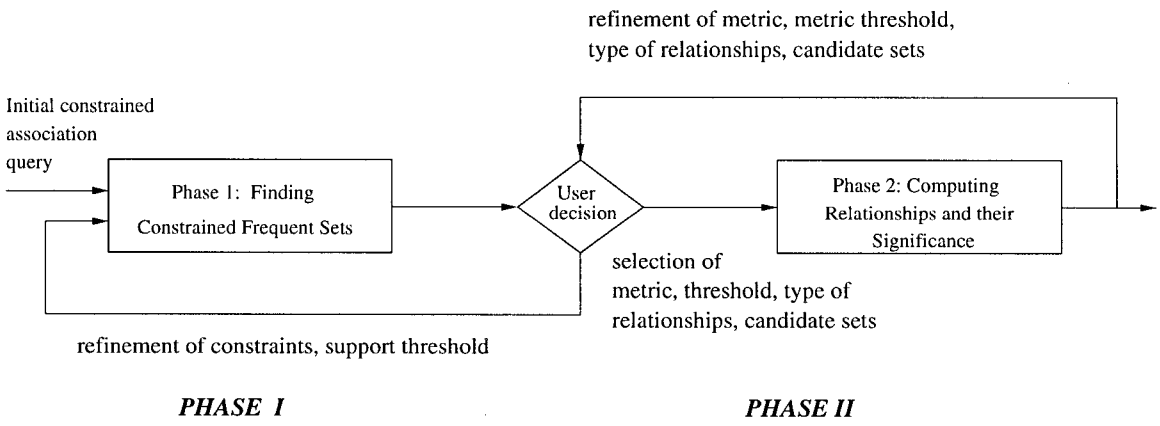

Figure 3.1: An Architecture for Exploratory Association Mining

finding all large itemsets.

The basic requirement of the support of S_i being larger than some given threshold can be thought of as a constraint. We embed this *frequency constraint*, written as $freq(S_i)$, in the constraint $\mathcal{C}$. Before we elaborate on the above definition, we first present an architecture for processing CAQs.

3.2 Architecture

Our proposed architecture, which is shown in Figure 3.1, is divided into two phases. In phase I, the system processes constrained association queries and outputs all the large itemsets satisfying the set of constraints $\mathcal{C}$ specified in the queries. As mentioned in the last section, each constraint in $\mathcal{C}$ may be applicable to the antecedent, or the consequent, or both and $\mathcal{C}$ also includes the frequency constraint. Upon seeing the output of the query, which is in the form of a list of pairs of candidates (S_a, S_c), for the antecedent and consequent satisfying $\mathcal{C}$, the user can either modify his or her query by adding, deleting, or refining the constraints etc, or change the support

thresholds. Then, the system accepts and processes the refined query. The process is repeated as many times as the user desires.

When the user is satisfied with the list of large itemsets found, the user can instruct the system to proceed to Phase II. The main function of the system in Phase II is to let the user specify the significance metric, the corresponding metric threshold, and any other further conditions that may be imposed on the antecedent and consequent. For example, as in classical association queries, a user could choose the confidence as the significance metric, specify the confidence threshold and require that (S_a, S_c) be frequent. Similar to Phase I, Phase II is also iterative. Upon seeing the final association relationship, the user can modify his or her specification of the metric, and/or thresholds values etc. The system will then process the refined query starting from the beginning of Phase II or even Phase I if necessary.

One important consequence of allowing user feedback at various breakpoints in the architecture is that the user can have the control and final approval in authorizing costly operations. For instance, one possible significance metric is correlation between the antecedent and consequent [BMS97]. Correlation is an expensive computation. It will be a waste of CPU time if the user can find out only at the end that the final answers of the query involve sets of items that are not of interest to the user.

Another important feature of the proposed architecture is that it is downward compatible. In other words, the system can answer classical association queries and support the classical mode of interaction. To do this, a user would simply set all the appropriate parameters at the beginning and turn off all the breakpoints so that the system will not prompt for feedback. Since a classical association query is a special case of a constrained association query, our formalism (*i.e.* the notion of CAQ and the proposed architecture) is a true generalization of the classical association query. But, of course, the real power of the generalized framework lies in the fact that

it addresses the problems of the classical association rules listed in section 1.3 by supporting efficient human-centered exploration for association mining.

3.3 Syntax and Examples

In Section 3.1, we briefly describe what is a CAQ. Here, we discuss it in more details and formally introduce the syntax used in CAQ. We also present some examples of CAQ.

In definition 3.1, $\mathcal{C}$ represents a set of constraints, where the constraints can be divided into two classes. A *single variable constraint* (1-var) is a constraint containing only one set variable. It is used in conditioning the antecedent and/or consequent separately. On the other hand, a *two variable constraint* (2-var) is a constraint with two set variables and is useful in expressing joint conditions on both the antedecent and the consequent.

To be more precise, $\mathcal{C}$ is a conjunction of constraints on S_1, S_2 drawn from the following classes of constraints.

1. Single Variable Constraints: A single variable $(1-var)$ constraint is of one of the following forms.

   (a) *Class Constraint*: It is of the form $S \subset A$, where S is a set variable and `A` is an attribute. It says S is a set of values from the domain of attribute `A`.

   (b) *Domain Constraint*: It is of one of the following forms.

      i. $S\theta v$, where S is a set variable, v is a constant from the domain that S comes from, and θ is one of the boolean operators $=, \neq, <, \leq, >, \geq$. It says that *every* element of S stands in relationship θ with the constant value v.

ii. $v\theta S$, where S, v are as above, and θ is one of the boolean operators $\in, \notin$. This simply says the element v belongs to (or not) the set S.

iii. $V\theta S$, or $S\theta V$, where S is a set variable, V is a set of constants from the domain S ranges over, and θ is one of $\subseteq, \not\subseteq, \subset, \not\subset, =, \neq$.

(c) *Aggregate Constraint*: It is of the form $agg(S)\theta v$, where agg is one of the aggregate functions $min, max, sum, count, avg$, and θ is one of the boolean operators $=, \neq, <, \leq, >, \geq$. It says that the aggregate of the set of numeric values in S stands in relationship θ to v.

2. Two Variable Constraints: A two variable constraint $(2-var)$ is of one of the following forms.

(a) $S_1\theta S_2$, where S_i is a set variable and θ is one of $\subseteq, \not\subseteq, \subset, \not\subset$.

(b) $(S_1 \circ S_2)\theta V$, where S_1, S_2 are set variables, V is a set of constants or $\emptyset$, $\circ$ is one of $\cup$,$\cap$, and θ is one of $=, \neq, \subseteq, \not\subseteq, \subset, \not\subset$.

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

We next illustrate the constraint syntax with examples. First of all, we need to specify the view on which we do the mining. We will refer to this as the *minable view*. It can be simply thought of as a set of relations. For example, `trans(TID, Itemset)` can be the relation to represent the transaction. Furthermore, as an example here, we assume that we are interested in the type, price and origin of the items. Therefore, a minable view could be the two relations, `trans(TID, Itemset)`, `itemInfo(Item, Type , Price , Origin )`. Here `Item` represents the set of all possible items and $S \subset$ `Item` would mean S is a set variable in the `Item` domain. We now consider some 1-var constraint examples. S.`Price` ≥ 50 says *all* items in S are of price greater than or equal to \$50; S.`Type` $\supseteq \{sodas\}$ insists S include some items whose type is sodas; S.`Origin` = `Canada` means all the items are made in Canada.

More complicated example can be like $S.\texttt{Type} \cap \{dairy\} = \emptyset \wedge sum(S.\texttt{Price}) \leq 100$, which says S is the set of items that do not include any dairy product and the total price is less than or equal to \$100.

2-var constraints can be constructed in a similar manner. For example, $S_1.\texttt{Type} \cap S_2.\texttt{Type} = \emptyset$ says S_1 and S_2 have no same type items in common; $avg(S_1.\texttt{Price}) \leq avg(S_2.\texttt{Price})$ insists that the average price of S_1 has to be less than the average price of S_2.

Below we give examples of complete CAQs. The CAQ

$$\{(S_1, S_2) | S_1 \subset \texttt{Item} \;\&\; S_2 \subset \texttt{Item} \;\&\; count(S_1) = 1 \;\&\; count(S_2) = 1 \;\&\; freq(S_1) \;\&\; freq(S_2)\}$$

asks for all pairs of single items satisfying frequency constraints. Since the body of any CAQ, $\mathcal{C}$, invariably contains the frequency constraints and the domain constraints $S_1 \subset \texttt{Item} \;\&\; S_2 \subset \texttt{Item}$, we hereafter suppress them and simply note that they are implicitly contained in the body of any CAQ. The CAQ

$$\{(S_1, S_2) | avg(S_1.\texttt{Price}) \leq 100 \;\&\; avg(S_2.\texttt{Price}) \leq 200\}$$

asks for pairs of itemsets, where the average price of S_1 has to be less than or equal to \$100 while that of S_2 has to be less than or equal to \$200. This is an example of using 1-var constraints. On the other hand, the CAQ

$$\{(S_1, S_2) | max(S_1.\texttt{Price}) \leq min(S_2.\texttt{Price})\}$$

represents an example with 2-var constraints, where the query asks for pairs of sets of cheaper items and sets of more expensive items. We can construct more complicated example by adding conjuncting constraints. The CAQ

$$\{(S_1, S_2) | S_1.\texttt{Type} = magazines \;\&\; S_1.\texttt{Origin} = \texttt{Canada} \;\&\; S_2.\texttt{Type} = toys$$
$$\&\; S_2.\texttt{Origin} \neq \texttt{Canada} \;\&\; max(S_1.\texttt{Price}) \leq min(S_2.\texttt{Price})\}$$

finds pairs of sets of cheaper Canadian magazine items and sets of more expensive imported toys. Apart from the `Item` domain, we can also ask for items from

another domain. For example, the CAQ

$$\{(T_1, T_2) | T_1 \subset \texttt{Type} \ \& \ T_2 \subset \texttt{Type}\}$$

finds pairs of sets of types (corresponding to items bought together). Similarly, we can also ask for items from different domains:

$$\{(S_1, S_2) | S_1 \subseteq \texttt{Item} \ \& \ S_1.\texttt{Origin} = \texttt{Canada} \ \& \ S_2 \subseteq \texttt{Origin} \ \& \ \texttt{Canada} \notin S_2\},$$

which asks for sets of domestic items and non-Canadian origins.

3.4 Classification of Constraints

3.4.1 Anti-monotonicity

In this section, we identify two properties of constraints that will be shown to be important when we consider performance optimization. The first property is *anti-monotonicity*. The motivation behind it is from the observation that the success of the Apriori Algorithm for classical association mining is based on the fact that if a set violates the frequency constraint, then some of its supersets will also violate the frequency constraint. We can generalize this property to other constraints and therefore have the following definition.

Definition 3.2 (**Anti-monotonicity**) A 1-var constraint C is anti-monotone if and only if for any set S,

$$\text{S does not satisfy C} \Rightarrow \forall S' \supseteq S, S' \text{ does not satisfy C.}$$

The power of the property lies in the fact that if a set is found to violate the constraints, all its supersets can be pruned away from further consideration. Thus, the number of candidate sets can be largely reduced. As we said above, this is the

1-var Constraint	Anti-Monotone	Succinct
$S\theta v, \theta \in \{=, \leq, \geq\}$	yes	yes
$v \in S$	no	yes
$S \supseteq V$	no	yes
$S \subseteq V$	yes	yes
$S = V$	partly	yes
$min(S) \leq v$	no	yes
$min(S) \geq v$	yes	yes
$min(S) = v$	partly	yes
$max(S) \leq v$	yes	yes
$max(S) \geq v$	no	yes
$max(S) = v$	partly	yes
$count(S) \leq v$	yes	weakly
$count(S) \geq v$	no	weakly
$count(S) = v$	partly	weakly
$sum(S) \leq v$	yes	no
$sum(S) \geq v$	no	no
$sum(S) = v$	partly	no
$avg(S)\theta v, \theta \in \{=, \leq, \geq\}$	no	no
(frequency constraint)	(yes)	(no)

Table 3.1: Classification of 1-var Constraints

reason why Apriori is efficient. If we can identify which classes of constraints satisfy the anti-monotone property, then we can incorporate the constraints along with the frequency constraint to achieve the same pruning power of the Apriori Algorithm.

We have analyzed different classes of 1-var constraints and have classified them according to whether they are *anti-monotone*. We summarize our findings in Table 3.1. The second column identifies which constraints are anti-monotone. The third column does the same for succinctness, which will be discussed in the next section. The first group of constraints is the domain constraints. $S\theta v, \theta \in \{=, \leq, \geq\}$, where v is a constant, is anti-monotone. In particular, it is anti-monotone for θ being $\leq$. The reason is that if $S \not\leq v$, then one of the elements in S is greater than v, and therefore any supersets of S will violate the same constraint because that one

element in S that is greater than v is also contained in the superset. On the other hand, $v \in S$ is not anti-monotone because if S does not contain v, it does not necessarily imply all of its superset will not contain v. A counter example is simply $S \cup \{v\}$. Thus, $v \in S$ does not satisfy the condition for anti-monotonicity. For the min-max type constraints, $min(S) \geq v$ is anti-monotone because if the minimum of S is less than v, then the minimum of all its supersets will also be less than v as the minimum can only be made smaller by adding in extra elements. Similarly, since the maximum of a set can only be increased by adding extra elements, the constraint $max(S) \leq v$ therefore also satisfies the anti-monotone property. The same idea applies to the constraint $count(S) \leq v$. The $count(S)$ (*i.e.* the number of elements in S) can only be increased if the size of the sets is increased. Therefore, if $count(S) > v$, $count(S')$ will also be greater than v for all supersets S'. Thus, $count(S) \leq v$ is anti-monotone as well. We can summarize our discussion by the following proposition.

Proposition 3.1 For each constraint C listed in Table 3.1, C is anti-monotone if and only if the table indicates so.

Optimization Using Anti-Monotonicity

The motivation behind the notion of anti-monotonicity and succinctness is to provide performance optimization based on these properties. One naive algorithm for mining constrained associations is to run Apriori first followed by constraint checking at the end. However, unless the constraints $\mathcal{C}$ is very non-restrictive, only a fraction of the answers found by Apriori would satisfy the constraints. A large portion of the computation may turn out to be unnecessary. However, the time spent on Apriori is typically a lot more than the time spent on constraint checking. The algorithm with constraint filtering at the end is therefore very inefficient. The basic idea of improving performance is then to "push" the constraint checking as early as

possible so that the constraints can help us prune away candidate sets before support counting. However, one has to be careful in pushing the constraints into lower levels (lower level here means smaller $k-$value where k is the size of the itemsets). This is because dropping candidate sets at lower levels may lead to missing frequent itemsets at higher levels. We say an algorithm is *complete* if all frequent sets satisfying the given constraints can be found. In other words, all solutions are included in the answers returned by the algorithm. On the other hand, we also want to make sure that the algorithm is *sound*, *i.e.* all the answers found are valid solutions that are frequent and satisfy the given constraints.

Anti-monotonicity allows us to prune away candidate sets at each level in a way similar to the case of of frequency constraint. In fact, the standard optimization used for the frequency constraint in the Apriori Algorithm is based on the following property (P1):

$$S,\ where\ |S| = k, is frequent \Longrightarrow \forall S' \subseteq S\ where\ |S'| = k-1, S' is frequent,$$

so that whenever any one of the size $k-1$ subsets of a size k candidate set is not frequent, the candidate set can be pruned away. In the case of a constrained association query $\{S_1, S_2|\mathcal{C}\}$, if $\mathcal{C}_{am}$ consists of all the anti-monotone constraints in $\mathcal{C}$, including the frequency constraint, then a similar optimization technique can be used based on the following property (P2) generalized from the above property (P1):

$$S,\ where\ |S| = k,\ satisfies\ \mathcal{C}_{am} \Longrightarrow \forall S' \subseteq S\ where\ |S'| = k-1, S'\ satisfies\ \mathcal{C}_{am}.$$

In other words, if L_{k-1} consists of all the sets of size $k-1$ that satisfy $\mathcal{C}_{am}$, then the set C_k of candidate sets of size k can be generated in exactly the same way as in the Apriori Algorithm. Then, C_k can be further pruned to become C_k^{am} by checking whether each element in C_k satisfies every constraint in $\mathcal{C}_{am}$ other than the frequency constraint. Any element that violates any constraint in $\mathcal{C}_{am}$ can be dropped away from support counting. As will be seen in Chapter 5, this optimization is incorporated in Algorithm CAP.

3.4.2 Succinctness

While *anti-monotonicity* provides strong pruning power, it involves an iterative generate-and-test process: at each level, a list of candidates is generated and then tested for the satisfaction of the constraints. It would be better if we can eliminate the generate-and-test paradigm. This leads to the the following two questions: (i) Under what circumstances can we succinctly characterize the set of all itemsets that satisfy a given constraint ? (ii) Given a constraint that has a succinct description, how can we generate all the itemsets that satisfy the constraint without admitting spurious itemsets ? The answer for the first question leads to the notion of *succinctness* discussed below while the second question lead us to the notion of a *member generating function.*

To help understand the definition, we consider the sample example in section 3.3, *i.e.* the minable view consists of the relations `trans(TID, Itemset)` and `itemInfo(Item,Type,Price,Origin)`. We denote the set of itemsets that satisfy any 1-var constraint C by $SAT_C(\texttt{Item})$, which we refer to as the *pruned space* of C. For example, if C_1 is the constraint $S.\texttt{Origin} = \texttt{Canada}$, then $SAT_{C_1}(\texttt{Item})$ contains all those itemsets whose origin is Canada. In the following, a *selection predicate* refers to any predicate that is allowed to appear as a parameter of a selection operation in relational algebra and $\sigma_p(I)$ represents the subset of I containing all the elements of I that satisfy the selection predicate p. In our example, $\sigma_p(I)$ with $I \subseteq \texttt{Item}$ is therefore $\{i \in I | \exists t \in \texttt{trans} \bowtie \texttt{itemInfo} : t.\texttt{Item} = i and t \text{ satisfies } p\}$. Moreover, we use the notation 2^I to denote the strict powerset of I, *i.e.* the set of all subsets of I except the empty set. We can now define succinctness as follows.

Definition 3.3 (Succinctness)

1. $I \subseteq \texttt{Item}$ is a *succinct* set if it can be expressed as $\sigma_p(\texttt{Item})$ for some selection predicate p.

2. $SP \subseteq 2^{\texttt{Item}}$ is a *succinct powerset* if there is a fixed number of succinct sets $\texttt{Item}_1, ..., \texttt{Item}_k \subseteq \texttt{Item}$ such that SP can be expressed in terms of the strict powersets of $\texttt{Item}_1, ..., \texttt{Item}_k$ using union and set difference.

3. Finally, a 1-var constraint C is *succinct* provided $SAT_C(\texttt{Item})$ is a succinct powerset.

Intuitively, the definition states that a constraint on a set of items is succinct if its solution set can be expressed as sums of sets that can be obtained by applying some selection predicate onto the set of all items. The constraint $C_1 \equiv S.\texttt{Origin} = \texttt{Canada}$ considered above is succinct because its pruned space $SAT_{C_1}(\texttt{Item})$ is simply $2^{\texttt{Item}_c}$, where $\texttt{Item}_c = \sigma_{\texttt{Origin}=\texttt{Canada}}(\texttt{Item})$. Now, let's consider a more complicated example. If $C_2 \equiv \{snacks, sodas\} \subseteq S.\texttt{Type}$, then the pruned space consists of all the sets that contain at least one item of type *snacks* and at least one item of type *sodas*. Let $\texttt{Item}_2, \texttt{Item}_3, \texttt{Item}_4$ be the sets $\sigma_{\texttt{Type}=snacks}(\texttt{Item}), \sigma_{\texttt{Type}=sodas}(\texttt{Item})$,and$\sigma_{\texttt{Type}\neq snacks \wedge \texttt{Type}\neq sodas}(\texttt{Item})$ respectively. Then, C_2 is succinct because the pruned space $SAT_{C_2}(\texttt{Item})$ can be expressed as:

$$2^{\texttt{Item}} - 2^{\texttt{Item}_2} - 2^{\texttt{Item}_3} - 2^{\texttt{Item}_4} - 2^{\texttt{Item}_2 \cup \texttt{Item}_4} - 2^{\texttt{Item}_3 \cup \texttt{Item}_4}.$$

Similar to the anti-monotone property, we have classified the representative class of constraints according to whether they are succinct or not in Table 3.1.

Proposition 3.2 For each constraint C listed in Table 3.1, C is succinct if and only if the table says so.

We have proved two positive cases above. Now, consider the constraint $max(S.A) \geq c$. Let $\texttt{Item}_1 = \sigma_{A<c}(\texttt{Item})$. Then the pruned space is the powerset of all the items excluding the powerset of $\texttt{Item}_1$. In other words, $SAT_C(\texttt{Item}) = 2^{\texttt{Item}} - 2^{\texttt{Item}_1}$. On the other hand, $sum(S)\theta v, \theta \in \{=, \leq, \geq\}$ is not succinct. The reason is that given any finite union and differences of succinct powersets, we cannot

rule out the possibility in general that there exists additional items not belonging to those powersets, but whose addition to S can still satisfy the constraint. Similar arguments apply to the remaining cases, and therefore, for brevity, we now skip the proof of all the other cases except to make a note of those constraints involving $count(S)$.

Consider the constraint $count(S) \leq v$. This constraint is not succinct and does not have an MGF of the kind introduced in Definition 3.4 below. However, we can have an MGF based on cardinality constraint, *i.e.* $\{X | X \subseteq \texttt{Item} \,\&|X| \leq v\}$. Therefore, we say that the constraint $count(S) \leq v$ is *weakly succinct.* Similarly, the other constraints involving $count(S)$ in Table 3.1 are also weakly succinct.

We now turn to the next question of how to generate $SAT_C(\texttt{Item})$ given C is succinct. The key concept here is the *member generating function.*

Definition 3.4 Member Generating Functions

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

For the constraint $C_1 \equiv S.\texttt{Origin} = \texttt{Canada}$ discussed above, an MGF is simply $\{X | X \subseteq \texttt{Item}_c \& X \neq \emptyset\}$ where $\texttt{Item}_c = \sigma_{\texttt{Origin=Canada}}(\texttt{Item})$. As for the constraint $C_2 \equiv \{snacks, sodas\} \subseteq S.\texttt{Type}$, an MGF is $\{X_1 \cup X_2 \cup X_3 | X_1 \subseteq \texttt{Item}_2 \;\&\; X_1 \not\emptyset \;\&\; X_2 \subseteq \texttt{Item}_3 \& X_2 \not\emptyset \;\&\; X_3 \subseteq \texttt{Item}_4\}$, where $\texttt{Item}_2, \texttt{Item}_3$, and $\texttt{Item}_4$ are as defined earlier.

Optimizations Using Succinct Constraints

The relationship between succinctness and MGF is that every succinct constraint has an MGF.

Proposition 3.3 If C is succinct, then C is pre-counting prunable (*i.e.* $SAT_C(\texttt{Item})$ has an MGF).

Proof Our argument is based on an induction on the number of minus operator in the succinctness expression. When there is no minus operation, $SAT_C(\texttt{Item}) = 2^{\texttt{Item}_1}$ with $\texttt{Item}_1 = \sigma_p(\texttt{Item})$ and p being the predicate in C. Then the MGF is simply $\{X | X \subseteq \texttt{Item}_1 \ \& \ X \neq \emptyset\}$. We assume that the proposition is true whenever the expression of $SAT_C(\texttt{Item})$ in terms of succinct powersets involves k minus operations. Now, consider any constraint such that $SAT_C(\texttt{Item})$ has $k+1$ minus operations, *i.e.* $SAT_C(\texttt{Item}) = 2^{\texttt{Item}} - 2^{\texttt{Item}_1} - \cdots - 2^{\texttt{Item}_k} - 2^{\texttt{Item}_{k+1}}$, where $\texttt{Item}_i$ are succinct sets, $\texttt{Item}_i = \sigma_{q_i}(\texttt{Item})$ for some predicate q_i, $i = 1, \ldots, k+1$. By the induction assumption, $SAT_C(\texttt{Item}) = \{X_1 \cup \cdots \cup X_n | X_i \subseteq \sigma_{p_i}(\texttt{Item})\} - 2^{\texttt{Item}_{k+1}}$, which this is equal to $\{X_1 \cup \cdots \cup X_n | X_i \subseteq \sigma_{p_i \wedge \neg q_{k+1}}(\texttt{Item})\}$ of the form stated in Definition 3.4. (One of the X_j has to be non-empty, otherwise $SAT_C(\texttt{Item})$ is too trivial). Thus, the proposition is also true for $k+1$.

The importance of this proposition is that, for succinct constraints, we can now operate in a generate-only fashion, instead of in a generate-and-test manner. Furthermore, the satisfaction of the constraint alone is not affected in any way by the result of the iterative support counting.

So far, we have considered only a single constraint. To obtain MGFs for a more general constraint consisting of multiple constraints, we can simply combine the MGFs of two constraints at a time. In other words, suppose C_1 and C_2 are constraints with MGFs $\{S_1 \cup \cdots \cup S_m | S_i \subseteq \sigma_{p_i}(\texttt{Item})\}$ and $\{T_1 \cup \cdots \cup T_n | T_j \subseteq \sigma_{q_j}(\texttt{Item})\}$ respectively, then the MGF for C_1 & C_2 is simply $\{R_{11} \cup \cdots \cup R_{mn} | R_{ij} \subseteq$

$\sigma_{p_i \wedge q_j}(\texttt{Item})\}$.

We incorporate the optimization idea presented in this section into our proposed CAP Algorithm. Before we present a detailed description of the CAP Algorithm, we turn to some of the basic algorithms in the following chapter.

Chapter 4

Basic Algorithms

In this chapter, we develop a few algorithms for computing frequent sets that satisfy a set of given constraints. In the next chapter, we will present the Algorithm CAP which takes advantage of the optimization ideas given in Section 3.4. We will compare the performance of CAP and the other algorithms in Chapter 6.

4.1 Apriori$^+$

The first algorithm is a straightforward extension of the classical Apriori Algorithm [AS94]. The idea is to run Apriori first, followed by constraints checking at the end. It is called Apriori$^+$ and is listed below.

Algorithm 4.1 (Apriori$^+$)

1 C_1 consists of all sets of size 1; $k = 1$; $Ans = \emptyset$;

2 while (C_k not empty) {

2.1 conduct database scan to form L_k from C_k;

2.2 form C_{k+1} from L_k based on C_{freq}; $k++$;}

3 for each set S in some L_k:

add S to Ans if it satifies $(\mathcal{C} - \mathcal{C}_{freq})$.

Here $\mathcal{C}$ is used to denote the set of given constraints, one of which is the frequency constraint, $\mathcal{C}_{freq}$. All the other symbols (C_k, L_k, and Ans) have the usual meaning discussed before in Chapter 2. Apriori$^+$ is almost the same as the Apriori (Algorithm 2.1) except in Step 3 where each of the frequent sets computed from Step 1 and Step 2 is verified against the remaining constraints (*i.e.* $\mathcal{C} - \mathcal{C}_{freq}$).

4.2 Full Materization

Algorithm Apriori$^+$ is not bad if the frequency constraint $\mathcal{C}_{freq}$ is more selective than the remaining ones $(\mathcal{C} - \mathcal{C}_{freq})$. However, the converse may be true. In other words, it is possible that most of the frequent sets found do not satisfy the remaining constraints. In that case, checking $(\mathcal{C} - \mathcal{C}_{freq})$ first, followed by verifying C_{freq}, might be more efficient. This lead to the following algorithm which we call *Full Materization* (FM).

Algorithm 4.2 (FM)

1 C_1 consists of all sets of size 1; $Ans = \emptyset$;

2 $C^\star = C_1$;

3 For every possible S such that for all $S' \subseteq S \wedge |S'| = 1 \Rightarrow S' \in C_1$:

add S' to $C^\star$ if S' satisfies $(\mathcal{C} - \mathcal{C}_{freq})$;

4 conduct database scan for sets in $C^\star$; add sets that are frequent to Ans.

While Algorithm FM avoids counting support for sets that do not satisfy the given constraints, it is not without its own problem. Its problem is in Step 3 where all the sets are generated and tested against the constraints in $(\mathcal{C} - \mathcal{C}_{freq})$. The number of sets involved grows exponentially with large problem size (*i.e.* the number of items). In that case, Algorithm FM becomes very inefficient.

4.3 Hybrid(m)

If the frequency constraint $\mathcal{C}_{freq}$ is much more selective than the remaining constraints in $(\mathcal{C} - \mathcal{C}_{freq})$, Algorithm Apriori$^+$ is more efficient. Conversely, if the constraint $(\mathcal{C} - \mathcal{C}_{freq})$ are much more selective, then Algorithm FM performs better. The two algorithms, in fact, represent two extreme cases where either the frequency constraint $\mathcal{C}_{freq}$ or the remaining constraints $(\mathcal{C} - \mathcal{C}_{freq})$ is clearly more selective than the other. However, the selectivity of the two sets of constraints ($\mathcal{C}_{freq}$ and $\mathcal{C} - \mathcal{C}_{freq}$) are often comparable. It may be a good idea to combine the above two algorithms in a divide-and-conquer approach where we run one algorithm up to a certain level and switch to the other afterwards. We call the result Hybrid(m), where m is the level when the switching occurs. With m treated as a parameter, Hybrid(m), in fact, represents not only a single algorithm but a class of algorithms.

Algorithm 4.3 (Hybrid(m))

1 C_1 consists of sets of size 1; $k = 1$; $Ans = \emptyset$;

2 while (C_k not empty and $k \leq m$) {

  2.1 conduct database scan to form L_k from C_k;

  2.2 form C_{k+1} from L_k based on C_{freq}; $k++$;}

3 if $m > 0$ then for each set S in some L_k;

  add S to Ans if S satisfies $(C - C_{freq})$;

4 if C_k not empty then {

4.1 $C^{\star} = C_{m+1}$;

4.2 for every possible S such that for all $S' \subseteq S \wedge |S'| = m+1 \Rightarrow S' \in C_{m+1}$:
add S' to $C^{\star}$ if S' satisfies $(C - C_{freq})$;

4.3 conduct database scan for sets in $C^{\star}$; add sets that are frequent to Ans.}

If $m = 0$, Steps 2 and 3 will not be executed and the algorithm (Hybrid(0)) becomes Algorithm FM. Otherwise, Algorithm Hybrid(m) runs Apriori^{+} up to level m ($m > 1$) in Step 1 to Step 3. It checks C_{freq} for m iterations to produce the standard C_{m+1} set which consists of all candidate sets of size $m + 1$. This takes advantage of the pruning affected by the frequency constraint. Then, instead of continuing with Apriori^{+}, Hybrid(m) switches to FM in Step 4. The motivation for the switch is to reduce the I/O costs involved with the database scanning in each iteration of Apriori^{+}. A tradeoff is a higher CPU cost in Step 4.2 where all the possible sets are generated and tested again the constraints $(C - C_{freq})$. However, compared with Algorithm FM, the CPU cost for the same step is much less and may become acceptable. In fact, the CPU cost will be much less in Hybrid(n) than in Hybrid(m) whenever $n < m$. The reason is that the generation in Step 4.2 needs to consider a set S only when all the subsets of size $m + 1$ of S are in C_{m+1}. As m increases, the number of sets in C_{m+1} decreases rapidly, thus reducing the number of sets required to consider in Step 4.2.

To summarize, the decision of when the switching should occur (*i.e.* the value of m) represents a tradeoff between CPU and I/O costs. With small m value, the amount of database scanning required is less and the I/O costs is reduced, at the expense of increasing CPU cost in Step 4.2. On the other hand, if m is larger, the number of candidate sets in Step 4 is smaller and the CPU time is reduced, at the expense of increasing the I/O cost of Step 2. Moreover, the two extreme

cases of $m = 0$ and $m = \infty$ correspond to Algorithm FM and Algorithm Apriori$^+$ respectively.

Chapter 5

Algorithm CAP

While all the algorithms considered in the previous chapter have their own merits, they all fail to exploit the properties of the constraints. In particular, the anti-monotone and succinct properties introduced in Section 3.4 are not being used. Algorithm CAP discussed in this chapter attempts to make maximum use of these properties by pushing the constraints as deep "inside" the computation as possible. To this end, we classify all the constraints into four categories:

I. Constraints that are both anti-monotone and succinct (e.g., $min(S) \geq v$);

II. Constraints that are succinct but not anti-monotone (e.g., $min(S) \leq v$);

III. Constraints that are anti-monotone but not succinct (e.g., $sum(S) \leq v$); and

IV. Constraints that are neither (e.g.,$avg(S) \leq v$).

A different strategy is tailored for each of the above cases.

5.1 Succinct and Anti-monotone Constraints

When the constraint C is both succinct and anti-monotone, we propose that the general form of the MGF for $SAT_C(\texttt{Item})$ in Definition 3.4 reduces to the form $\{S|S \subseteq \sigma_p(\texttt{Item}) \ \& \ S \neq \emptyset\}$. Then the set C_1 of candidate sets of size one in the Apriori Algorithm can simply be replaced by $C_1^c = \{e|e \in C_1 \& e \in \sigma_p(\texttt{Item})\}$. Moreover, since C is anti-monotone, sets containing any element *not* in C_1^c need not be considered. Therefore, we have the following strategy proposed for succinct anti-monotone constraints.

Strategy I:

- Replace C_1 in the Apriori Algorithm by C_1^c defined above.

5.2 Succinct but Non-anti-monotone Constraints

In the case of succinct non-anti-monotone constraints, one cannot guarantee that all sets satisfying the constraints must be subsets of C_1^c. For example, consider the constraint $S.\texttt{Type} \supset \{sodas\}$ which says S includes at least one soda item. It is succinct because $SAT_C(\texttt{Item})$ has a MGF $\{S_1 \cup S_2|S_1 \subseteq \sigma_{Type=sodas}(\texttt{Item}) \ \& \ S_1 \neq \emptyset \ \& \ S_2 \subseteq \texttt{Item}\}$. If a set S does not contain a soda item, S's supersets may. Therefore, the constraint is not anti-monotone. For this constraint, C_1^c would contain all the size 1 itemsets where the items in the itemsets are all sodas. However, sets such as $\{Coke, Milk\}$ satisfy the constraint but are not subset of C_1^c. Thus, Strategy I is not complete. Fortunately, in this case, we can make use of the structure given by the MGF of the constraint. As in our example, we consider MGF's that are of the form $\{S_1 \cup S_2|S_1 \subseteq \sigma_{p_1}(\texttt{Item}) \ \& \ S_2 \subseteq \sigma_{p_2}(\texttt{Item})\}$.

Strategy II:

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

- Define $C_2 = \{\{e, f\} | e \in L_1^c \& f \in (L_1^c \cup L_1^{\neg c})\}$, and L_2, as usual, the set of frequent sets in C_2, *i.e.* , $L_2 = \{S | S \in C_2 \& freq(S)\}$.

- In general, C_{k+1} can be obtained from L_k in exactly the same way as in the Apriori Algorithm with only one modification. In the classical case, a set S is in C_{k+1}, if all its subsets of size k are in L_k, *i.e.* ,

$$\forall S' : S' \subset S \text{ and } |S'| = k \Rightarrow S' \in L_k$$

  In the case of succinct, non-anti-monotone constraints, we only need to check subsets S' that intersect with L_1^c. In other words, subsets S'' that are disjoint with L_1^c are excluded in the above verification. These are sets that only contain elements from $L_1^{\neg c}$, and that are not counted for support. Because we do not know whether they are frequent or not, we give them the benefit of the doubt. Hence, we have the following modification in candidate generation. A set S is in C_{k+1} if for all subsets:

$$\forall S' : S' \subset S \text{ and } S' \cap L_1^c \neq \emptyset \Rightarrow S' \in L_k.$$

  As usual, $L_{k+1} = \{S | S \in C_{k+1} \ \& \ freq(S)\}$.

Example 5.1 Suppose that `Item` is the set $\{1, \ldots, 100\}$, C_1^c is $\{1, \ldots, 50\}$, and $C_1^{\neg c}$ is $\{51, \ldots, 100\}$. Furthermore, suppose that L_1^c and $L_1^{\neg c}$ turn out to be $\{1, \ldots, 20\}$ and $\{71, \ldots, 100\}$. Then, C_2 is constructed by considering the Cartesian product $\{1, \ldots, 20\} \times \{1, \ldots, 20, 71, \ldots, 100\}$ which gives all the size 2 sets that can be frequent and that contain at least one element from C_1^c. We further suppose that $\{1, 71\}$ and $\{1, 72\}$ are frequent, and that $\{1, 73\}$ is not. Now, in considering whether $\{1, 71, 72\}$ should be in C_3, we only check whether $\{1, 71\}$ and $\{1, 72\}$ are frequent.

In the classical case, we would check $\{71, 72\}$ as well. However, we do not check it here because its support is unknown. Given the benefit of the doubt, $\{1, 71, 72\}$ is included in C_3. However, neither $\{1, 71, 73\}$ nor $\{1, 72, 73\}$ is in C_3, because $\{1, 73\}$ is not in L_2.

We can generalize the above Strategy for more general MGF. For example, the constraint $C_2 \equiv \{snacks, sodas\} \subseteq S.\texttt{Type}$ is succinct, but not anti-monotone. Its MGF is $\{X_1 \cup X_2 \cup X_3 | X_1 \subseteq \texttt{Item}_2 \ \& \ X_1 \neq \emptyset \ \& \ X_2 \subseteq \texttt{Item}_3 \ \& \ X_2 \neq \emptyset \ \& \ X_3 \subseteq \texttt{Item}_4\}$, where $\texttt{Item}_2, \texttt{Item}_3$, and $\texttt{Item}_4$ are as defined earlier. Our strategy is to first form the sets $L_1^{X_i} = \{e | e \in \texttt{Item}_{i+1} \ \& \ freq(\{e\})\}$ for each $X_i, i = 1, 2, 3$. Then we generate the candidate set $C_2 = L_1^{X_1} \times L_1^{X_2}$. From C_2, we can find L_2 (by counting the support of the sets in C_2). Next, we form $C_3 = L_2 \times (L_1^{X_1} \cup L_1^{X_2} \cup L_1^{X_3})$. After that, L_k and C_{k+1} are computed as usual, with the only modification in candidate generation that S is in C_{k+1} iff: for all subsets $S' \subset S$ and $|S'| = k$ and $S' \cap L_1^{X_1} \neq \emptyset$ and $S' \cap L_1^{X_2} \neq \emptyset \Rightarrow S' \in L_k$.

5.3 Non-Succinct but Anti-monotone Constraints

So far, we have assumed that the constraint is succinct. If the constraint C is not succinct, then it does not admit an MGF which can be used to avoid the generate-and-test paradigm completely. Fortunately, we can still make use of the anti-monotone property. This is done by checking whether the candidate sets, generated at each level, satisfy C, *before* counting is done. Because of anti-monotonicity, candidate sets that do not satisfy C can be safely dropped right away without counting for their support. Thus, we have the following strategy.

Strategy III:

- Define C_k as in the classical Apriori Algorithm. Drop a set $S \in C_k$ from count-

ing whenever S fails satisfying the constraints, *i.e.* , constraint satisfaction is tested before counting is undertaken.

5.4 Non-Succinct and Non-anti-monotone Constraints

If the constraint C is neither succinct nor anti-monotone, then it seems that we cannot use the constraint to perform any optimization. Fortunately, from our experience, many such constraints induce *weaker constraints* that might be anti-monotone and/or succinct. If such constraints can be found, then they can be exploited using one of the strategies outlined above. In other words, the weaker constraints are used to find the frequent sets. The only modification is that at the end of the algorithm, one final round of testing the frequent sets for satisfaction of the original constraint C is necessary. It is because the frequent sets, while guaranteed to satisfy the weaker constraints, may not necessarily satisfy C. If the weaker constraints do not admit too many spurious solutions (*i.e.* frequent sets that satisfy the weaker constraints, but not C), then the above strategy should be fairly efficient because the last of constraints checking is relatively trivial.

Example 5.2 Consider the constraint $C \equiv avg(S.A) \geq v$. It is neither succinct nor anti-monotone. However, it induces the weaker constraint $C' \equiv max(S.A) \geq v$. In other words, every set S that satisfies C is guaranteed to satisfy C'. Since C' is a succinct, but non-anti-monotone constraint, Strategy II can be applied. After all frequent sets are found, each one of them has to be tested for satisfaction of C to generate the final answers.

Strategy IV:

- Induce any weaker constraint C' from C. Depending on whether C' is anti-monotone and/or succinct, use one of the strategies I-III above for the gener-

ation of frequent sets.

- Once all frequent sets are generated, test them for satisfaction of C.

5.5 Multiple Constraints

So far, we has assumed that we are dealing with a single constraint. However, the set of constraints $\mathcal{C}$ in a CAQ may more often contain multiple constraints. An important question is then how the strategies we proposed for individual constraints can be combined to handle multiple constraints. To this end, the set of constraints $\mathcal{C}$ is divided into four sets of constraints according to whether the constraints are succinct and/or anti-monotone. We denote the set of constraints that are both succinct and anti-monotone by $\mathcal{C}_{sam}$. Similarly, $\mathcal{C}_{suc}$ denotes constraints that are succinct but not anti-monotone; $\mathcal{C}_{am}$ denotes constraints that are anti-monotone but not succinct; Finally, $\mathcal{C}_{none}$ denotes constraints that are neither anti-monotone nor succinct.

Strategy for Handling Multiple Constraints:

- Combine the MGFs of all the constraints in C_{sam}. Apply Strategy I with the combined MGF.
- Combine the MGFs of all the constraints in C_{suc}. Apply the Strategy II with the combined MGF.
- For all constraints in C_{am}, follow Strategy III.
- For each constraint in C_{none}, induce weaker constraints and apply Strategy IV.

Example 5.3 Let C be the constraints $min(S.\texttt{Price}) \geq 100$ & $S.\texttt{Type} \supseteq \{sodas\}$. Then, $\mathcal{C}_{sam} \equiv min(S.\texttt{Price}) \geq 100$ and $\mathcal{C}_{suc} \equiv S.\texttt{Type} \supseteq \{sodas\}$. To find fre-

quent sets that satisfy C, we apply Strategy I with the MGF of the constraint $min(S.\texttt{Price}) \geq 100$ and Strategy II with the MGF of the constraint $S.Type \supseteq \{sodas\}$.

5.6 Summary

The basic algorithms discussed in the previous chapter do not take into account the property of the constraint. In designing our optimization strategies, we make use of the two identified properties, namely, anti-monotonicity and succinctness, and try to apply constraints checking as early as possible. We design four different strategies (**Strategy I-IV**)depending on whether the constraint is anti-monotone and/or succinct.

We also discuss how to handle multiple constraints in Section 5.5. We incorporate all the optimization strategies discussed in this chapter into the following algorithm called CAP (Constrained Apriori) for finding frequent sets that satisfy a set of given constraints $\mathcal{C}$.

Algorithm 5.1 (CAP)

1 if $C_{sam} \cup C_{suc} \cup C_{none}$ is non-empty, prepare C_1 as indicated in Strategies I, II and IV; $k = 1$;

2 if C_{suc} is non-empty {

  2.1 conduct database scan to form L_1 as indicated in Strategy II;

  2.2 form C_2 as indicated in Strategy II; $k = 2$; }

3 while (C_k not empty) {

  3.1 conduct database scan to form L_k from C_k;

  3.2 form C_{k+1} from L_k based on Strategy II if C_{suc} is non-empty, and Strategy III for constraints in C_{am};}

4 if C_{none} is empty, $Ans = \bigcup L_k$. Otherwise, for each set S in some L_k, add S to Ans iff S satisfies C_{none}.

Algorithm CAP is one of the most important contribution in this thesis. As we will see in the following chapter, the performance of Algorithm CAP is much better than the other basic algorithms discussed in Chapter 5. To conclude this chapter, we state the soundness and completeness of Algorithm CAP as a proposition.

Proposition 5.1 A constraint C can be in one of the categories:

I Succinct and Anti-monotone

II Succinct but Non-anti-monotone

III Non-succinct but Anti-monotone

IV Non-succinct and Non-anti-monotone

If C is in category X (X=I-IV), then Strategy X proposed in Sections 5.1-5.4 is sound and complete with respect to computing the frequent sets that satisfy C.

Chapter 6

Experimental Evaluation

In this chapter, we evaluate the performance of the various mining algorithms given in Chapter 4 and 5 based on our experimental results.

6.1 Implementation and Experimental Environment

We implemented the algorithms Apriori$^+$, Hybrid(m), and CAP in C. Instead of writing three different programs, we wrapped all the algorithms under one single program which we call `caq`. The rationale is to try to share as much data structures and library modules as possible so that a fair comparison can be achieved. In running `caq`, choosing which mining algorithm to use is simply a matter of supplying different flags. The command `caq -apriori (-hybrid(m), -cap)` finds all the frequent itemsets with the Apriori$^+$ (Hybrid(m), CAP respectively) Algorithm. The different constraints were hard-coded using compiler directives.

One major decision in the implementation was what data structures should be used in representing itemsets. Our initial attempt was to use bit vectors. Set operations such as determining subset relationship were achieved by bitwise oper-

ations which were very efficient. However, we later realized that using bit vectors leads to a memory problem. With 1000 items, each itemset would require 125 bytes or 32 integers (assuming 4 bytes for an integer). The number of itemsets in C_2 could be of the order of 500K. This implies that at least 60MB is required just to hold C_2! Therefore, we gave up the bit vectors representation. Instead, we used lists. A size k itemset is represented by a list of k integers where each integer corresponds to the item's identification number. Since the maximum size of a large itemset is typically less than 10, we save at least 2/3 of the memory when compared with bit vector representation.

We used the program **gen** developed at IBM Almaden Research Center [AS94] to generate transactional databases. **Gen** is a program that can generate data synthetically for both associations and sequential patterns. It was downloaded from the IBM QUEST web site (`http://www.almaden.ibm.com/cs/quest`). It requires several input parameters. The most important ones are the number of transactions in the databases and the number of items. While we experimented with various databases, the results cited below are based on a database of 100,000 transactions and a domain of 1000 items. Moreover, the average number of items in a transaction is 25 and the maximal size of a large itemset is set to 10. The items are represented by integers from 1 to 1000. The page size used was 4 Kbytes.

Our experiments were conducted in a time-sharing SPRAC-10 environment. The memory size was 128MB and the typical number of users was about five. We scheduled our experiments mainly at night so that the workload for other processes was relatively low and uniform. We also made sure that our program was the only major running job.

6.2 Succinct and Anti-monotone Constraints

Our first set of experiments compares the various algorithms using the succinct and anti-monotone constraint $max(S.\texttt{Price}) \leq v$. Different values of v correspond to different selectivities of the constraint. If there are x-% of the items whose price is less than or equal to v, then we say the selectivity is x-%. The translation between selectivity and v is as follows. We number the items from 1 to 1,000. Without loss of generality, the `Price` value of each item is taken to be exactly its number. Then, the selectivity for a particular v value is simply $\frac{v}{10}$%. We then run our experiments with different selectivities and different support values.

We show our results by plotting speedup against selectivity and support threshold. The speedup of various algorithms is defined to be the execution time of Apriori$^+$ divided by that of the algorithm being considered. In other words, we measure the speedup relative to Algorithm Apriori$^+$. Figure 6.1 plots the speedup as a function of constraint selectivity, with support threshold set at 0.5%, and Figure 6.2 shows the relationship between the speedup and support threshold, with the selectivity fixed at 30%.

We first discuss our results for the Hybrid(m) algorithms. We find that Hybrid(0), Hybrid(1), and Hybrid(2) all take much too long when compared with Apriori$^+$. This shows that the CPU cost of finding all the sets that satisfy the constraints in a generate-and-test mode is prohibitive. Hybrid(3), Hybrid(4), etc. take more or less the same time as Apriori$^+$, almost always within 5% of each other. Algorithm Hybrid(m) is the same as Algorithm Apriori$^+$ at the first m iterations. It then tries to reduce remaining database scanning time at the expense of increasing CPU time. However, the I/O cost is dominated by the first few iterations. Therefore, for $m = 3, 4, ..,$ savings on the I/O cost corresponding to later iterations do not lead to any observable reduction in execution time when compared with Apriori$^+$. It is quite possible that with a larger database size, Hybrid(m) algorithms will become

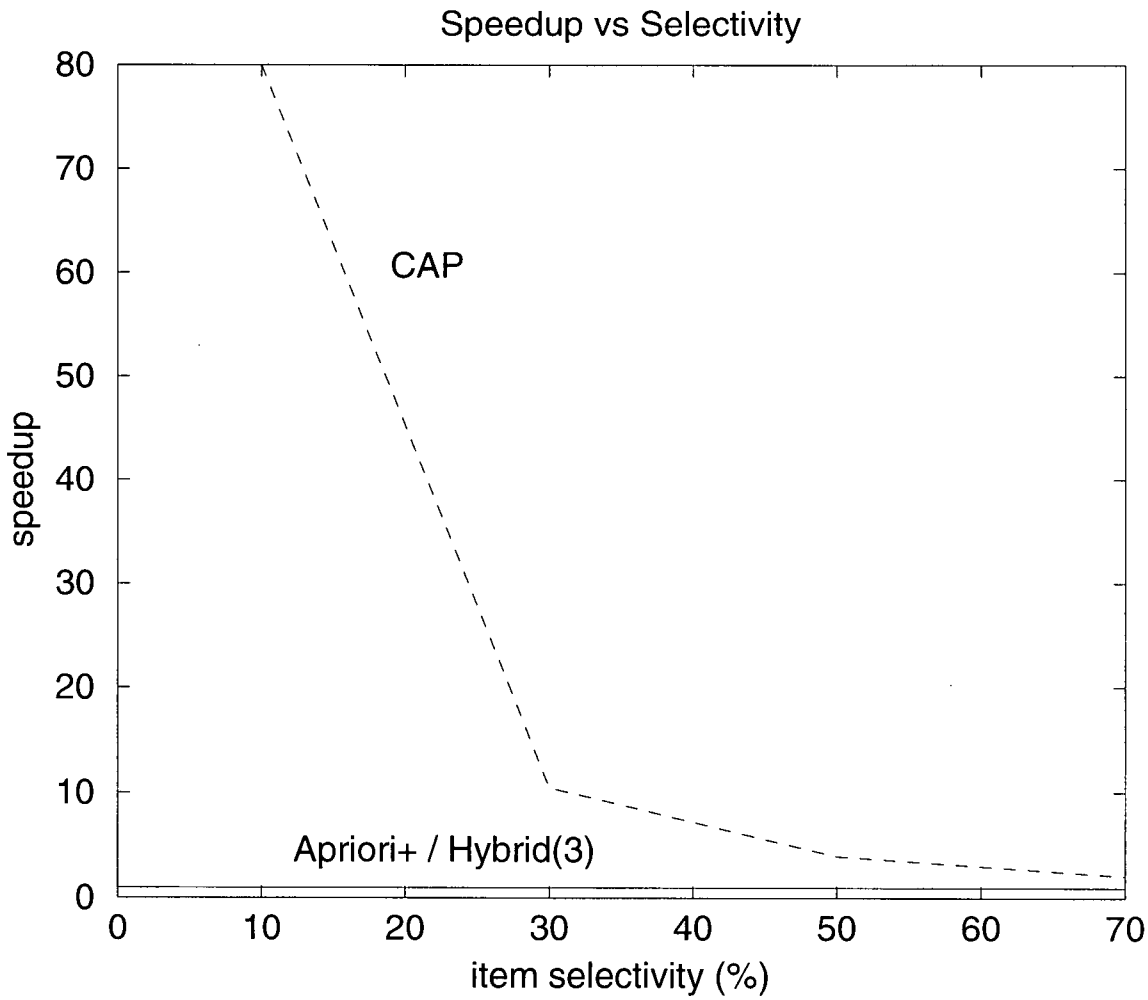

Figure 6.1: Speedup vs Selectivity for a Succinct and Anti-monotone Constraint

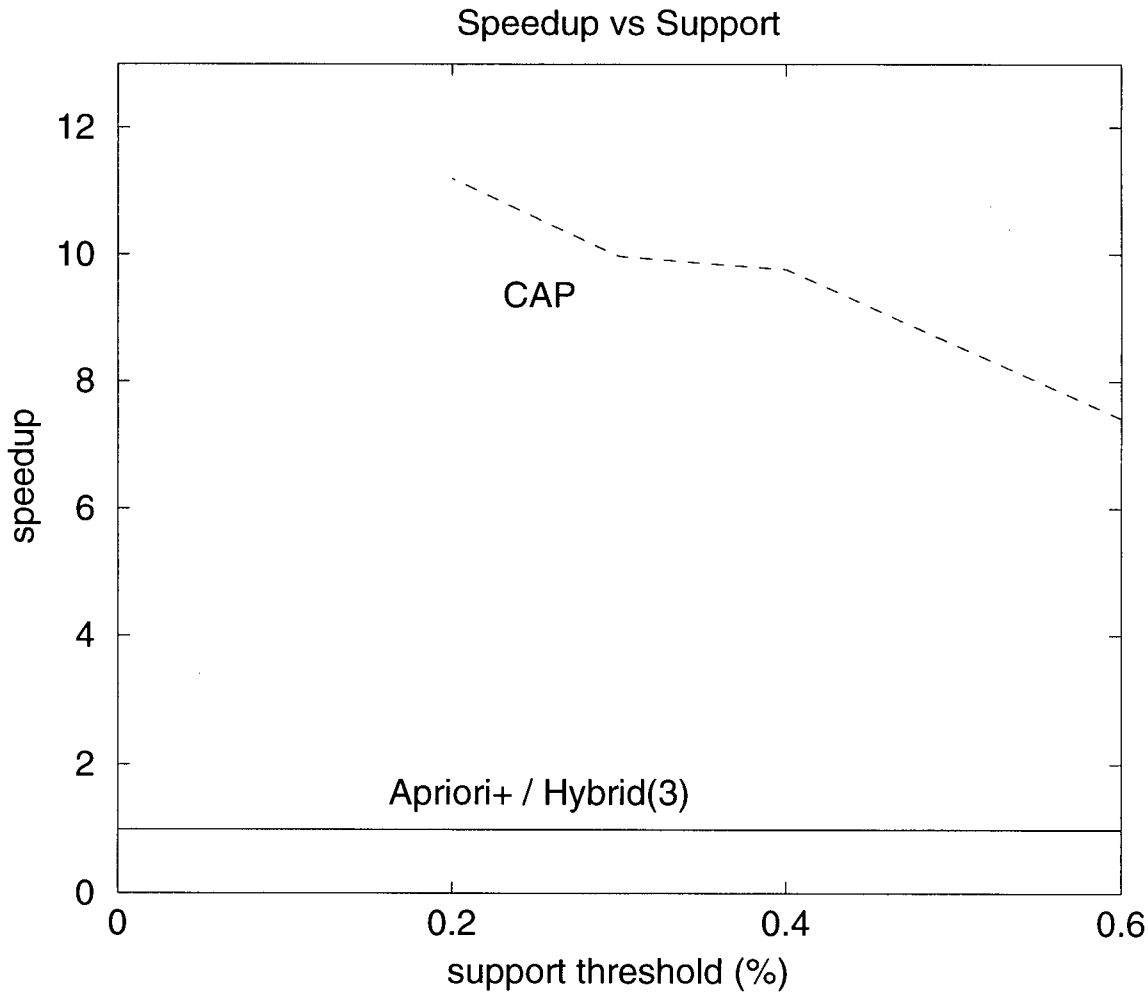

Figure 6.2: Speedup vs Support for a Succinct and Anti-monotone Constraint

selectivity	5	10	30	50	70
L_1	19/362	40/362	112/362	66/362	254/362
L_2	0/66	0/66	14/66	26/66	41/66
L_3	0/91	0/91	10/91	21/91	45/91
L_4	0/105	0/105	5/105	15/105	40/105
L_5	0/77	0/77	1/77	6/77	22/77
L_6	0/35	0/35	0/35	1/35	7/35
L_7	0/9	0/9	0/9	0/9	1/9
L_8	0/1	0/1	0/1	0/1	0/1

Table 6.1: Number of frequent sets that satisfy $max(S.\texttt{Price} < v)$ /number of frequent sets for different selectivities ($v/10$) with support = 0.5%

more efficient than Apriori$^+$. However, it is quite unlikely that their efficiency can be comparable with Algorithm CAP. Therefore, in the following discussion, our focus will be on Algorithm CAP.

We include Table 6.1 and 6.2 to help illustrate our results. In both tables, each column corresponds to a different setting. The rows correspond to the sizes of the frequent sets. Each entry is of the form a/b, where a is the number of frequent sets satisfying the constraint, and b is simply the total number of frequent sets of that size. Table 6.1 contains results for the various selectivities used in Figure 6.1, whereas Table 6.2 shows the results for different support thresholds corresponding to the settings used in Figure 6.2.

We now consider the relationship between speedup and item selectivity. As one can see from Figure 6.1, Algorithm CAP clearly outruns Apriori$^+$. At 10% selectivity, the speedup for CAP is about 80 times. Even at a selectivity of 30%, CAP still runs 10 times faster than Apriori$^+$.

One can understand such a huge speedup (> 80) of CAP for selectivities less than 10% by examining the first two column in Table 6.1. For a 5% selectivity (*i.e.* the first column), Apriori$^+$ has to find all the 362 frequent sets of size 1, 66

support	0.6%	0.5 %	0.4%	0.3%	0.2%
L_1	98/313	112/362	131/421	159/503	174/582
L_2	1/12	14/66	22/160	40/333	79/969
L_3	0/1	10/91	11/186	17/390	29/1140
L_4	0/0	5/105	5/210	6/427	8/1250
L_5	0/0	1/77	1/70	1/309	1/934
L_6	0/0	0/35	0/18	0/140	0/451
L_7	0/0	0/9	0/8	0/36	0/132
L_8	0/0	0/1	0/2	0/4	0/20

Table 6.2: Number of frequent sets that satisfy $max(S.\texttt{Price}) < v$) /number of frequent sets for different support values at 30% selectivity

frequent sets of size 2, 91 frequent sets of size 3, $\cdots$ and so on , before it stops. On the contrary, CAP stops after the first iteration. The number of frequent sets it has to find is only 19. Furthermore, the number of database scan it has to perform is only 1 compared with 8 for Apriori$^+$. Similarly, CAP can stop after the first iteration for 10% selectivity. At 30% selectivity, CAP stops after 5 iterations and the speedup is smaller than 80. But still, the speedup is as large as 10 times. This is because the number of frequent sets (*i.e.* 112, 14, 10, 5, and 1 for frequent sets of sizes 1, 2, 3, 4, and 5 respectively) CAP has to process is much smaller than that of Apriori$^+$'s (*i.e.* 362, 66, 91, 105, and 77 for frequent sets of sizes 1, 2, 3, 4, and 5 respectively).

The trend shown in Figure 6.1 is that as the constraint becomes less restrictive (*i.e.* as selectivity increases), the speedup of CAP decreases. This can also be understood by examining the entries in Table 6.1. As the constraint becomes less restrictive, the pruning power provided by the constraint decreases. The number of frequent sets which need to be found by CAP increases (as shown in Table 6.1). Savings obtained by pushing the constraint in the CAP Algorithm become less obvious, thus the speedup decreases.

We now turn to the relationship between speedup and support threshold. The graph in Figure 6.2 shows that the speedup of CAP is about 7 to 11 times for support threshold from 0.2% to 0.6%. We choose the support thresholds such that Apriori$^+$ requires between 5 and 8 iterations. The speedup shown in Figure 6.2 decreases slowly with increasing support threshold.

When the support threshold is low, the number of frequent sets is large. Most of the frequent sets may turn out not to satisfy the constraint. For example, for a support of 0.2% and sets of size 4, Apriori$^+$ finds 1250 frequent sets, but only 8 of which need to be found by CAP ! Moreover, Apriori$^+$ has to find frequent sets of size 6 and up, whereas CAP can stop after size 5. Thus, Apriori$^+$ is relatively much less efficient than CAP leading to a speedup of more than 11 times.

When the support threshold is high, the total number of frequent sets is smaller, and Apriori$^+$ behaves relatively better. At 0.6% support, Apriori$^+$ only iterates one more time than CAP. However, it still processes too many frequent sets potentially violating the constraint, and the speedup for CAP is still as large as 8 times.

To conclude, if the support thresholds are chosen to be small, Apriori$^+$ would require more iterations and the overall speedup of CAP would be larger. If the support thresholds are chosen to be higher so that number of iterations required by Apriori$^+$ decreases, the speedup achieved by CAP would be lower, but it would still be substantial. The reason is that the major savings contribution achieved by CAP come from the very first few iterations.

6.3 Succinct but Non-anti-monotone Constraints

In the next series of experiments, we use the succinct but non-anti-monotone constraint $\{soda\} \subseteq S.\mathsf{Type}$. Similar to the previous case, we illustrate our results by

selectivity (%)	10	13	15	20	50	70	90
L_1	40/362	54/362	60/362	82/362	184/362	254/362	324/362
L_2	0/66	12/66	19/66	35/66	57/66	65/66	66/66
L_3	0/91	15/91	36/91	67/91	87/91	91/91	91/91
L_4	0/105	20/105	55/105	89/105	104/105	105/105	105/105
L_5	0/77	15/77	50/77	71/77	77/77	77/77	77/77
L_6	0/35	6/35	27/35	34/35	35/35	35/35	35/35
L_7	0/9	1/9	8/9	9/9	9/9	9/9	9/9
L_8	0/1	0/1	1/1	1/1	1/1	1/1	1/1

Table 6.3: Number of frequent sets that satisfy $\{soda\} \subseteq S.\mathsf{Type}$ /number of frequent sets for different selectivities with support = 0.5%

plotting speedup against item selectivity (see Figure 6.3) and speedup versus support threshold (see Figure 6.4). An x-% selectivity in Figure 6.3 means that there are x-% of the items whose type is *soda*.

Once again, CAP clearly outruns other algorithms. For example, for selectivities of 5%, 10%, and 20%, CAP runs 9, 4, and 2.5 times faster than Apriori$^+$ respectively (see Figure 6.3). At a selectivity of 13%, CAP can achieve a speedup of about 3.5 times for support thresholds between 0.3% and 0.6% (Figure 6.4).

To capture the pruning achieved by CAP, we show the number of frequent sets that satisfy the constraint (a) and the number of frequent sets (b) in the form a/b in the Tables 6.3 and 6.4. Table 6.3 corresponds to the settings used in Figure 6.3, whereas Table 6.4 uses the same settings as that in Figure 6.4.

When compared with the previous case (*i.e.* the case of succinct and antimontone constraints), the gain here shown by CAP comes entirely from the succinctness of the constraint. For example, at 5% selectivity, the speedup is about 8 times. This shows the relevance of the succinctness property in performance optimization.

When compared with the previous case, the speedup is smaller. At 5% selectivity, it was 80 times in the previous case compared with 8 times here. This

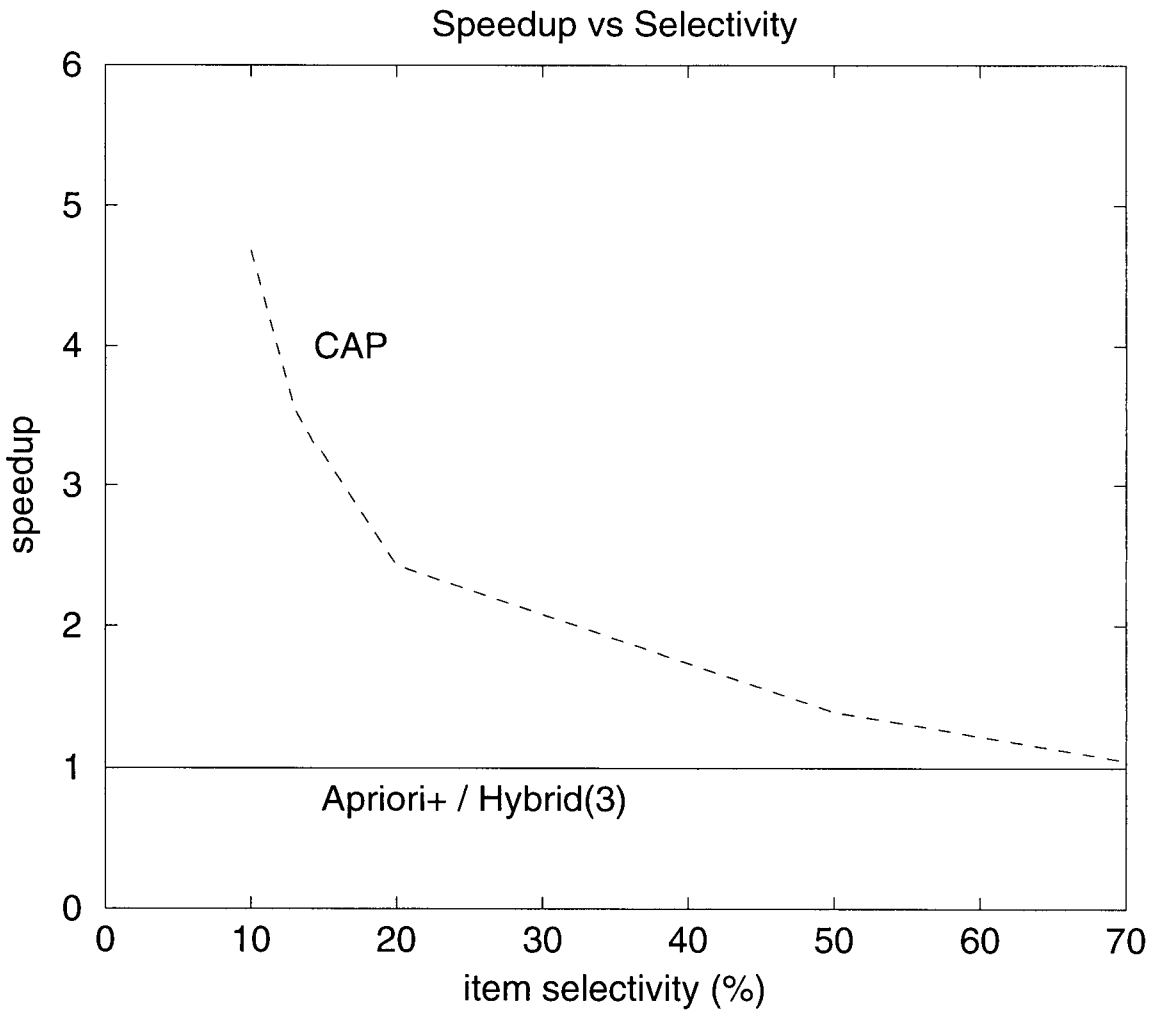

Figure 6.3: Speedup vs Selectivity for a Succinct and Non-anti-monotone Constraint

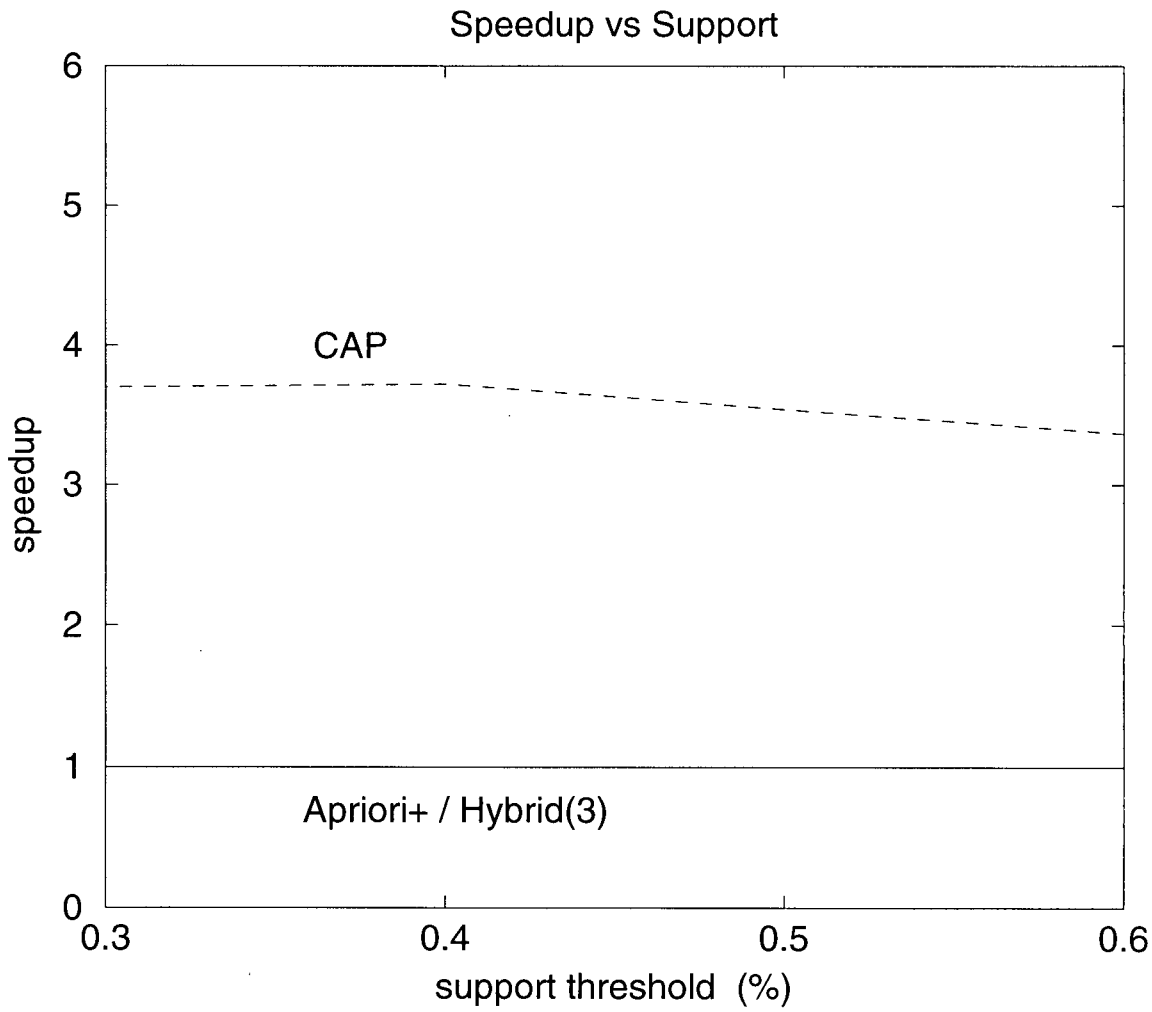

Figure 6.4: Speedup vs Support for a Succinct and Non-anti-monotone Constraint

support	0.6%	0.5 %	0.4%	0.3%
L_1	49/313	54/362	60/421	72/503
L_2	3/12	12/66	39/160	96/333
L_3	0/1	15/91	64/186	157/390
L_4	0/0	20/105	85/210	215/427
L_5	0/0	15/77	70/154	180/309
L_6	0/0	6/35	34/70	90/140
L_7	0/0	1/9	9/18	25/36
L_8	0/0	0/1	1/2	3/4

Table 6.4: Number of frequent sets that satisfy $\{soda\} \subseteq S.\texttt{Type}$ /number of frequent sets for different support values at 13% selectivity

demonstrates that a constraint that is both succinct and anti-monotone effects much more pruning than a constraint that is only succinct. Succinctness combined with anti-monotonicity can be exploited to produce a powerful compound effect on performance optimization.

One way to understand why the speedup in this case is smaller than the previous case is to compare the number of frequent sets that satisfy the constraints in Table 6.1 with that in Table 6.3. At the same selectivity, the number of frequent sets that satisfy the succinct but non-anti-monotone constraint $\{soda\} \subseteq S.\texttt{Type}$ is larger than that for the succinct and anti-monotone constraint $max(S.\texttt{Price}) \leq v$. For example, at 50% selectivity, the number of frequent sets that satisfy the constraint $\{soda\} \subseteq S.\texttt{Type}$ is 184, 57, 87, and 104 for sizes 1, 2, 3, and 4, whereas those numbers are 66, 26, 21, and 15 for the constraint $max(S.\texttt{Price}) \leq v$. This is reasonable as the latter constraint is also anti-monotone and therefore intuitively it should be more restrictive. The speedup for a more restrictive constraint should be higher which is consistent with what we observe in our experiments.

One cannot compare directly Table 6.2 with Table 6.4 because the selectivity for the two tables are set at different values. Nevertheless, we can still see that the

number of frequent sets that satisfy the constraint $max(S.\texttt{Price}) \leq v$ decreases much more rapidly with the size of the frequent sets than that of the constraint $\{soda\} \subseteq S.\texttt{Type}$. This is again consistent with the observation that the speedup in this case is less than the previous case.

6.4 Non-Succinct but Anti-monotone Constraints

Next, we consider the case of non-succinct but anti-monotone constraints. The constraint used is $sum(S.\texttt{Price}) \leq MaxSum$. Similar to the case of succinct and anti-monotone constraints, we assume that the `Price` value of each item is exactly its number, *i.e.* from 1 to 1,000. $MaxSum$ is set to be 500, 1000, 2000 and so on. In contrast with the previous cases, there is no simple translation from $MaxSum$ to item selectivity. Thus, we plot speedup against $MaxSum$ in Figure 6.5, instead of speedup against item selectivity.

Contrary to the previous two cases, the dominance of CAP drops very rapidly as the value of $MaxSum$ increases. While the speedup is above 7 times for $MaxSum \approx 500$, CAP shows no gain over Apriori$^+$ for $MaxSum \geq 2000$. It can be explained by Table 6.5.

When $MaxSum$ is 500, the constraint helps reduce the number of frequent sets at level 1 by a half (*i.e.* 184 out of 362). This reduction is compounded at subsequent levels. Such pruning provided by the constraint leads to a speedup of over 7 times. When $MaxSum$ reaches 1000, all the size 1 frequent sets pass the constraints. Pruning provided from the constraints only starts at level 2 (*i.e.* 39 out of 66). This modest reduction is then compounded at higher levels, giving an overall speedup of about 2 times. However, when $MaxSum$ is increased to 2000, there is no pruning at the first two levels. Pruning only comes at higher levels. Since the major portion of the computation is spent on lower levels, the pruning coming from

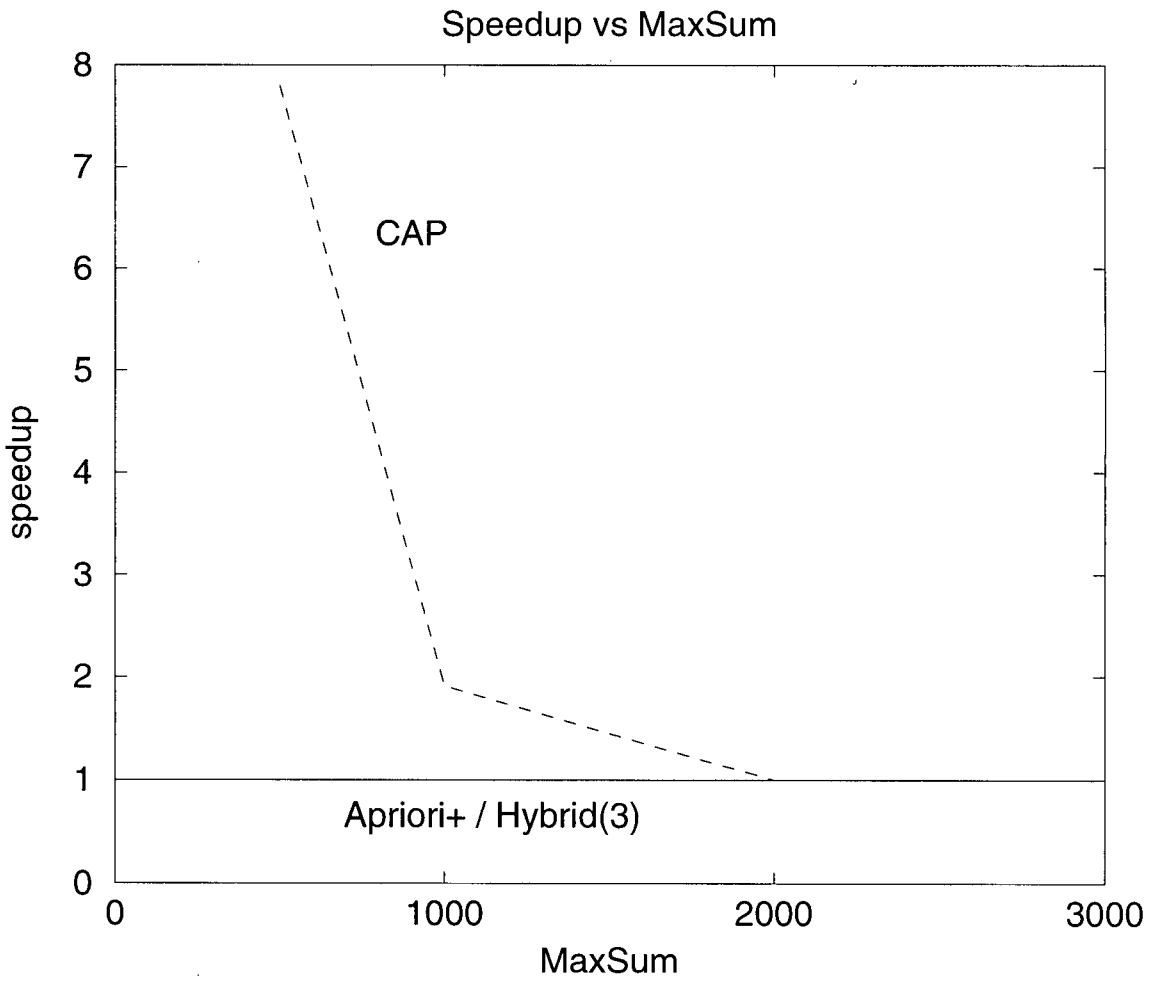

Figure 6.5: Speedup vs Selectivity for a Non-succinct and Anti-monotone Constraint

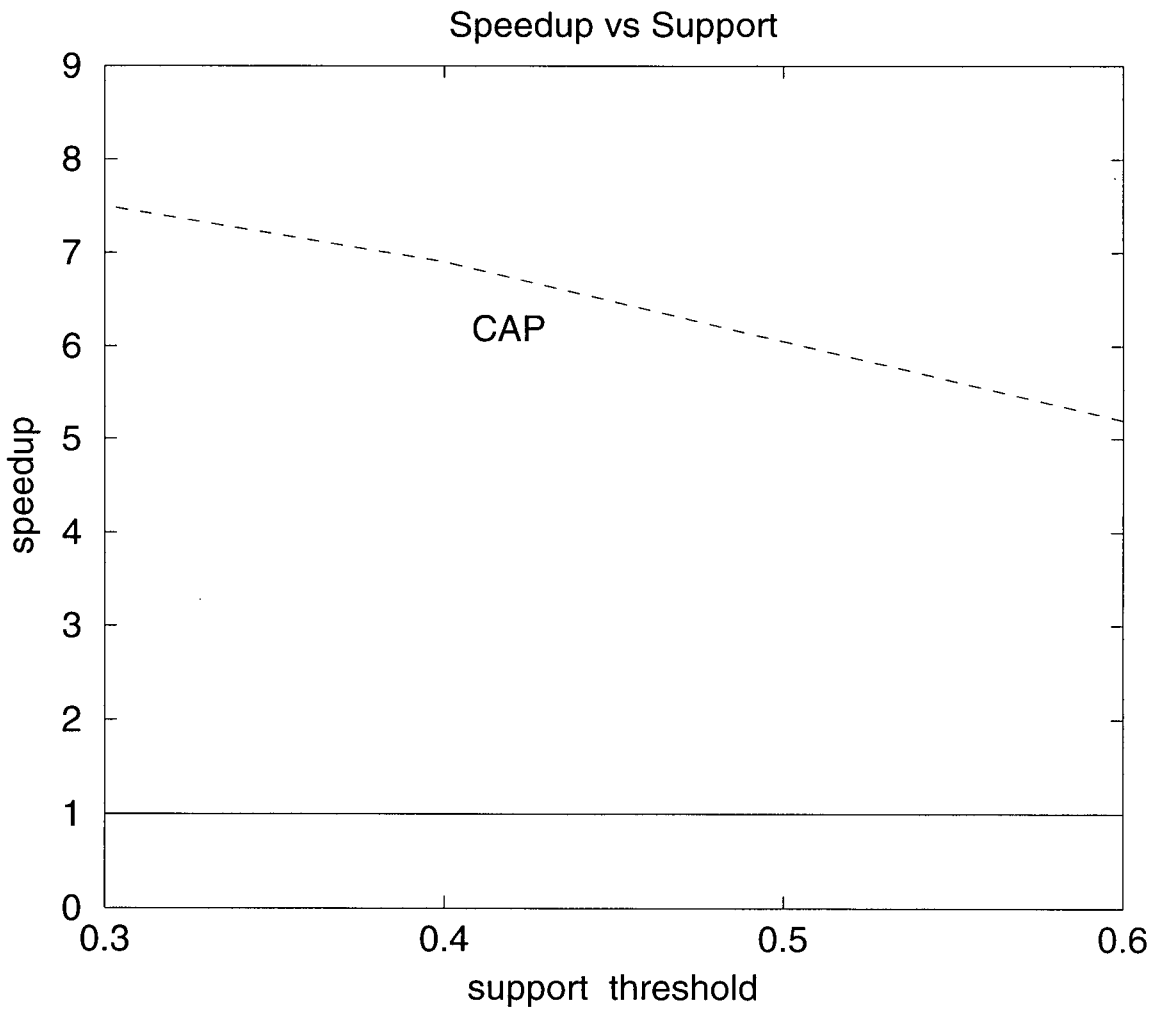

Figure 6.6: Speedup vs Selectivity for a Non-succinct and Anti-monotone Constraint

$MaxSum$	500	1000	2000	3000
L_1	184/362	362/362	362/362	362/362
L_2	16/66	39/766	66/66	66/66
L_3	4/91	28/91	86/91	91/91
L_4	0/105	15/105	77/105	105/105
L_5	0/77	1/77	40/77	72/77
L_6	0/35	0/35	13/35	30/35
L_7	0/9	0/9	1/9	8/9
L_8	0/1	0/1	0/1	0/1

Table 6.5: Number of frequent sets that satisfy $sum(S.\texttt{Price}) < MaxSum$) /number of frequent sets for different $MaxSum$ with support = 0.5%

support	0.6%	0.5%	0.4%	0.3%
L_1	160/313	184/362	207/421	250/503
L_2	1/12	16/66	32/160	58/333
L_3	0/1	4/91	5/186	13/390
L_4	0/0	0/105	0/210	1/427
L_5	0/0	0/77	0/154	0/309
L_6	0/0	0/35	0/70	0/140
L_7	0/0	0/9	0/8	0/36
L_8	0/0	0/1	0/2	0/4

Table 6.6: Number of frequent sets that satisfy $sum(S.\texttt{Price}) < MaxSum$) /number of frequent sets for different support values with $MaxSum = 500$

later levels does not help. Therefore, we can conclude that in order for any pruning optimization to be significant, it has to be effective *as early as possible* at the first two levels. In fact, any reduction from lower levels is compounded to higher levels giving rise to a significant speedup.

In Figure 6.6 and Table 6.6, we show our results for the relationship between speedup and support threshold. Similar to the other cases, if the constraint produces fairly large pruning, the speedup is as large as 5 to 8 times for various support thresholds from 0.3% to 0.6%.

6.5 Non-Succinct and Non-anti-monotone Constraints

Finally, the last category to investigate is non-succinct and non-anti-monotone constraints. In particular, we consider the constraint $Avg(S.\texttt{Price}) \leq v$. This constraint is difficult to optimize. Fortunately, it induces the weaker constraint $min(S.\texttt{Price}) \leq v$. In order for S to satisfy the first constraint, S has to satisfy the latter constraint. Since the constraint $min(S.\texttt{Price}) \leq v$ is succinct but non-anti-monotone, we can use Strategy II in Section 5.2 to compute frequent sets that satisfy it. Then, for each of the frequent sets found, we check whether it satisfies the original constraint $Avg(S.\texttt{Price}) \leq v$. Figure 6.7 and 6.8 shows our results.

Similar to the other cases, we also supplement the figures with two tables (Table 6.7 and 6.8) to illustrate what is going on. Instead of just a/b, the entries in these tables are in the form $a/b/c$ where a, b, and c are the number of frequent sets that satisfy the original constraint, the induced constraint and no constraint respectively.

Figures 6.7 and 6.8 show that the speedup of CAP over Apriori$^+$ is very similar to the case for a typical succinct but non-anti-monotone constraint, even though only a small portion of the frequent sets that satisfy the induced constraint does

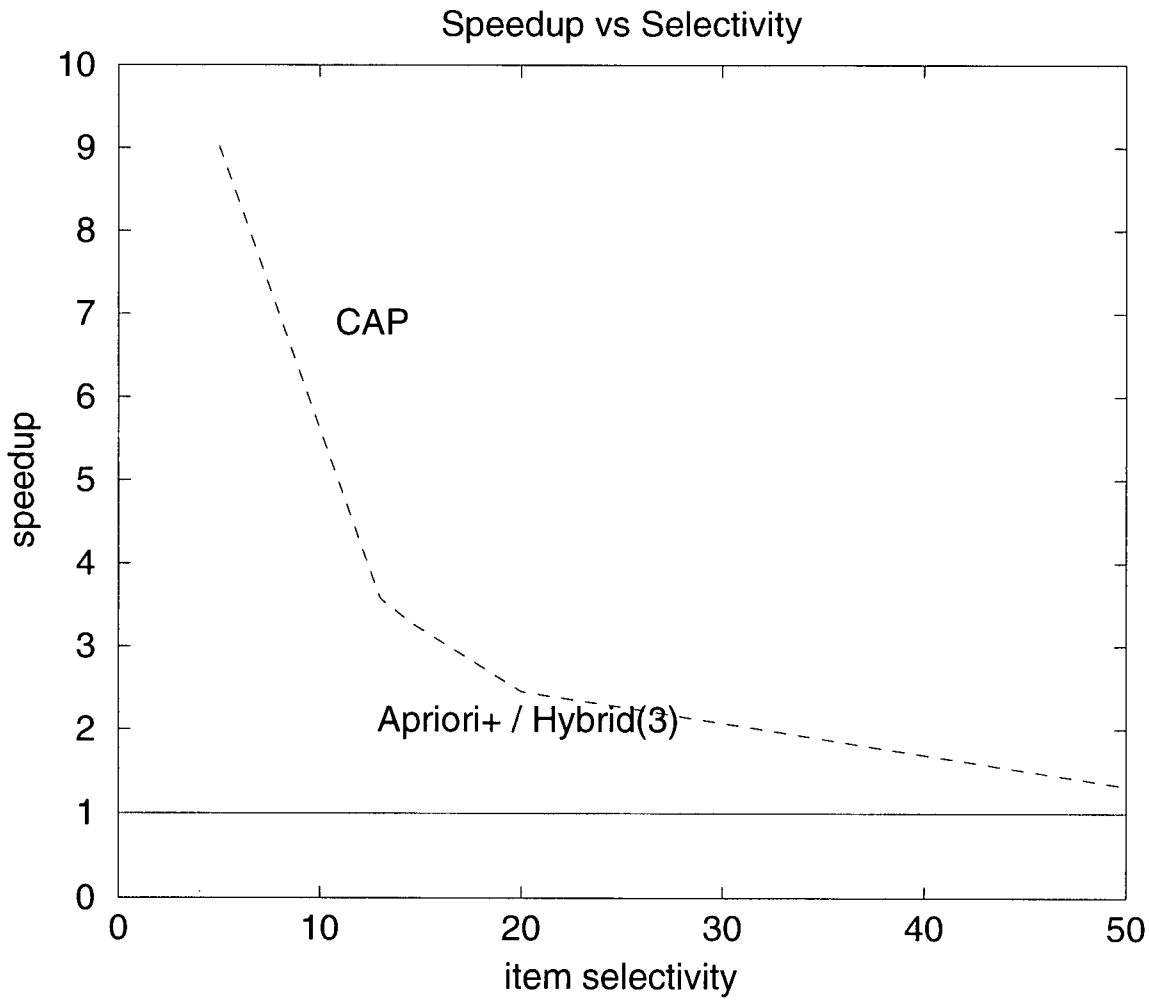

Figure 6.7: Speedup vs Selectivity for a Non-succinct and Non-anti-monotone Constraint

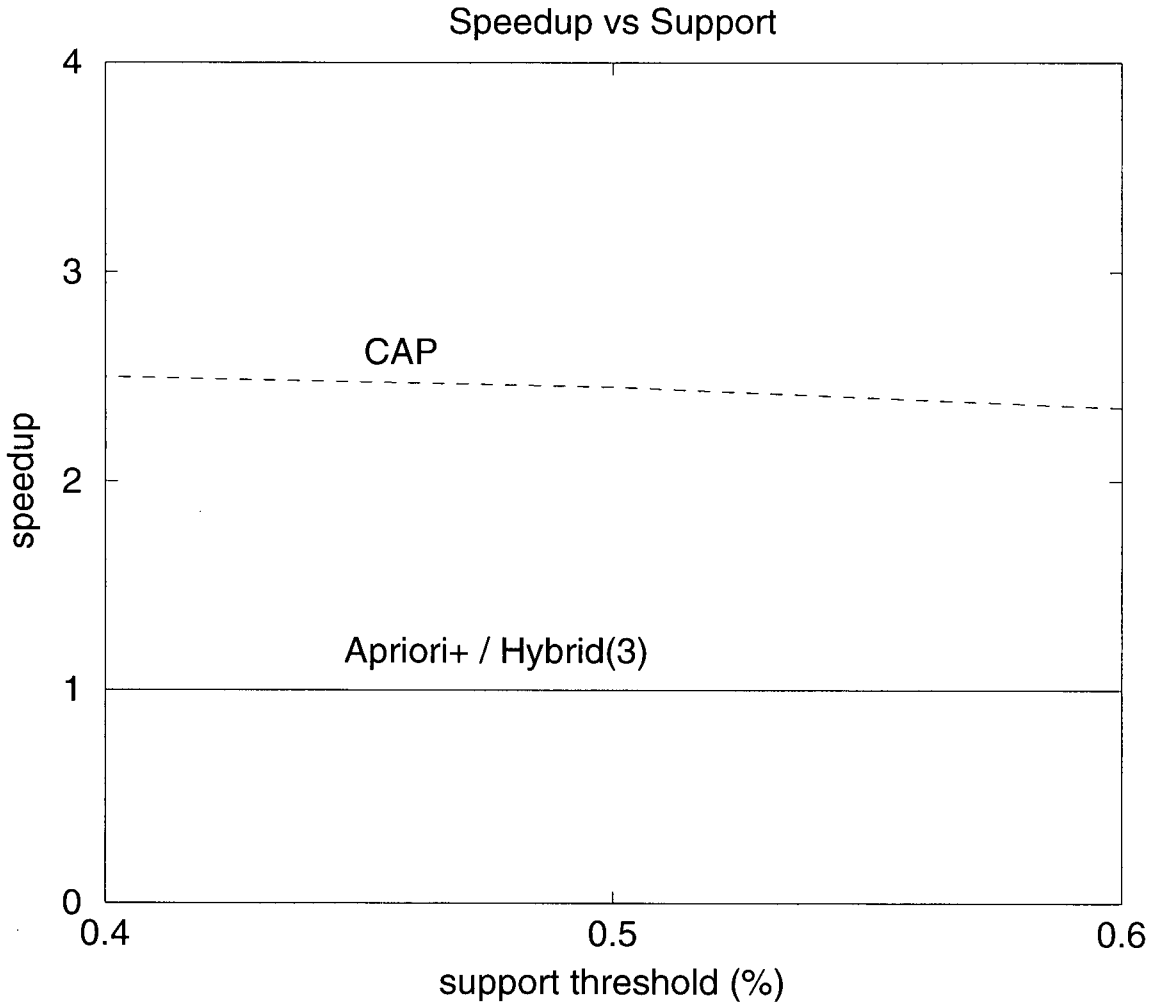

Figure 6.8: Speedup vs Support for a Non-succinct and Non-anti-monotone Constraint

AvgPrice	50	130	145	200	500
L_1	19/19/362	54/54/362	59/59/362	82/82/362	184/184/362
L_2	0/3/66	1/12/66	2/19/66	9/35/66	39/57/66
L_3	0/0/91	0/15/91	0/36/91	7/67/91	64/87/91
L_4	0/0/105	0/20/105	0/55/105	5/89/105	77/104/105
L_5	0/0/77	0/15/77	0/50/77	1/71/77	63/77/77
L_6	0/0/35	0/6/35	0/27/35	0/34/35	30/35/35
L_7	0/0/9	0/1/9	0/8/9	0/9/9	8/9/9
L_8	0/0/1	0/0/1	0/1/1	0/1/1	1/1/1

Table 6.7: Number of frequent sets that satisfy $Avg(S.\texttt{Price}) < AvgPrice$ / number of sets that satisfy $min(S.\texttt{Price}) < AvgPrice$ / number of frequent sets for different values of $AvgPrice$ with support = 0.5%

support	0.6%	0.5%	0.4%	0.3%
L_1	74/74/313	82/82/362	96/96/421	114/503/503
L_2	0/5/12	9/35/66	18/79/160	38/158/333
L_3	0/0/1	7/67/91	10/131/186	23/268/390
L_4	0/0/0	5/89/105	5/174/210	12/350/427
L_5	0/0/0	1/71/77	1/141/154	2/282/309
L_6	0/0/0	0/34/35	0/68/70	0/136/140
L_7	0/0/0	0/9/9	0/18/18	0/36/36
L_8	0/0/0	0/1/1	0/2/2	0/4/4

Table 6.8: Number of frequent sets that satisfy $Avg(S.\texttt{Price}) < AvgPrice$ / number of sets that satisfy $min(S.\texttt{Price}) < AvgPrice$ / number of frequent sets for different support values with $AvgPrice = 200$

happen to satisfy the original constraint. This indicates that the final verification step takes a relatively small amount of time and the idea of using induced weaker constraints works quite well.

6.6 Summary

To summarize, among all the various algorithms discussed in Chapters 4 and 5, CAP is the most efficient. Hybrid(m) $m \geq 3$ are comparable with Apriori$^+$. On the other hand, Hybrid(0), Hybrid(1), and Hybrid(2) are the slowest. While the exact speedup of the CAP Algorithm over the Apriori$^+$ depends on many factors and varies under different situations, we find that CAP can easily outrun Apriori$^+$ by a factor of 5 to 10 in many cases. We also find that the speedup increases with decreasing item selectivity (*i.e.* with more restrictive constraints) and decreases with increasing support thresholds. The main factor in determining the speedup of CAP is on how much pruning the constraints provide. Both succinctness and anti-monotonicity produce significant pruning power and their combined effect is even stronger. Thus, the speedup of CAP can be as much as 80 times in the case of succinct and anti-monotone constraints.

Chapter 7

Conclusions

7.1 Conclusions

There has been an increasing demand for discovering useful information from very large databases. Data mining, which can be defined as an automatic process of discovering hidden useful patterns from large databases, has recently generated a high degree of interest. Among the most popular data mining patterns discovered in large transactional databases are association rules. A classical association rule takes the form of $X \Rightarrow Y$, where $X \cup Y$ is a set of items that appears at least $minsup\%$ of the times in the database and among those transactions that contain X, at least $c\%$ of them also contains Y. $Minsup$ and c are two input parameters referred to as the support threshold and the confidence respectively. The motivation of mining association rules is to find out what items tend to appear together. In a classical association query, the user only need to specify the values for the support threshold and the confidence. The answer to the query is a list of all the association rules found.

However, this classical association mining model suffers from (i) lack of user

exploration and control, (ii) lack of focus, and (iii) rigid notion of relationship. In particular, there may be no association rules found or thousand of rules found with most of them not being what the user is looking for. In this thesis, we have defined a model where a user can specify additional constraints in the query. The type of constraints can include class, domain, and aggregation constraints. Then, the system can focus the mining task on associations that satisfy the specified constraints. This allows the system to find associations that have a better chance of satisfying the user's need. We have suggested a two-phase architecture for the mining process providing numerous opportunities for user feedback, control, and approval. More importantly, we develop techniques for pushing the constraints deep inside the mining process to optimize performance so that the work performed by the system is commensurate with the focus expressed by the user via constraints. Towards this end, we have identified the anti-monotonicity and succinctness properties according to which we characterize constraints into various categories. For each category, we have developed specific strategies for finding frequent itemsets that satisfy the given constraints. The resulting algorithm, called CAP (Constrained Apriori), is based on the basic idea of exploiting the given constraints as early as possible in the mining process. We have implemented both CAP and several other basic algorithms that do not use the anti-monotonicity and succinctness properties offered by the constraints. A series of experiments were performed to compare the performance of the various algorithms. We found that CAP clearly outruns all the other algorithms. The speedup of CAP over other algorithms is about 5-10 times on average and more than 80 times in the best case.

To enable the technology of data mining to reach its full potential, many researchers believe that it is important to have a framework where a user can interact with a mining system in much the same way that the user does with a Relational Database Management System today (e.g, [IM96]). We believe that the work presented in this thesis represents an important step in this direction.

7.2 Future Work

While we believe that the work presented in this thesis is significant, due to time and scope limitation, several important issues have been left out for future work. They are discussed below.

(1) **Phase II:** The entire thesis has been focused on Phase I of finding frequent itemsets. The implementation of Phase II, the pros and cons of different significance metrics in the association relationship, and the corresponding algorithms are not discussed. In fact, all of these issues will be investigated in another thesis work [Phase II].

(2) **2-var Constraints:** Our investigation has been focused on 1-var constraints in this thesis. A similar idea of pushing constraints deep into the computation can also be applied to 2-var constraints. We have already carried out a detailed analysis of 2-var constraints. Details will be found in a forthcoming paper.

(3) **Segment Support:** Next, we discuss an optimization technique which is not specific to CAQ mining, but general enough that it can be applied to classical association queries and its variants.

The technique involves dividing transaction databases into segments. Given a transaction database $\mathcal{D}$, we partition it into M segments, denoted by $\mathcal{D}_i (i = 1, \ldots, M)$, in some arbitrary way. Then, instead of using just one integer for the support of a candidate set, we use M integers where each integer corresponds to the support of the set over one transaction database segment. We refer to the M integers as the *segment supports*:

Definition 7.1 (Segment Support) Given a transaction database $\mathcal{D}$ and a partition $\{\mathcal{D}_i, i = 1..M\}$ of the database (*i.e.* $\mathcal{D} = \cup_{i=1}^{M} \mathcal{D}_i$, $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset$ for $i \neq j$), the *ith segment support* of any set S is defined to be the support S over $\mathcal{D}_i$ and is denoted

S	P_1	P_2	P
{A,B}	100	200	300
{A,C}	200	200	400
{B,C}	200	100	300
{A,B,C}	100	100	200

Table 7.1: An example to illustrate the idea of segment support

by $Support_i(S)$.

The sum of the segment supports gives the *total support* which is the support in usual sense. We note that segment support satisfies the same anti-monotone property of the total support. In other words, $\forall S, L, \quad S \subset L \Rightarrow Support_i(S) \geq Support_i(L)$.

The motivation of introducing segment support is best illustrated with an example.

Consider the following simple example where the number of segments is two (*i.e.* $M = 2$) and the support threshold is 250. The first three rows in Table 7.1 show the segment support for the three set $\{A, B\}, \{A, C\}$, and $\{B, C\}$. The last column shows the total support of the three sets. Since all (total) supports are larger than 100, the support threshold, all the sets, $\{A, B\}, \{A, C\}$, and $\{B, C\}$, are size two frequent sets. Now, consider the generation of size three candidate sets. Without the information provided by segment support, the set $\{A, B, C\}$ would be a valid candidate set in C_3. However, the maximum possible segment support for the set $\{A, B, C\}$ in each segment has to be equal to the minimum among the segment supports over the same segment of its subsets. The reason for that is because of the anti-monotone property of the segment support discussed above. In segment one, the minimum of the segment supports among $\{A, B\}, \{A, C\}$, and $\{B, C\}$ is 100. Therefore, the segment support of $\{A, B, C\}$ can be at most 100. Similarly,

the maximum segment support of $\{A, B, C\}$ in segment two is 100. Thus, the total support of $\{A, B, C\}$ must be equal or less than 200. This implies the set $\{A, B, C\}$ cannot be frequent and can be pruned away from C_3 *before* counting for support.

The above example illustrates that we can prune away many potential candidate sets in the generation of C_{k+1} from L_k. The power of this pruning power is based on the following inequality.

For any set $S \in \mathcal{P}(\texttt{Item})$,

$$Support(S) \leq \sum_{j=1}^{M} \min_{S_i \subset S} \{Support_j(S_i)\}, \tag{7.1}$$

where $Support(S)$ means the total support of S and the minimum operator runs over all the S's subsets. The right hand side of the above equation can be taken as the *estimated support* for the set S.

To prove the above inequality, we note that the total support of S is equal to the sum of the support of S over each database segment D_i, *i.e.* $Support(S) = \sum_{j=1}^{M} Support_j(S)$. However, as a support function, we have $Support_j(S) \leq Support_j(S_i)$, $\forall S_i \subset S$, and thus, $Support_j(S) \leq \min_{S_i \subset S}\{Support_j(S_i)\}$ which leads to the above inequality.

The pruning power of the segment supports increases with the number of segments. In fact, if one goes to the extreme case where the number of segment is the number of transactions, the estimated support will be the *actual* support. The actual choice of the number of segments represents a tradeoff among computer resources. If there is enough memory resources, one would like to use a larger number of segments because the pruning power provided by the segment supports is higher. With fewer candidate sets, CPU costs (and related I/O costs) are reduced. On the other hand, if there is not enough memory, a large number of segment supports will consume up memory that could have been used for other purposes such as holding transaction data. This will increase the I/O costs.

Experimental investigation of the implication of the segment support is underway. Applications of the above idea to real life data should be an interesting future work.

(4) **Jumping Levels:** Finally, another direction for future work is based on the observation that for each frequent set found at the end of the Apriori$^+$ Algorithm, every one of its subsets has been counted. For example, in order to find the frequent itemset $\{A, B, C, D\}$, Apriori$^+$ needs to count support for *every* one of the sets $\{A\}, \{B\}$, $\{C\}, \{D\}$, $\{A, B\}$, $\{A, C\}$, $\{A, D\}, \{B, C\}$, $\{B, D\}, \{C, D\}$, $\{A, B, C\}, \{A, B, D\}$, $\{A, C, D\}$, and $\{B, C, D\}$. However, if somehow we count $\{A, B, C, D\}$ *before* counting its subsets and realize that it is frequent, then we can *a posteriori* prune away all its subsets from counting because we know all its subsets would be frequent. This leads to the idea of *jumping level.* In other words, Apriori$^+$ is a level-by-level algorithm and the pruning of candidate sets is based on our knowledge on lower levels. We can prune away a candidate set if any of one of its proper subsets is not included in the list of frequent sets found in lower levels. However, if we have already counted some higher level to get all the frequent sets of size larger than the current size of the candidate sets, then we can also prune away those candidate sets that are subsets of some frequent sets at higher levels. This is because those candidate sets are guaranteed to be frequent and therefore can be dropped from counting.

Using a scheme of jumping ahead to count higher levels can provide additional pruning power. However, there are still many issues one has to resolve. For example, counting higher levels can pay off only when there are a significant number of frequent sets to be found. If all the candidate sets turn out to be non-frequent, then no information is gained. Moreover, it is not clear which level to "jump". Intuitively, if there are still a significant number of frequent sets, one would like to jump to higher levels as the chance of payoff is higher. However, more analysis is required to make use of the above idea and to come up with an efficient algorithm.

Bibliography

[AIS93] R. Agrawal, T. Imielinski and A. Swami, Mining association rules between sets of items in large databases, *SIGMOD 93*, p. 207-216.

[ALSS95] R. Agrawal, K. Lin, H. S. Sawhney, K. Shim, Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases, *VLDB 95.*

[AMS97] K. Ali, S. Manganaris, and R. Srikant, Partial Classification using Association Rules, *American Association for Artifical Intelligence 1997.*

[AS94] R. Agrawal and R. Srikant, Fast algorithms for mining association rules, *VLDB 94*, p. 487-499.

[AS95] R. Agrawal and R. Srikant, Mining Sequential patterns, *Proc. 1995 International Conference on Data Engineering*, March 1995.

[AS96] R. Agrawal and J. C. Shafer, Parallel mining of association rules: Design, implementation, and experience, *IEEE TKDE*, 8, p. 962-969, 1996.

[BMS97] S. Brin, R. Motwani, and C. Silverstein, Beyond market basket: Generalizing association rules to correlation, *SIGMOD 97*, p. 265-276.

[BMUT97] S. Brin, R. Motwani, J. Ullman, and S. Tsur, Dynamic itemset counting and implication rules for market basket data, *SIGMOD 97*, p. 255-264.

[CHNW96] D. W. Cheung, Jiawei Han, V. T. Ng, and C. Y. Wong, Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique, *Proceedings of the International Conference on Data Engineering*, Feb. 1996.

[FMMT96] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama, Mining Optimized Association Rules for Numeric Attributes, *PODS 96*, p. 182-191.

[HKK97] E.-H. Han, G. Karypis, and V. Kumar, Scalable Parallel Data Mining for Association Rules, *SIGMOD 97*, p. 277-288.

[HF95] J. Han and Y. Fu, Discovery of multiple-level association rules from large databases, *VLDB 95*, p. 420-431.

[IM96] T. Imielinski and H. Mannila, A database perspective on knowledge discovery, *Communications of ACM*, 1996, p. 58-64.

[KN97] E. Knorr and Raymond T. Ng, A Unified Notion of Outliers: Properties and Computation, *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, 1997,* p. 219-222.

[MAR96] M. Mehta, R. Agrawal and J. Rissanen, SLIQ: A Fast Scalable Classifier for Data Mining, Proc. of the Fifth International Conference on Extending Database Technology, March 1996.

[Meta Group] Mining Your Own Business, Information Week cover story, March 16, 1998.

[MPC96] R. Meo, G. Psaila, and S. Ceri, A new SQL-like operator for mining association rules, *VLDB 96*, p. 122-133.

[MY97] R. J. Miller and Y. Yang, Association rules over interval data, *SIGMOD 97*, p. 452-461.

[NH94] R. T. Ng and Jiawei Han, Efficient and Effective Clustering Methods for Spatial Data Mining, *VLDB 94*, p. 144-155.

[NLHP98] R. T. Ng, L. V. Lakshmanan, J. Han, and A. Pang, Exploratory Mining and Pruning Optimizations of Constrained Associations Rules, *SIGMOD 98*, p. 13-29.

[PCY95] J. Park, M. Chen, and P. Yu, An Effective Hash-Based Algorithm for Mining Association Rules, *SIGMOD 95*, p. 175-186.

[Phase II] Teresa Mah, Master of Science Thesis, Computer Science, University of British Columbia, 1999.

[QUEST] R. Agrawal, A. Arning, T. Bollinger, M. Mehta, J. Shafer, R. Srikant, The Quest Data Mining System, *KDD 96*.

[SA95] R. Srikant and R. Agrawal, Mining generalized association rules, *VLDB 95*, p. 407-419.

[SA96] R. Srikant and R. Agrawal, Mining quantitative association rules in large relational tables, *SIGMOD 96*, p. 1-12.

[SVA97] R. Srikant, Q. Vu, and R. Agrawal, Mining association rules with item constraints, *KDD 97*, p. 67-73.

[TUACMN97] D. Tsur, J. Ullman, S. Abitboul, C. Clifton, R. Motwani, and S. Nestorov, Query flocks: A generalization of association rule mining, *SIGMOD 98*, P. 1-12.