INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

DATA MINING WITH BILATTICES

XIAOHONG WANG

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

JANUARY 2001

Abstract

Data Mining with Bilattices

Xiaohong Wang

Data mining has become a key research area in database after Agrawal and Sirkant introduced association rules in data mining and proposed Apriori for mining association. However, most of the works focuses on finding patterns on itemsets, especially associations between items. With the fast development of high technologies and large scale of information collection tools, the need for data mining has gone far beyond association mining. In this thesis, a new framework is established that largely extends the current existing data mining field. While the traditional data mining problems can be viewed as computing itemset lattice, yet another equal important data mining problem is mining circumstance which forms a second lattice: circumstance lattice. In itemset lattice, extensive work is done to introduce constraints in correlations mining and algorithms for computing different useful correlation queries are provided. For the new concept of circumstance mining, a close relationship is set up between computing circumstance lattice and data cube. Several algorithms, including new algorithms and modification of existing data cube algorithms, are given to answer circumstance mining queries. Finally, algorithms for the dual mining on bilattices–from itemset lattice to circumstance lattice and back, or vise versa are presented to compute the Armstrong Basis for a given transaction database.

Acknowledgments

Great thanks must go to my supervisors, Dr. Laks Lakshmanan and Dr. Gösta Grahne, for their knowledgeable input and guidance throughout the duration of this research. It is them who had patiently lead me into this area.

I would like to thank my parents, my brother and my husband. They all have always encouraged me, believed in me, and supported me when I needed it. Especially my husband, who just want to have a housewife, but end up supporting me to get two master degrees. Special thanks go to my grandma. Without her great care, there would be no me in this world. I dedicate this thesis to her.

I would also like to thank all my friends, who have endured me over years. Especially Minghao Xie, who has helped me to implement algorithms for mining circumstances. Without him, my work can not be finished so soon.

Contents

List of Figures vii

List of Tables ix

1 Introduction 1

1.1 Introduction to Data Mining 1

1.2 Current State of Research on Data Mining 3

1.3 Current State of Research on Datacubes 5

1.4 Structure and Contributions of the Thesis 5

2 Data Mining on Itemset Lattice, Circumstance Lattice and Bilattice 9

2.1 Problem Definition 9

2.2 Structure of the Bilattice 11

2.3 Properties of Bilattice and General Pruning Strategy 15

3 Data Mining on Itemset Lattice 17

3.1 Introduction and Motivation of the Problem 17

3.2 Constrained Correlations 20

3.2.1 Correlation queries 20

3.2.2 Adding constraints 21

3.3 Algorithms for Constrained Correlations 25

3.3.1 Computing valid minimal answers 26

3.3.2 Computing minimal valid answers 28

3.3.3 Analysis of the algorithms 31

3.4 Experiments 32

3.4.1 Test data 32

3.4.2 Experimental evaluation 33
3.4.3 Summary and conclusions 42
3.5 Related Work 42

4 Data Mining on Circumstance Lattice **45**
4.1 Motivation and Background 45
4.1.1 Motivation 45
4.1.2 Theoretical background and relations with datacube 47
4.2 A Special Circumstance – Time 48
4.3 Algorithms for General Circumstance Mining 49
4.3.1 Relevance to CUBE 53
4.4 Experimental Results 55
4.4.1 Test data 55
4.4.2 Experimental evaluation 56
4.4.3 Summary and conclusions 63
4.5 Related Work 63

5 Data Mining on Bilattice **64**
5.1 Introduction and Motivation 64
5.2 Theory of Armstrong Basis 65
5.3 Algorithms for Computing Armstrong Basis 66
5.4 Summary and related Work 71

6 Conclusions and Future Work **73**
6.1 Conclusions 73
6.2 Future Work 75

Bibliography **76**

List of Figures

1 An Example Itemsets Lattice 12
2 An Example Circumstance Lattice 13
3 An Scratch View of Mining on Both Lattices 14
4 The Borders of Correlation and CT-support in the Itemset Lattice . . 19
5 The Valid Minimal Solutions and Minimal Valid Solutions. 24
6 Algorithm BMS. 26
7 Algorithm BMS+. 27
8 SIG and NOTSIG for Algorithm BMS++. 28
9 Algorithm BMS*. 29
10 SIG and NOTSIG for Algorithm BMS**. 30
11 The Interplay Between Constraints and Correlation 31
12 Effect of Transaction Number with Anti-monotone & Succinct Constraints 34
13 Effect of Selectivity with Anti-monotone & Succinct Constraints . . . 35
14 Effect of Transaction Number with Anti-monotone Constraints 36
15 Effect of Selectivity with Anti-monotone Constraints 38
16 Effect of Transaction Number with Monotone and Succinct Constraints for Valid Minimal Answers 39
17 Effect of Selectivity with Monotone and Succinct Constraints for Valid Minimal Answers 40
18 Effect of Transaction Number with Monotone and Succinct Constraints for Minimal Valid Answers 41
19 Effect of Selectivity with Monotone and Succinct Constraints for Minimal Valid Answers 43
20 Pseudo Code for Algorithm for Finding Minimal Intervals Where an Itemset is Frequent. 50

21 Adaptation to Chunk Array Algorithm for Finding Minimal Circumstances. 51
22 Algorithm BottomUpCube(BUC). 52
23 Adaptation to BUC Algorithm for Finding Minimal Circumstances. . 52
24 A Linked List Based Mining Algorithm for Finding Minimal Circumstances. 54
25 Algorithms Comparison ($memoryresident$, $skewness = 1$, $threshold = 100$, $No.ofCirc. = 5$) 57
26 Algorithms Comparison ($diskresident$, $skewness = 0$, $threshold = 100$, $No.ofCirc. = 5$) 58
27 Algorithms Comparison ($diskresident$, $skewness = 1$, $threshold = 100$, $No.ofCirc. = 5$) 58
28 Algorithms Comparison ($diskresident$, $skewness = 1$, $threshold = 100$, $No.ofCirc. = 6$) 59
29 Algorithms Comparison ($diskresident$, $skewness = 1$, $threshold = 100$, $No.ofTrans. = 100000$) 60
30 Algorithms Comparison ($diskresident$, $No.ofCirc. = 5$, $threshold = 100$, $No.ofTrans = 100000$) 61
31 Algorithms Comparison ($memoryresident$, $No.ofCirc. = 5$, $threshold = 100$, $No.ofTrans = 100000$) 61
32 Algorithms Comparison ($memoryresident$, $skewness = 1$, $threshold = 1000$, $No.ofCirc. = 5$) 62
33 Algorithms Comparison ($memoryresident$, $skewness = 1$, $threshold = 5$, $No.ofCirc. = 5$) 62
34 Decision Tree for Choosing a Mining Algorithm. 63
35 Algorithm for Computing the Armstrong Basis of a Pattern in a Transaction Database 67
36 An Example FP-tree (support threshold assumed to be 2) 68
37 An Example Circumstance Prefix Tree with Leaves Pointing to Itemset FP-trees 71

List of Tables

1 Example Contingency Table. 20

Chapter 1

Introduction

With the opening of the new millenium, our needs for information processing and analysis have become a basic necessary. As I was writing this thesis, one article from September 10th's "The Detroit News" came cross my eyes. The first sentence from this article was: "When Bill Clinton and his staffers vacate the white House in January, they'll leave behind a historic mess: eight years' worth of memos, snapshots and bureaucratic bric-a-brac that must be preserved for posterity by the National Archives, the government's official record keeper. And oh yes, there is also the little matter of 40 million e-mails." While people believe that this historic mess could be a gold mine for future historians, just imaging the difficulties to store all of this, let alone to mine the useful information from this mess, if it is not impossible.

This simple news raises a very important issue, called **data mining**. Data mining, also known as knowledge discovery in database, could be defined as a broad range of activities that attempt to discover new information from existing information, where usually the original information is gathered for a purpose entirely different from the way it is used for data mining. With the coming of the digital age, data mining is becoming more and more important.

1.1 Introduction to Data Mining

Data mining has been introduced as a new area in database research for more than 7 years. Before the introducing of data mining, massive amounts of data were largely left unexplored and either stored primarily in an off-line store or on the verge of being

thrown away. This is because normal database systems offer little functionality to support such "mining" applications. On the other hand, machine learning techniques which do have such functionality, usually perform poorly when applied to very large data sets. With the wide applications of computers and automated data collection tools, people start to realize the importance of data mining. The basic problem is: how could people find useful information from massive amounts of data collected and stored in database. Typically, these applications involve large-scale information banks such as data warehouses or datacubes.

One of the fundamental operations behind the common data mining tasks is to *find patterns* in database. The first introduced pattern in data mining is associations [AIS93]. The discovery of interested association relationships among huge amounts of data will help marketing, decision making, and business management. Since then several other patterns including correlations, causality, sequential patterns, episodes, multidimensional patterns, max-pattern, partial periodicity, and emerging pattern are also been introduced.

Over these years, researchers have studied various problems related to mining interested patterns from large databases. They developed faster (both sequential and parallel) algorithms for associations, its quantitative variants, sequential patterns, extensions, and generalizations ([AS94, Klem+94, HKK97, SBMU98, KLKF98] are some representative works).

Most of the work has emphasized on techniques for improving the performance of algorithms for association rules in large database. Numerous algorithms have been proposed for association rule mining. Among these, one particularly well-studied problem is the search for association rules in market basket data. In this setting, the base information consists of register transactions of retail stores. The query to be answered is "which items are bought together often?"

In 1997, Brin et al [BMS97] proposed correlations, which generalized beyond market baskets and associations used with them. In their setting, association rules were but one of the many types of recurring patterns that could or should be identified by data mining. Queries now can be extended to not only items that are bought together often, but any items that have dependency with each other. So the query to be answered becomes "which items are related to each other?". The first part of this thesis is based on correlations.

However, a more interesting query could be "under what conditions or circumstances a certain pattern holds?" An obviously example is: customers will buy lots of ice creams and cold drinks in a hot day, while the sale for these items will drop dramatically in a freezing day. So it is not only important to find out the patterns, but also the *circumstances* under which these patterns hold. This thesis introduces the new concept of circumstance mining, and establishes the relationship between itemset mining and circumstance mining.

1.2 Current State of Research on Data Mining

In 1994, Agrawal and Srikant [AS94] proposed the concept of association rule as the following:
We say that $i_1 \Rightarrow i_2$ if

1 i_1 and i_2 occur together in at least s % of the n baskets (the support);

2 and, of all the baskets containing i_1 , at least c % also contain i_2 (the confidence).

They also proposed an algorithm, called Apriori, to discover all significant association rules between items in a large database of transaction. They compared Apriori with two previously known algorithms, AIS and SETM. Experimental results showed that Apriori always outperformed the previous algorithms. The performance gap increases with the transaction size, and ranges from a factor of three for the small problems to more than an order of magnitude for large transaction.

Apriori achieves good performance by possibly significantly reducing the size of candidate sets. However, in situations with prolific frequent patterns, long patterns or quite low minimum support thresholds, it still suffers. Because the cost of Apriori is exponential to the number of items, and it is tedious to repeatedly scan database and check a large set of candidates by pattern matching. This is especially true for mining long pattern. So Han *et al* [HPY00, PHM00] proposed a new algorithm using frequent pattern tree (FP-tree) for mining frequent patterns without candidate generation. The efficiency of their algorithm, *FP-growth*, is achieved with three techniques: (1) a large database is compressed into a highly condensed, much smaller data structure to avoid costly, repeated database scans; (2) FP-growth adopts a pattern fragment

growth method to avoid the costly generation of a large number of candidate sets; (3) a partitioning-based, divide-and-conquer method is used to decompose the mining task into a set of smaller tasks for mining confined patterns in conditional databases, which dramatically reduces the search space. The performance study showed that FP-growth method was efficient and scalable for mining both long and short frequent patterns, and was about 10 times faster than Apriori.

Bayardo [Bay98] showed a different approach to solve the same problem. He proposed an algorithm, called *Max-Miner*, which scaled roughly linearly in the number of maximal patterns embedded in a database irrespective of the length of the longest pattern, while Apriori based algorithms scaled exponentially with the longest pattern length. Experiments on real data showed at least 10 times faster than the Apriori based algorithms.

Yet Ng *et al* [NLHP98] noticed that from the standpoint of supporting human-centered discovery of knowledge, all the above models of mining association rules suffer from the following serious shortcomings: (1)lack of user exploration and control, (2) lack of focus, and (3) rigid notion of relationships. In effect, these models function as a black-box, and admit little user interaction in between. So they proposed an architecture that opened up the black-box, and supported constraint-based, human-centered exploratory mining of associations. The foundation of this architecture is a rich set of constraint constructs, including domain, class, and SQL-style aggregate constraints, which enable users to clearly specify what associations are to be mined. They also proposed an algorithm called CAP. The introducing of constraints into mining, not only solves the above shortcomings, but also restricts the number of candidate sets needed to be generated. So experimental performance shows great improvement on several previous algorithms.

While lots of researchers working on association mining, others realize that association is not the only rule needed to be mined from large database. The need for mining queries beyond association may give even more interesting results. So Brin *et al* [BMS97] introduced correlation mining using chi-squared metrics. However, either association or correlation infers with a statistical relationship such as confidence and chi-squared test between items, to better understand the nature of the relation between items, Silverstein *et al* [SBMU98] introduced causality in 1998.

1.3 Current State of Research on Datacubes

By the definition of data mining, the applications need to perform data mining normally involve with datacubes and they are closely related to On-Line Analytical Processing (OLAP). This is another database research area that attracts a lot of attention recently. As the basic operation of datacubes, CUBE operator generalizes the standard group-by operator to compute aggregates for every combination of group-by attributes. This results in huge computation and memory requirements. So researchers also did a lot of work on improving the performance of CUBE operator to make the computation as efficient as possible.

In 1999, Beyer and Ramakrishnan [BR99] proposed an algorithm called BUC for cube operator. BUC avoids computation of larger group-bys that do not meet the minimum support. The pruning in BUC is similar to the pruning in Apriori for associations, except that BUC trades some pruning for locality of reference and reduces memory requirements. The pruning in BUC, combined with an efficient sort method, enables BUC to outperform most of the previous algorithms for sparse CUBEs, and dramatically improves another type of cube computation, called Iceberg-CUBE. The Iceberg-CUBE problem is to compute all group-by partitions for every combination of the grouping attributes that satisfy an aggregate selection condition, as in the HAVING clause of an SQL query. BUC works actually quite well even for unconstrained cubes.

Zhao et al [ZDN97] proposed another approach to compute CUBE using chunked array. They used minimum memory spanning tree to reduce the memory requirement. Their method overlaps the computation of different group-bys, and by optimizing the dimension order the total memory requirement is minimized. Experimental results show great performance.

1.4 Structure and Contributions of the Thesis

As mentioned at the beginning of the introduction, associations answer queries such as which items are bought together often. Correlations extend the queries to which items are related to each other. Actually, all such previous work on data mining has been done on one lattice, called *itemset lattice*. Yet an even more interesting query should be under what *circumstances* (conditions) certain items have a special pattern.

These conditions or circumstances form a new lattice called *circumstance lattice*. In Chapter 2, the concept of itemset lattice, circumstance lattice and the relationship between these two lattices are introduced. [1]

Although there has been an extensive amount of work done on mining associations, people have so far realized that associations are not appropriate for all situations (for example, see [BMS97]). Alternate patterns/rules need to be explored as well. Some examples are given in the previous section. Same as introducing constraints in associations, applications for mining minimal correlated sets satisfying given constraints should also be considered. In Chapter 3, an extensive work on correlation mining is proposed. Brin *et al*'s [BMS97] algorithm for mining correlations has been modified to cooperate with constraints. Several different algorithms are proposed for different type of constraints and experimental evaluations are presented. The performances are greatly improved.

The new concept of circumstance mining and its close relationships with datacube are discussed in Chapter 4. Two existing algorithms for CUBE operation are modified for circumstance mining and a new algorithm is proposed. Experimental comparison is conducted to summarize the conditions under which a certain algorithm should be chosen.

In general, mining is exploratory in nature. An analyst might wish to find the circumstances under which a pattern holds for an itemset and then find out which other itemsets satisfy the pattern under those circumstances. Chapter 5 presents data mining on bilattice. In this chapter, a new notion of *Armstrong Basis* is proposed as a basis for supporting a variety of the above queries. A couple of algorithms are presented to compute the Armstrong Basis.

In this thesis, the concept of bilattice is defined, constrained correlations for itemset lattice are then studied, and the circumstances under which a given pattern (or set of patterns) holds in a large database in the circumstance lattice are then determined. Finally, algorithms for computing Armstrong Basis – a bilattice computation are provided. The following is the contributions made in this thesis:

1. Formalize the notions item, transaction, and circumstance (Chapter 2). This abstraction helps make the results useful for a wide variety of applications.

[1] Briefly speaking, each transaction can contain a set of items, but can only contain a unique value for each circumstance. For more detail, see Chapter 2

2. For constrained correlation queries, in general, the answer set (i) valid minimal correlated and CT-supported itemsets is a proper subset of (ii) minimal valid correlated and CT-supported itemsets, they coincide whenever all constraints in the user query are anti-monotone. Techniques are developed for computing either of the answer sets above and a series of experiments are conducted to validate the analysis.

3. For many applications, the space of circumstances forms a lattice, which is called the *circumstance lattice* (Chapter 2). This lattice generalizes the cube lattice [G*96]. (See also [BR99]. for example). So a close relationship is established between the problem of finding all circumstances under which a given pattern holds, and data cube computation (Chapter 4). Given the similarity to cube, adaptations of two representative popular algorithms for cube computation are made for circumstance mining, including BUC ([BR99]) and chunked array cube ([ZDN97]); For patterns satisfying either the monotone or the anti-monotone property, an algorithm, called *minCirc*, is developed to determine the strongest (resp., the weakest) circumstances under which they hold (Chapter 4).

4. For queries going beyond merely finding circumstances under which a pattern holds, a notion called *Armstrong Basis* is proposed and we have showed that a variety of useful queries can be answered from this *Armstrong Basis* (Chapter 5). The motivation for the Armstrong Basis is similar to that of cube. While one does not expect a user to necessarily want a complete cube, the utility of the cube comes from its ability to quickly service a whole class of queries involving (possibly multiple) group-bys. In a similar manner, Armstrong Basis can be huge and is unlikely to be interested in its entirety. Yet, given its utility, it is useful to devise algorithms for its efficient computation. The last contribution of this thesis is a couple of algorithms for this purpose (Chapter 5).

Selected publications related to this thesis: Before closing this chapter, the selected publications from this thesis are listed below.

1. G. Grahne, L. V. S. Lakshmanan, Xiaohong Wang, Minghao Xie. On Dual Mining: From Patterns to Circumstances, and Back, accepted by ICDE 2001.

2. G. Grahne, L. V. S. Lakshmanan, Xiaohong Wang. Efficient Mining of Constrained Correlated Sets. ICDE 2000, pages 512-521 San Diego, CA, 2000.

3. G. Grahne, L. V. S. Lakshmanan, Xiaohong Wang. Interactive Mining of Correlations - A Constraints Perspective. ACM SIGMOD workshop on research issues in data mining and knowledge discovery, pages 7-1, Philadelphia, 1999.

Chapter 2

Data Mining on Itemset Lattice, Circumstance Lattice and Bilattice

This chapter proposes the concept of bilattice on data mining. Many data mining problems could be generally viewed as computing lattice. The traditional data mining such as associations and correlations is based on itemsets, which form a lattice, called *itemsets lattice*. A new issue of data mining is discussed in this chapter which is based on circumstance instead of itemset. This new problem of mining circumstance can form another lattice called *circumstance lattice*. Thus data mining becomes mining information on either lattices or on the bilattice.

2.1 Problem Definition

It is well known that traditional framework of data mining regards each attribute-value pair as an "item" and attempts to discover patterns such as associations between sets of items using measures such as support and confidence. However, given a set of transactions, not all attributes are alike. Some attributes may have a unique value per transaction while others may have multiple values in a given transaction. As an example, items bought from a supermarket naturally correspond to an attribute that has a set of values per transaction. On the other hand, the timestamp associated with a transaction typically has a unique value per transaction. While treating all attributes in the same way is certainly possible, we contend that it misses out on the opportunities of exploring and exploiting the different structures and properties

associated with dissimilar attributes. For example, for an attribute such as time, different values are mutually exclusive in that a transaction can only have one value for it. By distinguishing among such attributes, people can formulate queries like "what are the intervals during which there is a significant volume of sales on dessert items". As another example, user can ask "which are the locations where there is a strong association among a set of items being purchased". Mining queries of this form can be best facilitated by developing a framework where attributes that are inherently like items (They are called "item attributes" below), i.e. have a set of values per transaction, are distinguished from those that are more like time (They are called "circumstance attributes" below), which have a unique value per transaction. If we have the following transaction:

1: Sept. 20, sunny, Detroit, milk $2.99, pork $3.98, biscuits $1.59

Obviously, Sept. 20, sunny, and Detroit are "circumstance attributes", while milk, pork, biscuits are "item attributes". The potential "item attributes" need further to be distinguished from "descriptive attributes," such as price or quantity. The above considerations motivate some definitions.

Consider a table over a set of attributes, tuples, or possibly sets of tuples, in this table can be viewed as transactions. Here, it is possible to designate any set of attributes in the table as a transaction id, as long as this attribute set uniquely determines the transaction. For example, the set {`customerID`, `timeID`} might constitute the transaction id for the transactions of a table containing a supermarket sales data. As another example, {`locationID`, `prodID`, `timeID`} might serve as a transaction id for tuples in a fact table of a data warehouse containing sales data. In general, given a table r over a set of attributes R, users can specifies a subset of attributes, say $K \subset R$, as its *transaction id*. This induces a partition on the table with tuples sharing the same K-value being in the same cell of the partition. [1] An attribute $A \in R \setminus K$ is called *circumstance attribute* provided the functional dependency $K \rightarrow A$ holds for r. For other attributes $A \in R \setminus K$, it follows from the axioms of functional and multivalued dependencies that the multivalued dependency $K \twoheadrightarrow Y$, where $A \in Y$, will hold in r. If in addition the functional dependency $KA \rightarrow Y \setminus A$ holds, A is an *item attribute*, otherwise A is a *descriptive attribute*.

[1] Each cell might be a singleton set.

Consider a table sales(tid, location, time, product, quantity) with tid acting as the transaction id, in the sense that each tid-value uniquely determines a set of tuples, in which a set of products is sold at a given time and location. We also have {tid}$\twoheadrightarrow${product, quantity}, and {tid, product} $\rightarrow$ {quantity}. Thus product is an item attribute and quantity is a descriptive attribute, while location and time are circumstance attributes. In the constrained data mining framework of [LNHP99, GLW00], constraints would typically be formulated from circumstance and descriptive attributes.

An *itemset* is then a set of attribute-value pairs, where the attribute is an item attribute and the value comes from its domain. A *circumstance*, indicates a conjunction of predicates of the form *attr*θ*value*, where *attr* is a circumstance attribute, and *value* comes from its domain. Here, typically, θ is set to be equality. When dealing with totally ordered circumstance attributes (for example, time, distance, etc.), θ is allowed to be $\leq$. since on such domains, intervals over which patterns hold are of interest. A transaction *satisfies* a circumstance ψ, provided it satisfies the predicate ψ.

Given a support threshold s, [2] an itemset S is *frequent* in a transaction database provided at least s transactions contain S. More generally, given an itemset S and a circumstance ψ, S is frequent under the circumstance ψ provided s transactions among those satisfying the circumstance ψ contain S.[3] The traditional notions of support and confidence in association mining can be relativized to circumstances: the *support* of S under ψ is the number of transactions satisfying ψ that contain S; for itemsets S and T, the *confidence* of a rule $S \Rightarrow T$ under ψ is the proportion, among the transactions satisfying ψ, the number of transactions containing $S \cup T$ over those that contain S. Other patterns in data mining could be reformulated in a similar way.

2.2 Structure of the Bilattice

For a given set of transactions, the domain of items forms a lattice called *itemset lattice*. The traditional data mining is to search relationships between items in this lattice. An example of itemsets lattice is showed at Figure 1.

[2]Recall, an absolute count number is used.

[3]Thus, if fewer than s transactions satisfy ψ, S is infrequent under ψ.

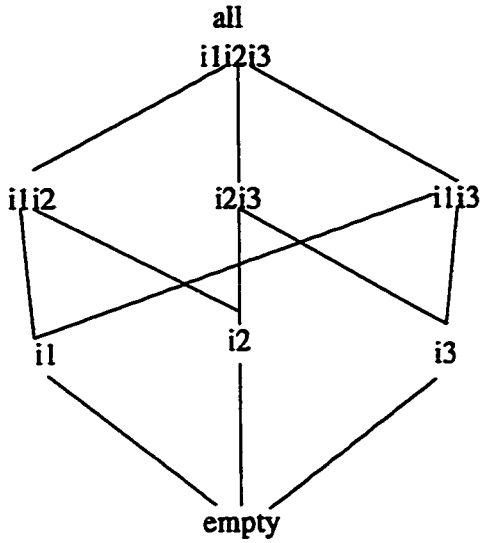

Figure 1: An Example Itemsets Lattice

For simplicity, there are only three items, namely i_1 and i_2 , and i_3. The bottom of the lattice is the *empty* itemset, and the top of the lattice is the *all* itemset.

It is quite common that there is a taxonomy (*is-a* hierarchy) on the items. This taxonomy normally comes from the *descriptive attribute*. As an example of a taxonomy, Jacket *is-a* Outerwear, Ski Pants *is-a* Outerwear, and Outerwear *is-a* Cloths, *etc.* So it is easy to extend the concept of traditional data mining into different level of taxonomy hierarchy. A rule may be held at the higher level of taxonomy, but does not hold at the lower level of taxonomy. A couple of papers have studied this problem [AS95, HaFu95].

Meanwhile, the space of circumstances forms another lattice, called the *circumstance lattice*. To see this, recall that circumstances are conditions comparing attributes with values. Define a partial order $\leq$ on the set of all circumstances Ψ, by setting $\psi \leq \phi$ exactly when ψ logically implies ϕ. It is well-known that a collection of sentences ordered this way forms a lattice. In particular, *true* is the top element and *false* is the bottom element. Figure 2 shows an example circumstance lattice for two circumstance attributes A, B each with two values in its domain. Suppose A represents *location* and B represents *time*, and $a1$ is *Detroit*, $a2$ is *Montreal*. Similarly, $b1$ is *morning*, $b2$ is *afternoon*. Then we have the combination of the following circumstances: { Detroit }, {Montreal }, { morning }, {afternoon }, { Detroit, morning }, { Detroit, afternoon }, etc. For simplicity, the figure only considers circumstances

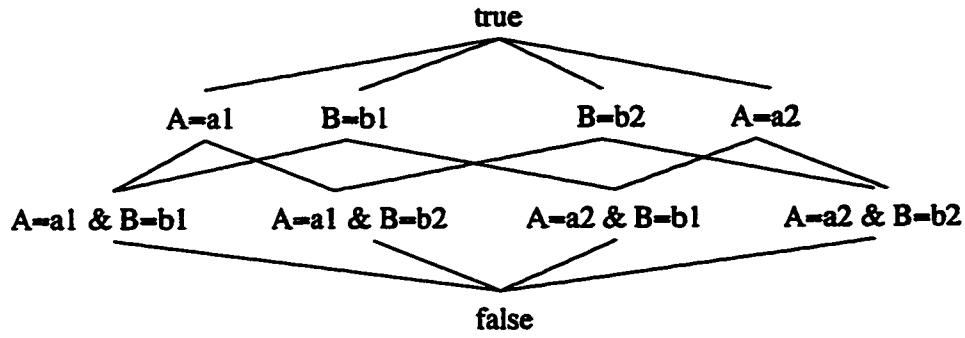

Figure 2: An Example Circumstance Lattice

involving equality conditions.

Circumstance attributes sometimes come with Hierarchies as well. For example, locations may have the hierarchy `storeId-city-state`. It is a straightforward matter to see that the notion of lattice extends to circumstance attributes with hierarchies, as well as to circumstances involving inequalities. However, for simplicity, in the rest of the thesis, unless otherwise specified, circumstances involving equality conditions and no attribute hierarchies are considered.

With the introducing of both lattices, mining queries such as "under what circumstance beer's sale is high, and what else also has a high sale?" could be answered by mining both lattices back and forth. Figure 3 gives a scratch view of mining in both lattices. From a given point in the circumstance lattice to mine itemset lattice, first filter out all the transactions that satisfy the given circumstance, then use the traditional mining algorithms such as Apriori to find the appropriate itemsets. For example, user may want to find the associations for a market basket in { Montreal, afternoon }, and get one of the result rules as { beer, diaper }. From a given point in the itemset lattice to mine circumstance lattice, first filter out all the transactions that contain the given itemset, then use the algorithms provided in Chapter 4 to mining circumstance. [4] For example, user may want to find out under what circumstances { beer, diaper } is bought frequent, and actually get results as { Montreal, afternoon } and { Detroit, afternoon }.

Let $(\mathcal{I}, \subseteq)$ denote the itemset lattice for a given application and $(\mathcal{C}, \leq)$ denote the circumstance lattice w.r.t. the circumstances pertaining to the application. A new structure $\mathcal{C} \times \mathcal{I}$ can be constructed by taking the direct product of the two

[4] Most of the datacube algorithms can be modified to do circumstance mining, for details see Chapter 4.

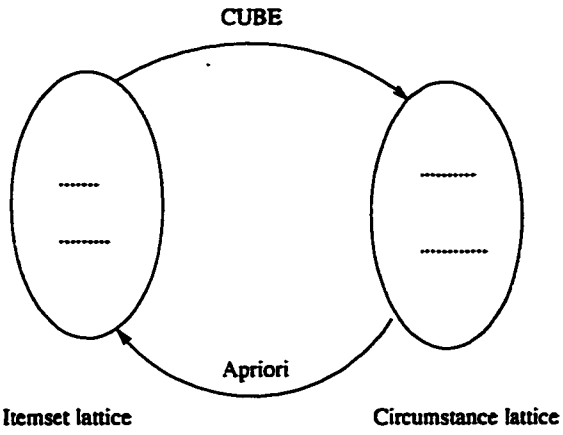

Figure 3: An Scratch View of Mining on Both Lattices

lattices. On this structure, two alternative orders can be defined: the first order $\leq^{\text{up}}_{\text{up}}$ is defined as $(\psi, S) \leq^{\text{up}}_{\text{up}} (\psi', S')$ iff $\psi \leq \psi'$ and $S \subseteq S'$; the second order $\leq^{\text{up}}_{\text{down}}$ is defined as $(\psi, S) \leq^{\text{up}}_{\text{down}} (\psi', S')$ iff $\psi \leq \psi'$ and $S' \subseteq S$. A structure $(\mathcal{L}, \leq_1, \leq_2)$ with two partial orders is called a *bilattice* [Fit91], provided each of $(\mathcal{L}, \leq_1)$ and $(\mathcal{L}, \leq_2)$ is a lattice, and a certain condition relating the two orders hold. It can be shown that $(\mathcal{C} \times \mathcal{I}, \leq^{\text{up}}_{\text{up}}, \leq^{\text{up}}_{\text{down}})$ is actually a bilattice, which is called the *circumstance-itemset bilattice.*

The bilattice formulation gives a semantic basis for known mining problems and inspires newer ones too. For example, the classical problem of mining frequent itemsets from a given transaction database reduces to finding all itemsets S such that S is frequent under *true*. More generally, to find all frequent sets in a selection view (not involving aggregates) defined by conditions over the transaction database, will become to find the set $\{S \mid S \text{ is an itemset frequent under } \psi\}$, where ψ is the conjunction of conditions defining the view. Let $\mathcal{F}$ denote the subset of the bilattice containing precisely those pairs (ψ, S) such that S is frequent under ψ. Then frequent set mining for a given ψ corresponds to finding $\{S \mid (\psi, S) \in \mathcal{F}\}$. Circumstance mining for a given itemset S corresponds to finding the set $\{\psi \mid (\psi, S) \in \mathcal{F}\}$.

The bilattice structure also allows people to prove some identities which can be helpful in pruning the search space. This issue will be illustrated later in this section. Besides, the bilattice structure is instrumental in shaping the concept of an Armstrong Basis, which is the main object of Chapter 5.

2.3 Properties of Bilattice and General Pruning Strategy

In this section, general patterns involving itemsets are classified according to the properties they satisfy. Since circumstances are also involved, some notations need to be defined first. Let a *pattern* over the bilattice $(\mathcal{C} \times \mathcal{I})$ as any predicate over it, *i.e.* any subset of the elements of the bilattice. For example, frequency corresponds to the collection of elements $(\psi, S) \in (\mathcal{C} \times \mathcal{I})$, such that S is frequent under ψ; Correlation corresponds to the collection of elements (ψ, S) such that S is a correlated set under ψ. As another example, consider the statement *minpurchase*(S) that says each item in itemset S must be bought in sufficient quantity over the set of transactions, the total dollar value of the purchase on items in S must exceed a threshold, or *purchasecap*(S) that says the total dollar value of the purchase on items in S must be below a threshold. Each of these defines a subset of the bilattice. $p_{\mathcal{D}}(\psi, S)$, or simply $p(\psi, S)$ is used as the notation to indicate pattern p holds for the element (ψ, S) in database $\mathcal{D}$. For example, when p is *purchasecap*, this means over those transactions satisfying circumstance ψ, the total dollar value of purchase of items in S does not exceed the given threshold.

A pattern p is *c-monotone*, provided whenever $p(\psi, S)$ holds, $p(\phi, S)$ holds as well, for every $\psi \leq \phi$. It is *c-anti-monotone*, provided whenever $p(\psi, S)$ holds, $p(\phi, S)$ holds as well, for every $\phi \leq \psi$. The notions *i-monotone* and *i-anti-monotone* are defined analogously and correspond exactly to what was referred to as *monotone* and *anti-monotone* in the frequent set mining literature [NLHP98]. In general, a pattern may be monotone or anti-monotone w.r.t. either lattice. For example, frequency is c-monotone and i-anti-monotone. The pattern *purchasecap* is c-anti-monotone and i-anti-monotone, while *minpurchase* is i-monotone and c-monotone. Being correlated as defined in [BMS97] is i-monotone but neither c-monotone nor c-anti-monotone.

Many of these patterns satisfy interesting identities. The following Proposition is proved using the axioms of bilatticies.

Proposition 1 *Let p be any pattern. Then we have the following:*

1. *if p is i-anti-monotone, then $\{\psi \mid p(\psi, S)\} \cap \{\psi \mid p(\psi, T)\} \subset \{\psi \mid p(\psi, S \cap T)\}$; a similar identity holds w.r.t. union.*

2. *if p is c-monotone, then $\{S \mid p(\psi \wedge \phi, S)\} \subset \{S \mid p(\psi, S)\} \cap \{S \mid p(\phi, S)\}$; a similar identity holds w.r.t. disjunction.*

3. *similar identities hold for i-monotone and c-anti-monotone patterns.*

Proof Sketch: Let's take the first part in proposition as an example. Suppose p is frequency, then $\{S \cap T\} \subset \{T\}$, and $\{S \cap T\} \subset \{T\}$. By definition, $p(\psi, S)$ means ψ is the strongest circumstances in which $p(S)$ holds, and S is the maximal itemset that is frequent in ψ . So $\{\psi \mid p(\psi, T)\} \subset \{\psi \mid p(\psi, S \cap T)\}$, and $\{\psi \mid p(\psi, S)\} \subset \{\psi \mid p(\psi, S \cap T)\}$; if we take the intersection of $\{\psi \mid p(\psi, T)\}$ and $\{\psi \mid p(\psi.T)\}$, of course it will be the subset of each individual one and we have the proof.

As an example, frequency satisfies the two identities in the proposition. On the other hand, *purchasecap* satisfies the first identity as well as the identity: $\{S \mid \textit{purchasecap}(\psi, S)\} \cap \{S \mid \textit{purchasecap}(\phi, S)\} \subset \{S \mid \textit{purchasecap}(\psi \wedge \phi, S)\}$.

The identities above illustrate what kind of pruning opportunities exist when processing complex queries. For instance, to find itemsets that are frequently bought in northeast and in fall. If we already have itemsets corresponding frequent purchases in northeast and frequent purchases in fall, one way of processing this query is to verify each itemset in the intersection for frequency in the circumstance "northeast and fall." A strategy like this makes sense when processing queries corresponding to circumstances at multiple granularities.

Chapter 3

Data Mining on Itemset Lattice

In this chapter, we will focus on mining patterns on itemset lattice. As mentioned in the previous chapter, from bilattice point of view, the traditional data mining problem is actually itemset mining under *true* circumstance [GLW00]. Although there has been a lot of work done on association mining, there is relatively little work on other patterns, such as correlations. This chapter will introduce constraints into correlations, which has not yet been studied.

3.1 Introduction and Motivation of the Problem

Besides association mining, it has been recognized that there is a need to explore alternate patterns/rules. One such notion is *correlation.* Brin *et al.* [BMS97] have studied the problem of efficiently finding (minimal) correlated (or dependent) sets of objects from large databases. Their definition of correlation is based on the chi-squared metric, which is widely used by statisticians for testing independence. The idea is that an itemset is said to be correlated with probability α provided its chi-squared metric exceeds the expected chi-squared value corresponding to the probability α. In analogy to the classical framework of associations, where frequency (support) of itemsets is used as a measure of statistical significance, Brin *et al.* used a notion of CT-supported [1] as a measure of statistical significance of an itemset.

However, this frame work of correlation has the following fundamental problems as association mining before the introduce of constraints: (i) lack of user exploration and

[1] CT stands for contingency table, explained later.

guidance (for example, expensive computation undertaken without user's approval), and (ii) lack of focus (for example, cannot limit computation to just a subset of rules of interest to the user). Many of a time, users are just interested in a subset of the whole answers. To solve these problems, user need to be allowed to express his/her focus, using constraints drawn from *descriptive attributes* including domain, class, and SQL-style aggregate constraints which capture the application semantics. The properties of these constraints can also be used in pruning the search space, for efficiently computing frequent sets that satisfy user-specified application-specific constraints.

So applications for mining minimal correlated sets satisfying given constraints naturally arise. For example, a manager of a supermarket may want to verify whether customers who do not want to spend a lot of money overall, only buy cheaper items. [2] The conjunction of constraints $S.\texttt{price} < c$ & $sum(S.\texttt{price}) < maxsum$ captures this situation. Both constraints are *anti-monotone*, meaning if a set satisfies the constraint, then so does every subset. In addition, the first constraint satisfies a property called *succinctness*, first defined in [NLHP98]. This intuitively means that the constraint can be pushed deep down into an Apriori-style level-wise algorithm so that it affects pruning even before anti-monotonicity takes effect. As another example, the manager may just want to find whether there is any correlation among items of a single type, which can be useful in mapping items to departments and in shelf planning. The constraint $|S.\text{type}| = 1$ corresponds to this situation. This constraint is also anti-monotone. In a third case, the manager may especially be interested in the correlations of those items whose total price is greater than a certain value, described by the constraint $sum(S.\texttt{price}) > minsum$. This constraint is neither anti-monotone nor succinct. (See [NLHP98, LNHP99] for a thorough analysis of various constraints and their use in pruning optimization of constrained frequent set queries.)

This chapter addresses the pruning optimization of constrained correlation queries as the following question: *given a set of constraints, how to efficiently find itemsets that are CT-supported, correlated, and are valid w.r.t. given constraints?* At first sight, a straightforward extension of the techniques in [NLHP98] might seem to solve this problem. If asking for *all* such itemsets, this is indeed the case. However, Brin *et al.* [BMS97] showed that being correlated was a monotone (upward closed) property:

[2] For the same total price, they prefer to buy more cheap items than fewer expensive items.

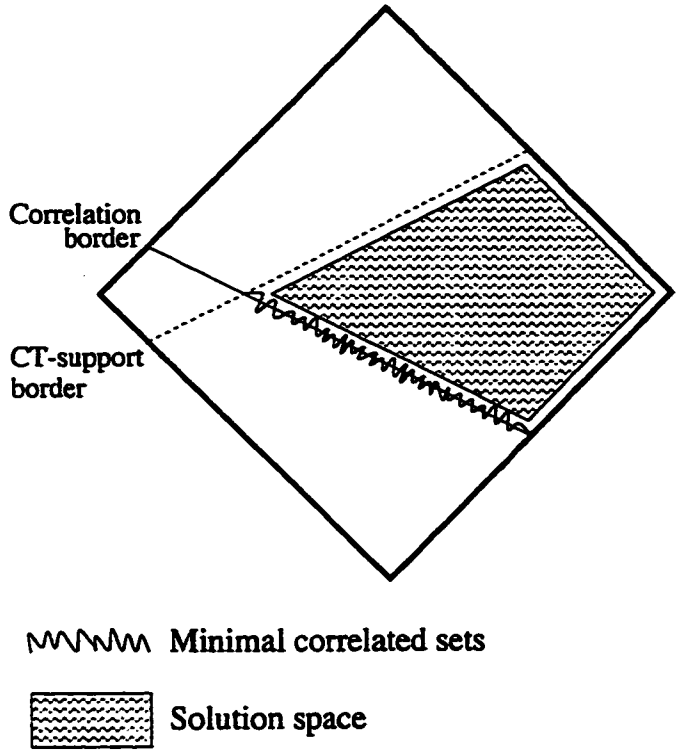

Figure 4: The Borders of Correlation and CT-support in the Itemset Lattice

all supersets of a correlated set were also correlated, while being CT-supported was an anti-monotone (downward closed) property: all subsets of a CT-supported set were CT-supported. Based on this observation, they made a case for computing just the minimal correlated (and CT-supported) sets. Figure 4 shows the solution space corresponding to itemsets that are both correlated and CT-supported. The lower border of the figure corresponds to the minimal itemsets in this space. The rationale then is that the user might be interested in the smallest "objects" in the space rather than the whole of them. Indeed, knowing that bread and butter are correlated is informative, while given this, it is less interesting to know additionally that bread, butter, and cereal are correlated, or that the set consisting of bread, butter, cereal, and toothpaste is statistically insignificant.

Now, consider the problem of finding all itemsets that are CT-supported, correlated, and valid w.r.t. given constraints, which in addition, are *minimal*. The first difficulty is that there are two ways of interpreting this minimality. This leads to two notions of answer sets – (i) valid minimal correlated and CT-supported itemsets and (ii) minimal valid correlated and CT-supported itemsets. These are not always

CT	Doughnuts	Doughnuts	Row Sum
coffee	30	39	69
coffee	20	11	31
Col Sum	50	50	100

Table 1: Example Contingency Table.

identical. There might be interest in computing either of these answer sets, depending on the applications. Different techniques are called depending on which of these answer sets is desired by the user. However, there are situations under which both answer sets coincide. A second difficulty comes from constraints which are monotone. Two examples of monotone constraints from the market basket domain are $sum(S.\texttt{price}) \geq 1000$ and $min(S.\texttt{price}) \leq 50$, for itemset S. We will see that a direct application of the techniques in [NLHP98] for the problem studied here may yield incorrect answers when monotone constraints are considered. Intuitively, monotone constraints exhibit a behavior similar to the property of being correlated, and this should be reflected in the way they are handled in pruning the search space. There is no analog of this in the framework of [NLHP98, LNHP99].

3.2 Constrained Correlations

3.2.1 Correlation queries

Brin *et al.* [BMS97] approached correlation through the notion of dependence. Two items are dependent provided the probability of occurrence of one given the other is different from the absolute probability of the first. They showed that dependence was monotone in that every superset of a dependent itemset was also dependent. Thus, there is an interest in finding minimal dependent itemsets. For measuring dependence, they use the chi-squared statistics. This can be obtained by constructing a contingency table for the itemset in question. Intuitively, the *contingency table* of an itemset S is a table that lists the count, in a given database D, of every minterm involving S. For example, Table 1, adapted from [BMS97], shows a possible contingency table for the itemset $\{\text{coffee}, \text{doughnuts}\}$.

The chi-squared statistic itself is calculated from a contingency table as $\chi^2 =$

$\Sigma_{r \in \text{minterms}(S)}(O(r) - E(r))^2/E(r)$, where $O(r)$ is the observed number of occurrences of the minterm r, while $E(r)$, its expected value, is calculated under the independence assumption [BMS97]. Corresponding to each contingency table, there is a degree of freedom, which is always 1 for boolean variables. In addition, there is a corresponding p value, a value in $[0, 1]$, which indicates the probability of witnessing the observed counts if the items in question were really independent. A low p value is thus grounds for rejecting the independence hypothesis. More concretely, an itemset is dependent (or correlated) at significance level α provided the p value corresponding to the chi-squared statistic calculated for this set is at most $1 - \alpha$.

Besides being correlated, a set must exhibit some kind of statistical significance. In [BMS97], the authors impose the following measure of significance. Let s be a user specified minimum support threshold and $p\%$ be a user supplied cutoff percentage value, to be considered statistically significant, an itemset S must be such that at least $p\%$ of the cells in the contingency table for S having their support not less than s. This is a property called *CT-supportedness* which, like frequency, can be readily shown to be anti-monotone. Brin *et al.* [BMS97] gave an efficient algorithm for finding all minimal correlated and statistically significant sets, where the parameters $\alpha, s, p\%$ were all chosen by the user.

3.2.2 Adding constraints

Intuitively, a *constrained correlation query* asks for itemsets that are CT-supported and correlated, and further satisfy a set of constraints. [NLHP98] gives an exposition of the constraint language. Here, an example is showed to illustrate it. The query $\{S \mid S$ is CT-supported and correlated $\&$ *snacks* $\notin S.\texttt{type}$ $\&$ $\{soda, frozenfood\} \subset S.\texttt{type}$ $\&$ $max(S.\texttt{price}) < 50$ $\&$ $sum(S.\texttt{price}) \geq 100\}$ asks for CT-supported and correlated itemsets which do not include any snack items, include at least one soda item and at least one frozen food item, and further with a maximum price less than \$50 and a total price of at least \$100. Formally, a *constrained correlation query* is an expression of the form $\{S \mid S$ is correlated and CT-supported $\&$ S satisfies $C\}$, where C is a conjunction of constraints drawn from the class of domain, class, and SQL-style aggregation constraints.

Recall that a constraint C is *monotone* provided every superset of a set that satisfies C also satisfies C. It is *anti-monotone* provided every subset of a set that satisfies

C also satisfies C. Here only constraints that either anti-monotone or monotone are considered. The following lemma shows that this still allows for a rich variety of constraints to choose from.

Lemma 1 Let C be any constraint of one of the following forms:

1. $agg(S.A)\ \theta\ c$, where *agg* is one of *max, min, sum, count* θ is one of $\leq, <, \geq, >$, and A is an attribute with a non-negative domain, and c is a value from it.
2. $CS\ \theta\ S.A$, where CS is a constant set drawn from a domain compatible with that of attribute A, and θ is one of $\subset, \not\subset$.
3. $CS \cap S.A\ \theta\ \emptyset$, where CS is a constant set drawn from a domain compatible with that of attribute A and θ is one of $=, \neq$.

Then C is either anti-monotone or monotone.

Proof Sketch: We will only show the proof of $min(S.A) \geq c$ is anti-monotone here, the proof of other cases is similar. Because the constaint is anti-monotone, if $min(S.A) \geq c$, and $S'.A \subset S.A$, we have $min(S'.A) \geq c$. Suppose this is not true, there must exist an element $e \in S'.A$ such that $e < c$. But then the same element e is contained in every superset of $S'.A$, including $S.A$. Thus it is necessary that $min(S'.A) \geq c$.

Indeed, a large portion of the constraints allowed in the constraint language introduced in [NLHP98] is either monotone or anti-monotone, according to Lemma 1. Two notable kinds of exceptions are: (i) constraints involving average and (ii) those of the form $agg(S.A) = c$, where *agg* is one of *min, max, sum, count*. average constraints will be discussed in Section 3.4.3. Note that a constraint of the form $agg(S.A) = c$ can be broken into $agg(S.A) \leq c$ & $agg(S.A) \geq c$. From the proof of the lemma above, it can be shown that one of the conjuncts must be monotone and the other anti-monotone. The techniques that developed can handle any conjunction of such constraints. Thus, the focus on constraints which are either monotone or anti-monotone does not restrict the expressive power too much.

Next, let's define answer sets for constrained correlation queries. Brin *et al.* [BMS97] argued that minimal CT-supported and correlated sets captured the essence of answering correlation queries (without constraints). In keeping with this rationale, it is appropriate to build in some notion of minimality in defining the answer sets.

The first definition is obtained by taking the definition of Brin *et al.* and imposing the condition that itemsets must be valid w.r.t. the constraints in the query.

Definition 1 Let $Q = \{S \mid S \subset \texttt{Item} \ \& \ C\}$ be a constrained correlation query. Then the set of *valid minimal answers* of Q is given by $\text{VALIDMIN}(Q) = \{S \mid S$ is a minimal correlated and CT-supported itemset & S satisfies $C\}$.

As an example, consider the query $Q_1 = \{S \mid S \subset \texttt{Item} \ \& \ max(S.\texttt{price}) \geq 100\}$, $\text{VALIDMIN}(Q_1)$ consists of all those minimal correlated and CT-supported itemsets, which in addition, have their maximum price not less than \$100.

A second definition of answer sets is obtained by considering the space of all correlated, CT-supported, and valid itemsets and asking for the minimal ones among them. This is meaningful only when the space of all answers is in a single region bounded from above and below by well-defined borders. This is indeed the case for the scenario studied by Brin *et al.* wherein correlation forms the lower border and CT-support forms the upper border (see Figure 4). When throwing in constraints, such a well-defined space will still hold as long as each constraint considered is either monotone (like correlation) or anti-monotone (like CT-support). Constraints of this type form either a lower or an upper border as the case may be and the answers looking for are within the region bounded by all these borders. It thus makes sense to ask for the minimal answers in this space, as they intuitively give the smallest objects which are interested.

Definition 2 Let Q be a constrained correlation query $Q = \{S \mid S \subset \texttt{Item} \ \& \ C\}$ such that each constraint in C is either monotone or anti-monotone. Then the set of *minimal valid answers* is given by $\text{MINVALID}(Q) = \{S \mid S \subset \texttt{Item} \ \& \ S$ satisfies C & S is CT-supported and correlated & S is minimal$\}$.

For the query Q_1 above, $\text{MINVALID}(Q_1)$ consists of all answers which are valid, CT-supported, and correlated, and are minimal among all such objects, *i.e.* none of their proper subsets satisfy these properties.

The two sets VALIDMIN and MINVALID are illustrated in Figure 5. It is easy to see that for any query Q, $\text{VALIDMIN}(Q) \subseteq \text{MINVALID}(Q)$: any minimal CT-supported and correlated sets that are also valid must be minimal sets that satisfy all three of these conditions. Still, there are cases where $\text{VALIDMIN}(Q)$ is a proper subset of

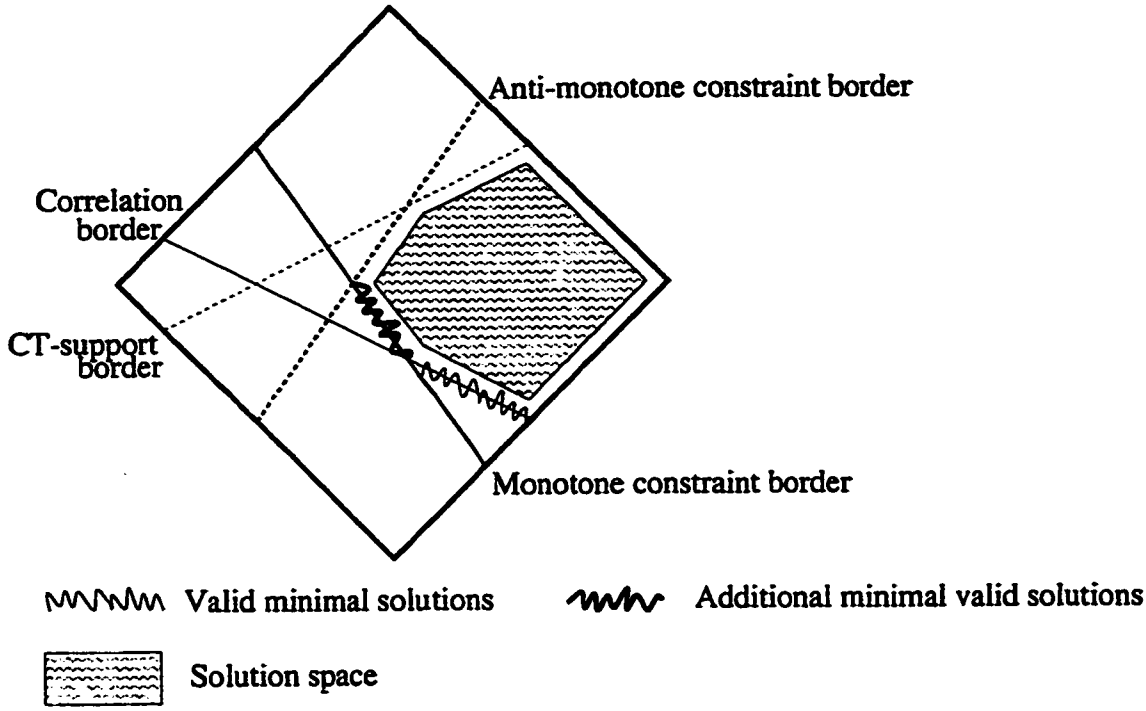

Figure 5: The Valid Minimal Solutions and Minimal Valid Solutions.

MINVALID(Q). To illustrate this, let's consider an example where the domain of Item has five items, 1, ..., 5, representing milk, bread, butter, cereal, cheese. For simplicity, let item i have price $\$i$. Suppose that all itemsets of size 2 are CT-supported and correlated, and further assume that all itemsets up to size 4 are CT-supported. Let the constraint be $C \equiv max(S.\texttt{price}) \geq 5$. Then VALIDMIN($Q$) = $\{\{i, j\} \mid i \geq 5 \vee j \geq 5\}$. In particular, the set $\{milk, bread\}$, which is both CT-supported and correlated, is not valid. However, the set $\{milk, bread, cheese\}$ is valid. It is also CT-supported and correlated. So, $\{milk, bread, cheese\} \in$ MINVALID(Q) but $\{milk, bread, cheese\} \notin$ VALIDMIN(Q). In summary, the following result could be found:

Theorem 1 Let Q be a constrained correlation query $Q = \{S \mid S \subset \texttt{Item} \ \& \ C\}$ such that each constraint in C is either monotone or anti-monotone. Then

1. VALIDMIN(Q) $\subseteq$ MINVALID(Q).
2. If all constraints are anti-monotone, then VALIDMIN(Q) = MINVALID(Q).

Proof Sketch: Figure 5 shows a very clear proof of the theorem. If the valid minimal answers are interested, according to its definition, we first find the minimal correlated border, then check the monotone constraint to eliminate the invalid

sets, and get the correct answers. If the minimal valid answers are interested, then according to its definition, we will check the monotone border and the correlation border, then to minimize the answers, which will give us the additional minimal valid solutions. If there is no monotone constraints, the monotone border will not exist. And obviously, we will get the same solution space for both interesting answer sets.

An additional property of constraints that should be exploited is succinctness [NLHP98]. Let us denote the solution space of a constraint C as $\mathrm{SAT}_C(\mathtt{Item}) = \{S \mid S \subset \mathtt{Item} \ \& \ S \text{ satisfies } C\}$. A constraint C is *succinct* provided there are itemsets $I_1, \ldots, I_k \subset \mathtt{Item}$ such that: (i) each I_j can be expressed as $I_j = \sigma_{p_j}(\mathtt{Item})$ for some selection condition p_j, $1 \leq j \leq k$, and (ii) the solution space $\mathrm{SAT}_C(\mathtt{Item})$ can be written as an expression involving powersets of $I_1, \ldots, I_k$ using union and minus. An example is the constraint $C_1 \equiv max(S.\mathtt{price}) \leq 100$. Let $I_1 = \sigma_{\mathtt{price} \leq 100}(\mathtt{Item})$. Then $\mathrm{SAT}_{C_1}(\mathtt{Item}) = 2^{I_1}$.[3] As another example, for the constraint $C_2 \equiv \{beer, chips\} \subset S.\mathtt{type}$, define $I_1 = \sigma_{\mathtt{type}=beer}(\mathtt{Item})$, $I_2 = \sigma_{\mathtt{type}=chips}(\mathtt{Item})$, and $I_3 = \sigma_{\mathtt{type} \neq beer \ \&\ \mathtt{type} \neq chips}(\mathtt{Item})$. Then $\mathrm{SAT}_{C_2}(\mathtt{Item}) = 2^{\mathtt{Item}} - 2^{I_1} - 2^{I_2} - 2^{I_3} - 2^{I_1 \cup I_3} - 2^{I_2 \cup I_3}$. In this expression, all itemsets that violate C_2 are eliminated from $2^{\mathtt{Item}}$. The main value of succinctness is that for a succinct constraint C, all and exactly the itemsets can be generated in the solution space $\mathrm{SAT}_C(\mathtt{Item})$ without recourse to generating all possible itemsets and testing them one by one for constraint satisfaction. It was shown in [NLHP98] that all succinct constraints C have a *member generating function* (MGF) of the form $\mathrm{SAT}_C(\mathtt{Item}) = \{X_1 \cup \cdots \cup X_k \mid X_j \subset \sigma_{p_j}(\mathtt{Item}), 1 \leq j \leq k \ \&\ X_j \neq \emptyset, 1 \leq j \leq m, \text{ for some } m \leq k\}$. As an example, $\mathrm{SAT}_{C_1}(\mathtt{Item}) = \{X \mid X \subset \sigma_{\mathtt{price} \leq 100}(\mathtt{Item})$. As another example, $\mathrm{SAT}_{C_2}(\mathtt{Item}) = \{X_1 \cup X_2 \cup X_3 \mid X_1 \subset \sigma_{\mathtt{type}=beer}(\mathtt{Item}) \ \&\ X_2 \subset \sigma_{\mathtt{type}=chips}(\mathtt{Item}) \ \&\ X_3 \subset \sigma_{\mathtt{type} \neq beer \ \&\ \mathtt{type} \neq chips}(\mathtt{Item}) \ \&\ X_1 \neq \emptyset \ \&\ X_2 \neq \emptyset\}$. It was also shown that MGFs for individual succinct constraints can be combined into an MGF for their conjunction [NLHP98].

3.3 Algorithms for Constrained Correlations

Let's first review Brin *et al.*'s algorithm showed in Figure 6, refer as Algorithm BMS below, for computing minimal correlated and CT-supported sets. This algorithm exploits the properties that CT-supportedness is anti-monotone and being correlated

[3] Excluding empty sets is a simple technicality.

Algorithm BMS (Brin-Motwani-Silverstein)
Input: A chi-squared significance level α, support s, support fraction p, and basket data D.
Output: The set of all valid minimal correlated itemsets from D.
Method:

1 Initialize $\text{CAND}_2 = \text{SIG} = \emptyset$;

2 for each $i \in I$ count $O(i)$;

3 for each pair $i_1, i_2 \in I$

3.1 if $(O(i_1) > s \;\&\&\; O(i_2) > s)$ add $\{i_1, i_2\}$ to CAND_2;

4 k = 2;

5 while ($\text{CAND}_k \neq \emptyset$) {

5.1 NOTSIG := $\emptyset$;

5.2 for each $S \in \text{CAND}_k$ {

5.2.1 construct the contingency table $CT(S)$;

5.2.2 if (S has CT-support $\geq p$)

5.2.2.1 if (chi-squared value of $CT(S)$ is $\geq \alpha$)
add S to SIG;
else add S to NOTSIG; }

5.3 $k++$;

5.4 Set CAND_k to be the set of all itemsets S of size k, such that every $k-1$-subset of S is in NOTSIG; }

6 output SIG;

Figure 6: Algorithm BMS.

is monotone. The former property is exploited in an Apriori-style pruning: processing sets in the itemset lattice from the bottom up, level by level, pruning candidate sets which cannot be CT-supported. The latter property is exploited by arguing that minimal sets capture the essence of the answers to correlation queries. Thus, the moment a (minimal) correlated and CT-supported set is found, there is no need to consider its supersets.

3.3.1 Computing valid minimal answers

The first algorithm for computing valid minimal answers of a constrained correlation query is obtained by a straightforward adaptation to Algorithm BMS. First run the original BMS algorithm, then check the output for constraints validation.

Clearly, Algorithm BMS+ is naive in that it completely ignores the selectivity and potential pruning power that may be provided by the constraints. The second algorithm is, however, obtained by taking these effects into accounts and making the following modifications to Algorithm BMS.

Algorithm BMS+
Input: A chi-squared significance level α, support s, support fraction p, a set of constraints C, and basket data D.
Output: The set of all valid minimal correlated itemsets from D.
Method:

1 Run Algorithm BMS up to step (5) and compute the set SIG containing all minimal CT-supported and correlated sets;

2 output those sets in SIG that satisfy the constraints C;

Figure 7: Algorithm BMS+.

I. Preprocessing: Algorithm BMS considers all pairs of frequent items as candidate sets of size 2. When constraints are present, some improvement can be made. Firstly, split the set of query constraints into $C = C_{ams} \cup C_{am\bar{s}} \cup C_{ms} \cup C_{m\bar{s}}$, respectively into constraints that are succinct and anti-monotone, anti-monotone but not succinct, succinct and monotone, and monotone but not succinct. Sometimes $C_{ams} \cup C_{am\bar{s}}$ is denoted as C_{am}, the set of anti-monotone constraints and $C_{ms} \cup C_{m\bar{s}}$ as C_m, the set of monotone constraints.

$\text{GOOD}_1 = \{i \mid i \in \texttt{Item}\ \&\ \{i\}$ satisfies $C_{am}\}$ denotes the 1-itemsets that satisfy all anti-monotone constraints. Let $\text{CAND}_1^+ = \{i \mid i \in \text{GOOD}_1\ \&\ \{i\}$ satisfies $C_{ms}\}$ and let $\text{CAND}_1^- = \text{GOOD}_1 - \text{CAND}_1^+$. Now, let $L_1^+ = \{i \mid i \in \text{CAND}_1^+\ \&\ O(i) \geq s\}$ and $L_1^- = \{i \mid i \in \text{CAND}_1^-\ \&\ O(i) \geq s\}$.

In the preprocessing stage, the new algorithm computes the sets L_1^+ and L_1^- as suggested above. This can be done in one scan of the database.

II. Forming candidate sets: For the sake of easy exposition, let's assume that the MGF for the conjunction of all succinct constraints in the query is of the form $\{X_1 \cup X_2 \mid X_1 \subset \sigma_{p_1}(\texttt{Item})\ \&\ X_2 \subset \sigma_{p_2}(\texttt{Item})\ \&\ X_1 \neq \emptyset\}$. Extension to the general forms of MGFs (see Section 3.2) is straightforward.[4] Candidate sets of size two are formed as follows.

$\text{CAND}_2 = \{\{i_1, i_2\} \mid i_1 \in L_1^+\ \&\ i_2 \in (L_1^+ \cup L_1^-)\}$.

More generally, for $k > 2$, set CAND_k to contain all k-itemsets S such that:
$\forall S' : (S' \subset S\ \&\ |S'| = k-1\ \&\ S' \cap L_1^+ \neq \emptyset \Rightarrow S' \in \text{NOTSIG})$. The rationale is below. Consider a k-itemset S and two $(k-1)$-subsets of S - S_1, S_2 such that $S_1 \cap L_1^+ \neq \emptyset$ and $S_2 \cap L_1^+ = \emptyset$. Now, by virtue of the way L_1^+ and L_1^- are computed and the way candidate sets are formed, we can see that $S_1 \in \text{CAND}_{k-1}$, but $S_2 \notin \text{CAND}_{k-1}$. In

[4] A subtle point, however, is that if a monotone succinct constraint requires more than one witness, then it cannot be included in L_1^+. It should be enforced later, much like $C_{m\bar{s}}$, so valid minimal answers can be correctly computed.

```
while CAND_k ≠ ∅ {
    for each S ∈ CAND_k {
        if (S satisfies C_am¯) {
            construct CT(S);
            if (S has CT-support ≥ p)
                if (CT(S) has chi-squared value ≥ χ²_α) {
                    if (S satisfies C_m¯) {
                        add S to SIG; }
                    else add S to NOTSIG; } }
    }
}
```

Figure 8: SIG and NOTSIG for Algorithm BMS++.

other words, we would construct contingency tables for all subsets of S that are valid with respect to $C_{am} \cup C_{ms}$, and for none of its subsets which are invalid with respect to these constraints. Thus the candidate formation logic of Algorithm BMS is modified to reflect this.

III. Computation of the SIG and NOTSIG sets: The main difference is that each potential member of SIG set needs to be checked for satisfaction of all non-succinct constraints. Of these, the anti-monotone constraints are handled similarly to the way CT-support test is performed, while the monotone ones are handled similarly to the way correlation is checked. Figure 8 summarizes this modification.

The algorithm obtained by applying modifications I-III above is refered as Algorithm BMS++, or "Constrained BMS for valid minimal answers."

3.3.2 Computing minimal valid answers

A straightforward algorithm for computing minimal valid answers is given first. In this algorithm, the sets SIG and NOTSIG are used in a context different from that used by Algorithm BMS. Assume the sets used by Algorithm BMS are renamed to SIG' and NOTSIG'. The idea is that when Algorithm BMS finishes, it computes all minimal CT-supported and correlated itemsets and leaves them in SIG'. Of these, we can add those that satisfy the constraints to the set SIG, which will eventually contain all minimal valid answers. At this point, SIG will contain all valid minimal answers.[5] The difficulty now is that the remaining minimal valid answers *cannot* be

[5]Recall that all valid minimal answers are also minimal valid answers.

Algorithm BMS*
Input: A chi-squared significance level α, support s, support fraction p, a set of constraints C, basket data D.
Output: The set of all minimal valid correlated itemsets from D.
Method:

1 Run Algorithm BMS to compute the sets SIG and NOTSIG; rename those sets to SIG' and NOTSIG';

2 NOTSIG = SIG = $\emptyset$;

3 for each $S \in$ SIG' {

3.1 if (S satisfies C_{am})
if (S satisfies C_m)
add S to SIG;
else add S to NOTSIG;}

4 Let k = (the least cardinality of any set in NOTSIG) + 1;

5 Set CAND_k to contain all k-sets S such that $(\forall S' : S' \subset S \;\&\; |S'| = k-1 \Rightarrow S' \in \text{NOTSIG})$;
(i.e. all $(k-1)$-subsets of S satisfy C_{am} but not C_m.)

6 while $\text{CAND}_k \neq \emptyset$ {

6.1 for each $S \in \text{CAND}_k$ {

6.1.1 construct $CT(S)$;
6.1.2 if (S has CT-SUPPORT $\geq p$)
if (S satisfies C_m)
add S to SIG;
add S to NOTSIG; }

7 $k++$;

8 Set CAND_k to contain all k-sets S such that all $(k-1)$-subsets of S are in NOTSIG;}

9 output SIG;

Figure 9: Algorithm BMS*.

found directly from SIG' and NOTSIG'. For this, once again, an upward, level-by-level sweep of the itemset lattice is needed for the checking of CT-support and satisfaction of all monotone constraints. Notice that we do *not* need to check for correlation (*i.e.* chi-squared test) since the sets being examined now are supersets of sets known to be correlated. The complete naive algorithm is given in Figure 9.

The next algorithm exploits the pruning affected by the query constraints as early as possible. As with Algorithm BMS++, this algorithm will be presented by describing the modifications to be made for Algorithm BMS.

I. Preprocessing: This step is identical to that for Algorithm BMS++. In particular, we compute the sets L_1^+, L_1^- as outlined early. [6]

II. Formation of candidate sets: The formation of CAND_k, for $k \geq 2$, is

[6] Regardless of the number of witnesses involved, here all succinct constraints can be incorporated in L_1^+, unlike for Algorithm BMS++.

1 $k = 2$;

2 while ($\text{CAND}_k \neq \emptyset$) {

2.1 $\text{SUPP}_k = \emptyset$;

2.2 for each $S \in \text{CAND}_k$ {

2.2.1 if (S satisfies $C_{am\bar{s}}$) {

2.2.1.1 construct $CT(S)$;

2.2.1.2 if (S has CT-support $\geq p$)
add S to SUPP_k; } }

2.3 $k++$;

2.4 Form CAND_k; }

3 $k = 2$; SIG = NOTSIG = $\emptyset$; $C_k = \text{SUPP}_k$;

4 while $C_k \neq \emptyset$ {

4.1 for each $S \in C_k$ {

4.1.1 if ($CT(S)$ has chi-squared $\geq \chi^2_\alpha$ && S satisfies $C_{m\bar{s}}$)
add S to SIG;
else add S to NOTSIG; }

4.2 $k++$;

4.3 Set C_k to contain all k-sets in SUPP_k such that $\forall S' : S' \subset S \;\&\; |S'| = k-1 \Rightarrow S' \in$ NOTSIG; }

Figure 10: SIG and NOTSIG for Algorithm BMS**.

identical to that in Algorithm BMS++.

III. Computation of the SIG and NOTSIG sets:

Instead of the SIG and NOTSIG sets, supported sets, $\text{SUPP}_k, k \geq 2$ are computed as follows. Use only CT-support and anti-monotone constraints and then use it along with the monotone constraints to compute NOTSIG and SIG (see Figure 10).

The algorithm obtained by applying modifications I-III described in this section is refered as Algorithm BMS** or "Constrained BMS for minimal valid answers."

The following results show the correctness of the various algorithms presented in this section.

Theorem 2 The following is true:

1. Algorithms BMS+ and BMS++ correctly compute all and only valid minimal answers.

2. Algorithms BMS* and BMS** correctly compute all and only minimal valid answers.

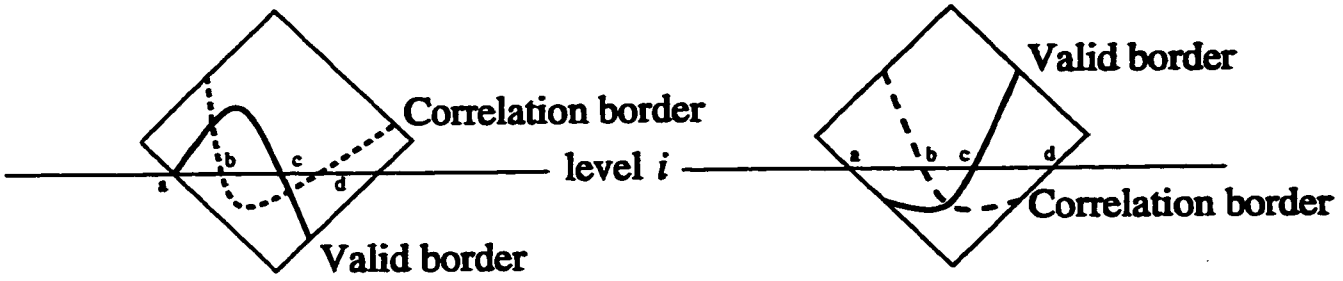

Figure 11: The Interplay Between Constraints and Correlation

3.3.3 Analysis of the algorithms

The number of sets each of the four algorithms needs to consider is analyzed here. Clearly, it is the dominating parameter since it involves database scanning. Computing *CT*-support etc. are cpu operations, and much less expensive.

Assume a fixed set I, a fixed database D, and some fixed cut-offs α, s, and p. Let c_i be the number of correlated sets at level i in the itemset-lattice. Likewise, let v_i be the number of valid sets at level i. Also, let cv_i be the set of correlated and valid sets at level i. Clearly $cv_i \leq c_i$ and $cv_i \leq v_i$, for any level i. This is illustrated in Figure 11. The lattice on the left uses anti-monotone (downward closed) constraints, while the lattice on the right uses monotone (upward closed) constraints. In both lattices, the interval $[b, d]$ corresponds to c_i, the interval $[a, c]$ corresponds to v_i, and the interval $[b, c]$ corresponds to cv_i.

Let k be the highest level on which there are (minimal) correlated sets, and let l be the highest level on which there are valid sets Let |BMS+| be the number of sets algorithm BMS+ needs to consider. The meaning of |BMS++|, and |BMS*|, |BMS**| is similar, mutatis mutandis. Then, $|\text{BMS+}| = \sum_{i=1}^{k} c_i$, $|\text{BMS++}| = \sum_{i=1}^{min(k,l)} cv_i$, $|\text{BMS*}| = \sum_{i=1}^{k} c_i + \sum_{i=k}^{l} v_i$, $|\text{BMS**}| = \sum_{i=1}^{l} v_i$.

The following conclusions can be drawn: If query Q contains monotone constraints, algorithms BMS+ and BMS++ compute the set ValidMin(Q), while algorithms BMS* and BMS** compute the set MinValid(Q). From the formulas above it can be seen that $|\text{BMS++}| \leq |\text{BMS+}|$. The relationship between BMS* and BMS** depends on the relative distributions of the c_i:s and v_i:s. If the selectivity of the constraint is low, it means that $\sum_i v_i \leq \sum_j c_j$. Then $|\text{BMS**}| \leq |\text{BMS*}|$, and Algorithm BMS** is expected to perform better than Algorithm BMS*. If constraint selectivity is high, the inverse quantitative relationship holds, and Algorithm BMS*

is expected to perform better.

When the query Q contains only anti-monotone constraints, VALIDMIN(Q) = MINVALID(Q). It is observed that $|BMS++| \leq |BMS+|$, and that $|BMS++| \leq |BMS**|$. The relationship between between BMS* (BMS+) and BMS** depends on the above constraint selectivity. With high selectivity, Algorithm BMS** should perform better than Algorithm BMS*, and with low constraint selectivity Algorithm BMS* is expected to perform better.

Indeed, as showed in the next section, the experimental evaluation verifies these expectations.

3.4 Experiments

3.4.1 Test data

To evaluate the various algorithms presented, synthetic data is used, generated by two different methods. The first method is developed at IBM Almaden Research Center by Aggrawal and Srikant [AS94]. The second method follows the standard practice in machine learning experiments [CN89, CF88], where test data is generated according to a set of prespecified correlation rules. While the purpose of the first method is to simulate the "real world", that of the second is to verify that the algorithms do really correctly mine out all the correlation rules, which are known in advance.

In Agrawal and Srikant's method, synthetic baskets are formed by simulating a retailing environment. The model of the "real" world is that people tend to buy sets of items together. According to their result, the basket size is first decided according to a Poisson distribution, then a set of "large itemsets" is defined. These large itemset is then assigned to the basket. Each large itemset in the basket has a weight associated with it. The weight corresponds to the probability that the itemset will be picked. This weight is chosen from an exponential distribution and is normalized. If a large itemset is inserted into a transaction, a few random elements are furthermore deleted from the set and some other randomly chosen items are inserted. The number of baskets is varied from 10,000 to 100,000 to study the pruning effect of different algorithms on basket numbers. The average basket size is set to be 20, the average size of large itemsets is set to be 4. The number of items is 1000.

In the second method, the synthetic data was generated based on ten given correlation rules. For each rule r_i, the significance level α_i is set to 0.95, the support threshold s_i is a random value between 70% and 90% of the number of baskets. Therefore, each basket contains a subset of the correlation rules. All other parameters are the same as mentioned above. Randomized items are picked up in case the correlation rules do not generate enough items for a particular basket. To see the effects of different constraints on mining correlations, constraint selectivity was also varied through separate experimental sets, using the same data.

In all experiments the minimum support and CT- support are kept constant. The threshold for support and CT- support are set to be 25%.[7] The confidence level is set to 0.9 for the χ^2-tests. All the experiments are conducted on a Pentium PC with a 200 MHz processor and 64 MB of memory.

3.4.2 Experimental evaluation

Anti-monotone and succinct constraint: In the first set of experiments, the anti-monotone and succinct constraint $\max(S.price) \geq v$ is used to compare the algorithms. Figure 12 shows the cpu usage as a function of the number of baskets for the three algorithms when the constraint-selectivity (proportion of items with price at least v) is set to be 50%, and the number of items is 1000. A conservatively high selectivity is chosen to test the behavior of the algorithms. As mentioned previously, for such constraints, BMS* becomes BMS+, and all four algorithms compute the same results.

For all the algorithms, the results show a similar linear trend of the cpu usage. The correlations that mined out only contained sets with less than four items. Figure 12 (a) (resp., 1(b)) [8] shows the result using synthetic data generated according using method 1 (resp., method 2). For method 2, all the correlations that used in generating the data are indeed found in the results, as well as some other weak correlations. which caused by the randomly distributed items in the transactions. As the number of baskets increases, the cpu usage of BMS++ in Figure 12(a) becomes much smaller than the other two algorithms. Compared to BMS+, BMS++ can speed up the

[7] We ran the experiments for other thresholds too and observed little variation in the trends of the results. Only these results are showed here.

[8] For all the experiments, (a) stands for data generated using method 1, (b) for date method 2.

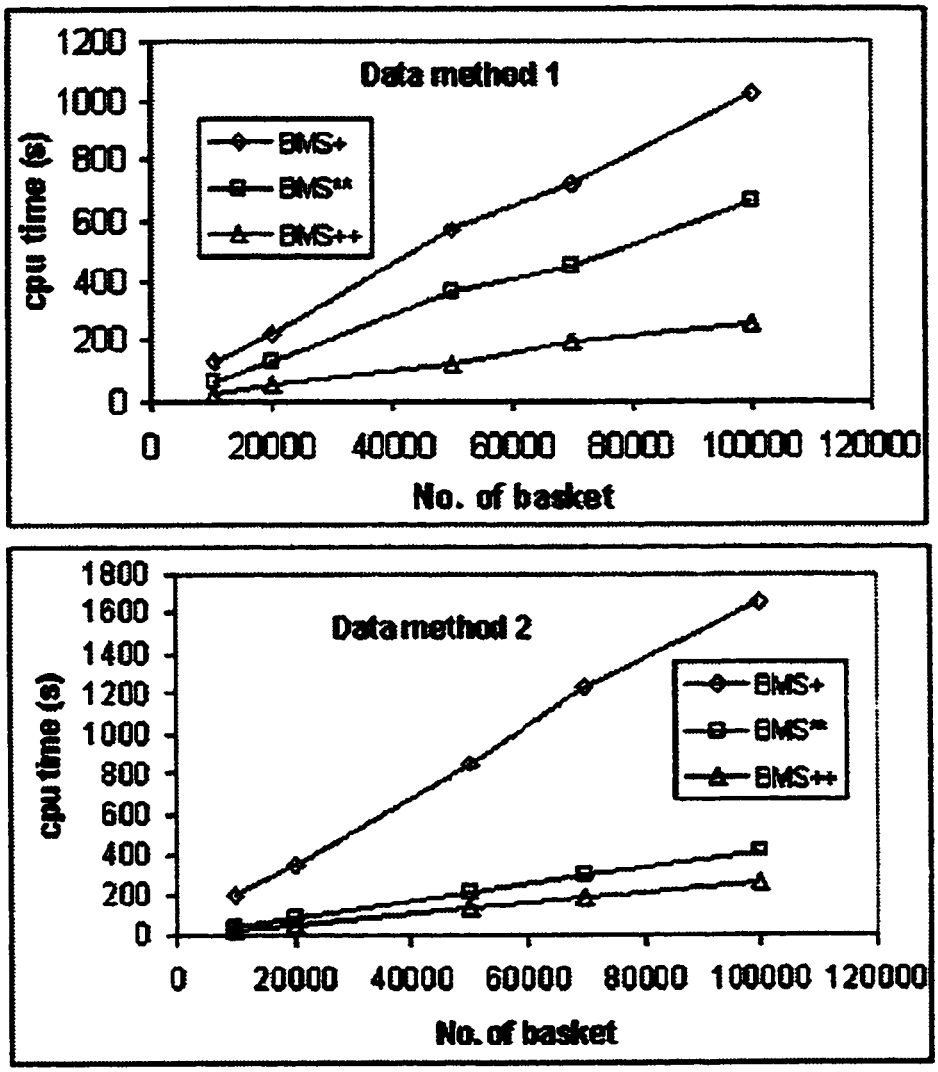

Figure 12: Effect of Transaction Number with Anti-monotone & Succinct Constraints

process by a factor of 10 to 50 for the experimental range tested. In Figure 12(b), the cpu usage for BMS** is close to BMS++, which is much lower than BMS+. This is caused possibly by the method of generating data.

Figure 13 shows the results of cpu usage as a function of the selectivity of constraints when the basket number is 100,000. In the constraint $\max(S.price) \geq v$, different values of v is used to correspond for different selectivities of the constraints. An $x-\%$ selectivity means that $x-\%$ of the items have a price larger than v, whatever that value may be. The y-axis represents the cpu time the various algorithms needed to complete the mining. For BMS+, the cpu usage stays constant as the

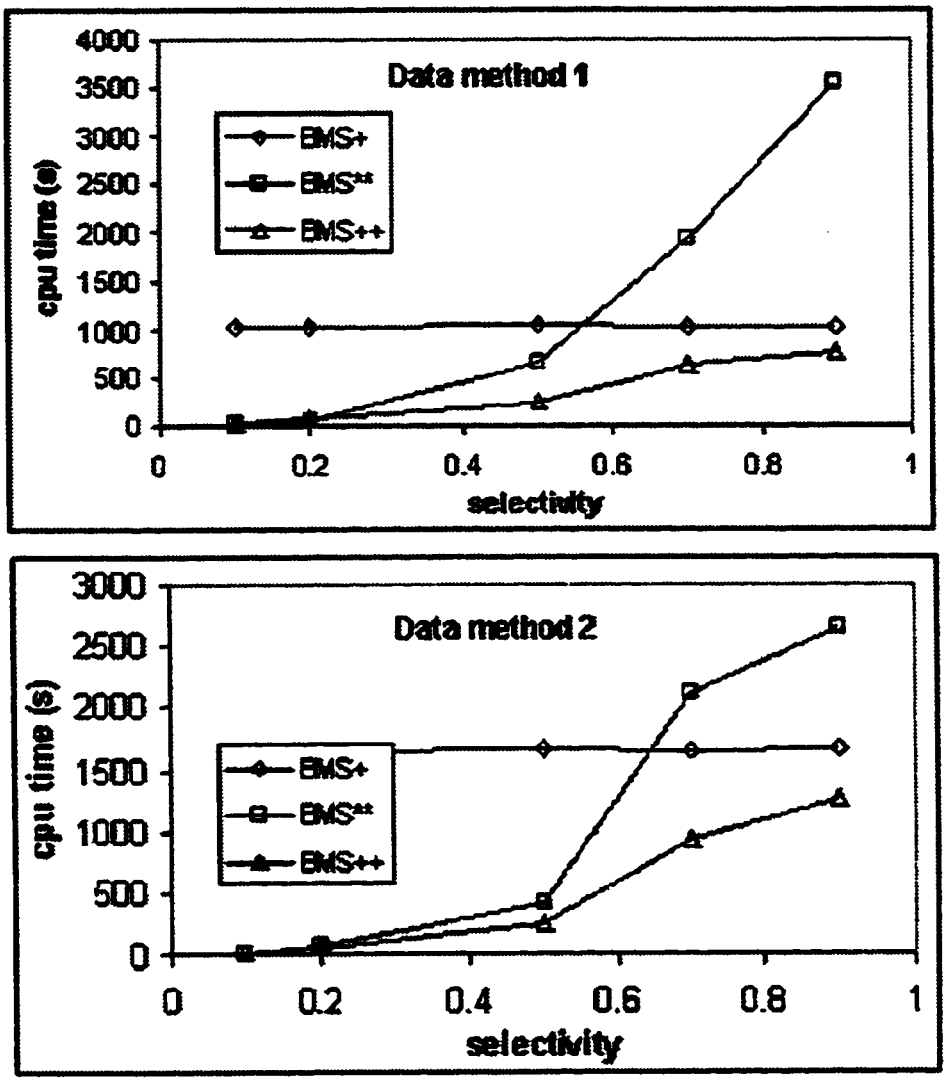

Figure 13: Effect of Selectivity with Anti-monotone & Succinct Constraints

selectivity increases. On the other hand, the cpu usage for BMS** and BMS++ decrease dramatically as the selectivity decreases. When the selectivity is below 30%, the speed-up from using anti-monotone and succinct constraints can be as high as 50 to 100. Even when the selectivity is 80%, the performance for BMS++ is still much better than BMS+, which demonstrates that anti-monotone and succinct constraints can greatly improve the mining performance.

Anti-monotone constraint: Figure 14 (a) and (b) show the amount of cpu usage as a function of basket number, for the two types of synthetic data, respectively. In this experiment series, the anti-monotone but not succinct constraint $sum(S.price) \leq$

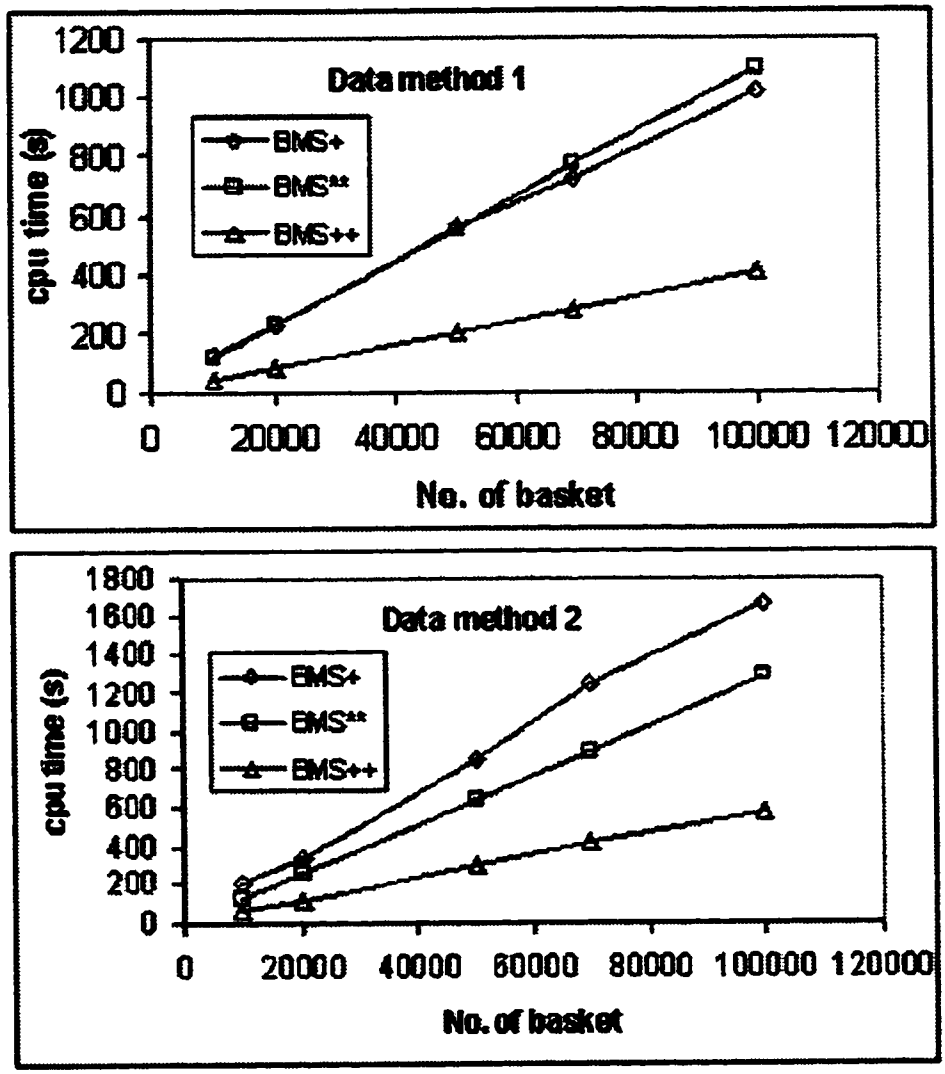

Figure 14: Effect of Transaction Number with Anti-monotone Constraints

Maxsum is used. Recall that for anti-monotone constraints, all four algorithms compute the same answer. The results presented in Figure 14 (a) and (b) have a constraint selectivity of 50%. Similar to Figure 12, a linear increasing trend is exhibited for all the algorithms. As the number of baskets increases, the difference of cpu usage between BMS++ and BMS+ increases. At a basket number of 100,000, the cpu usage for BMS++ is about 1/3 of BMS+, while for BMS** is either the same as the results for BMS+ or about 3/4 of that for BMS+ depending on the data set used.

The effect of constraint selectivity is also examined. Figure 15 (a) and (b) show

these results when the basket number is set to be 100,000. Unlike the previous constraints, the notion of constraint selectivity does not directly make sense in this case. Instead, the price of each item is assigned to be its item number. So for example, item 1 has a price of \$1. At lower values of Maxsum, both BMS** and BMS++ have better performance than BMS+. When the value of Maxsum is close to 4000, there is no pruning effect from the constraint anymore. So BMS+ and BMS++ begin to have the same performance, and the performance for BMS** is much worse than the other two. However, under all circumstances, BMS++ always gives the best performance. BMS** and BMS+ have a cross-over point, below which BMS** has a better performance, and above which BMS+ becomes better.

Succinct and monotone constraint: When the constraint is not anti-monotone, the results of valid minimal computation and minimal valid computation do not coincide. So, the four algorithms need to be examined separately. As in Theorem 2, Algorithm BMS+ and BMS++ correctly compute all and only valid minimal answers. Algorithm BMS** and BMS* correctly compute exactly the minimal valid answers. The constraint that used is $\min(S.price) \leq v$, which is monotone and succinct.

(i) Valid minimal answers. Figure 16 shows the performance of Algorithms BMS+ and BMS++. The selectivity is set to be 50%. Figure 16 (a) is the result using the first synthetic data set. And Figure 16 (b) corresponds to the result using the second synthetic data set. At a basket number of 100,000 the cpu usage for BMS++ is about 70% of the cpu usage of BMS+, in spite of this high selectivity. [9]

Figure 17 shows the selectivity effect on these two algorithms when the basket number is 100,000. When the selectivity is 10%, the cpu usage for BMS++ is only 1/3 of that for BMS+. But when the selectivity is above 70%, the pruning effect of the constraint is negligible, and the performance of BMS++ becomes similar to that of BMS+.

(ii) Minimal valid answers. Figure 18 shows the performance of Algorithms BMS* and BMS**. The selectivity is set to be 50%. Figure 18 is the result using the first synthetic data set. Figure 18 (b) is the result using the second synthetic data set. Unlike Figure 5, the gap between the two algorithms is much bigger.

Figure 19 shows the selectivity effect on these two algorithms when the basket number is 100,000. Unlike the computation of valid minimal answers, where BMS++

[9] Note that the higher the selectivity is, the less selective the constraint has.

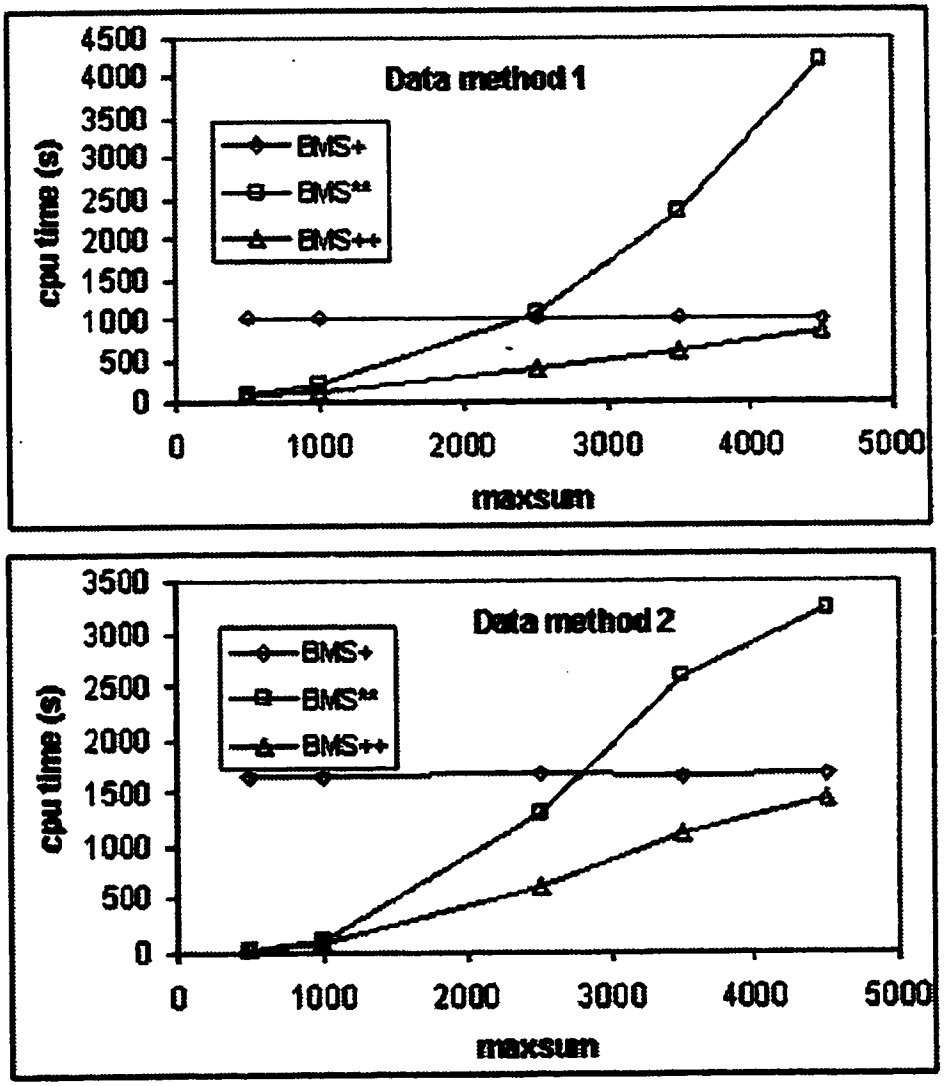

Figure 15: Effect of Selectivity with Anti-monotone Constraints

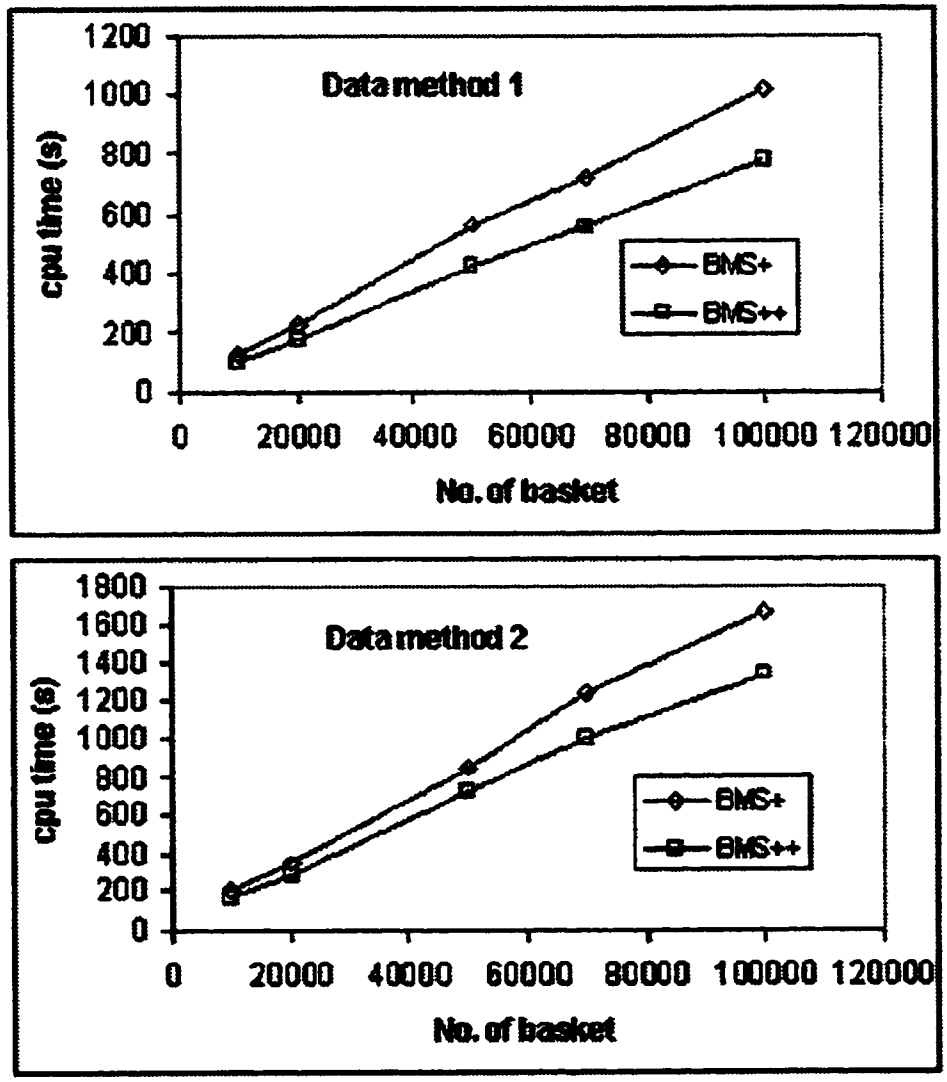

Figure 16: Effect of Transaction Number with Monotone and Succinct Constraints for Valid Minimal Answers

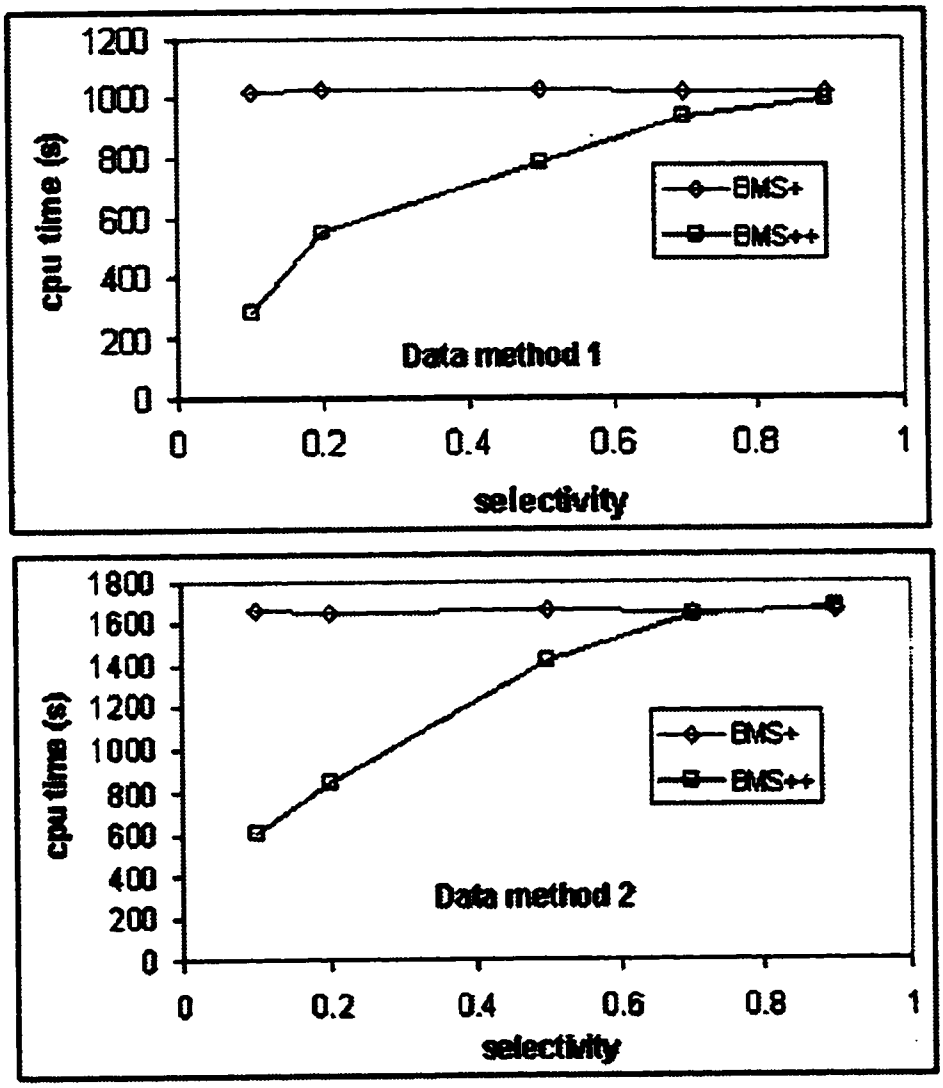

Figure 17: Effect of Selectivity with Monotone and Succinct Constraints for Valid Minimal Answers

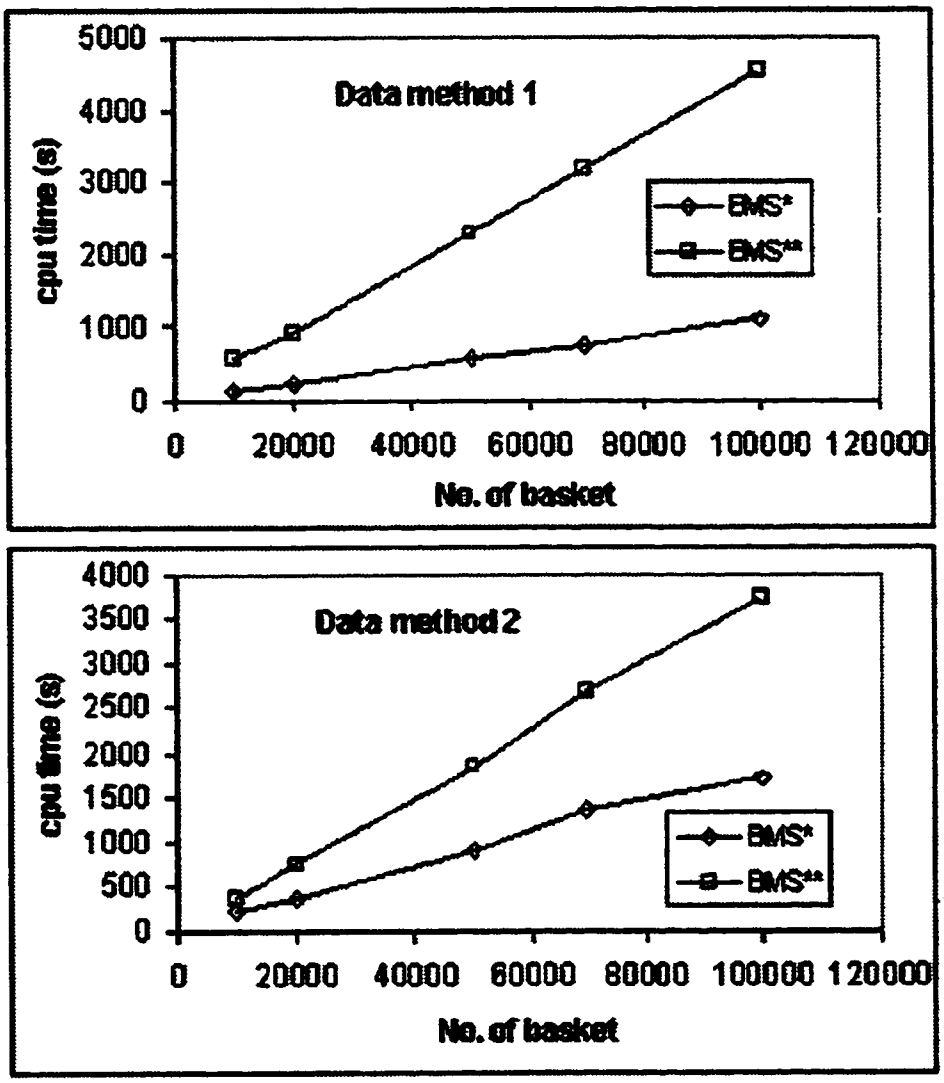

Figure 18: Effect of Transaction Number with Monotone and Succinct Constraints for Minimal Valid Answers

always performs better than BMS+, for minimal valid answers both algorithms BMS* and BMS** are affected by the selectivity. When the selectivity is below 20%, BMS** has better performance. Above this point, the performance for BMS* becomes better. In Figure 18, we deliberately show this situation when the selectivity is unfairly high at 50%. Figure 19 shows the cross-over point.

3.4.3 Summary and conclusions

From all the experiments, it can be concluded that for BMS+, the constraint type does not influence the overall performance. Algorithm BMS* is slightly affected by the selectivity. With the increase of the selectivity, the cpu usage of BMS* decreases. The performance of algorithm BMS** is heavily dependent on the constraint for pruning. So when the constraint selectivity is low, BMS** has better performance than BMS+, especially when the constraint is both anti-monotone and succinct. When constraint selectivity is high, BMS** does not perform well, it can become 2 to 3 times slower than BMS+ in the worst case. BMS++ shows the best performance under all circumstances. When constraint selectivity is low, BMS++ relies on the constraint to prune the candidate itemsets, and when the constraint selectivity is high, it relies on the upward closed property of being correlated to do the pruning.

3.5 Related Work

In this chapter, only Brin *et al.*'s work [BMS97] and the constrained frequent set framework of [NLHP98, LNHP99] are compared. Brin *et al.* [BMS97] defined their answer set as minimal correlated and CT-supported sets. They claimed that this completely characterizes the solution space. Technically, this is true only when one also returns, as part of the answer, some description of the upper border (in their case, CT-support border). When different kinds of constraints are added to this framework (for example, monotone, anti-monotone, *etc.*), a proper understanding of the solution space is needed before we can even advance minimal sets as a meaningful definition of the answer set. Notice that simply returning minimal sets does not completely cover all answers, unless it is clear where the upper border is. On the other hand, when the solution space is a single region (this is the case when all constraints are monotone or anti-monotone), there is an intuitive appeal to return minimal answers, as they are

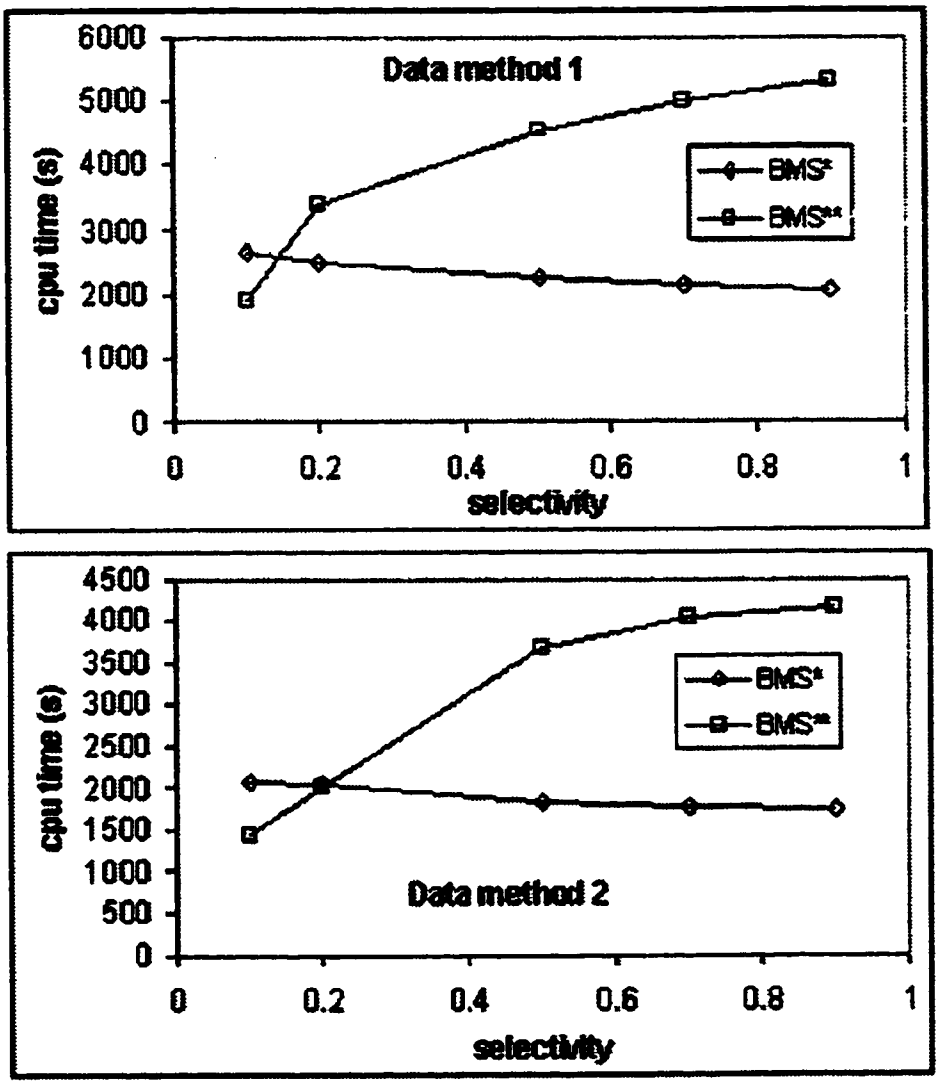

Figure 19: Effect of Selectivity with Monotone and Succinct Constraints for Minimal Valid Answers

in some sense the "smallest objects" in the solution space.

In [NLHP98, LNHP99], monotonicity is not exploited, since the answer sets in the studies are *all* frequent valid sets. This is in keeping with the classical framework of associations, where all frequent sets are computed (and used for forming associations). Handling both monotone and anti-monotone constraints is a novelty to this work. One may argue that [BMS97] has already handled such constraints (since being correlated is monotone and being CT-supported is anti-monotone). Still, in this work, the interaction between monotone and anti-monotone with succinct constraints is handled, which to the best of my knowledge, is done for the first time.

Chapter 4

Data Mining on Circumstance Lattice

In Chapter 2, the new data mining model on bilattice is proposed. Chapter 3 explains the extensive work of correlations on itemset lattice. In this chapter, we will change our focus to circumstance lattice, and try to determine the circumstances under which a given pattern (or a set of patterns) holds in a large database [GLW01]. So far in data mining area, the most widely used threshold is frequency. The frequent itemsets are the basis of a lot of patterns such as associations and correlations. Therefore instead of using correlations for consistency from Chapter 3, frequency is chosen as the given pattern on circumstance mining for the purpose of generality and simplicity.

4.1 Motivation and Background

4.1.1 Motivation

As mentioned in Chapter 1, all of the data mining studies up to date are concerned with answering the question "which patterns hold in the database." However many of a time, it is at least as interesting and useful to ask "*when* a given pattern holds in a database". Here the word *when* can be interpreted to mean, but need not be restricted to, temporal conditions, conditions involving locations, demographics, *etc.*, and more generally, the "underlying circumstances" under which the pattern of interest holds in a database.

But still why should people care about this question? Here are a few motivating

example queries.

Q_1: Under what circumstances (say, on time of purchase) is the itemset {beer, diaper} bought frequently?

Q_2: Under what circumstances (e. g. time, location) are interstate calls totaling over \$50 made by a customer?

Q_3: Under what circumstances (e. g. season, weather) are ice cream sales high? Which items(ets) have high sales under those circumstances?

Q_4: Which itemsets are seasonal favorites, *i.e.* they are bought frequently in fall but not in summer or vice versa?

Q_5: For each itemset bought frequently in fall, find further circumstances under which they are bought frequently.

Just as associations, correlations, and other mined patterns are useful in sales planning *etc.*, knowing the circumstances under which one or more patterns hold can be useful in "tuning" the plans around these circumstances. For example, if certain patterns hold for NJ locations and not for NY locations, then a manager might want to incorporate this in his planning. The examples above illustrate that one might be interested in a specific pattern (for example, a specific frequent itemset {beer, diaper} in Q_1), or in a set of patterns generated from another query (for example, the set of frequent itemsets bought in fall in Q_4), and ask for further circumstances under which they hold.

The problem that addressed in this chapter is to find all circumstances in which a given pattern holds for a given itemset, when the pattern satisfies some monotonicity properties. For the sake of concreteness, frequency is picked as a prototypical example. Since it is a c-monotone pattern, knowing an itemset is frequent under a circumstance tells us it is frequent in all circumstances implied by it. Thus, the real goal is to find the *minimal* circumstances under which a given itemset is frequent, in the sense that the itemset is not frequent in any stronger circumstance. For c-anti-monotone patterns such as *purchasecap*, the set of *maximal* circumstances in which the pattern holds w.r.t. a given itemset summarizes the whole space of circumstances where this pattern holds for that itemset.

4.1.2 Theoretical background and relations with datacube

First, it is interesting to notice that the circumstance lattice bears a strong similarity to the lattice used for data cube computation, which once again illustrates that data mining is closely related to datacubes. Indeed, when we limit ourselves to circumstances involving only equality from its definition in Chapter 2, the circumstance lattice is identical to the lattice corresponding to the (instances of) various group-bys. This suggests any of the algorithms developed for cube should be useful for mining circumstances. Actually, applying data cube algorithms on circumstance lattice will give us the computation result of the whole circumstance lattice. On the other hand, if absolute support count is used as the chosen pattern, a traditional Apriori-style level wise pruning strategy should be relevant as well, except the circumstance lattice is used to replace the itemset lattice. This interesting connection among circumstance mining, datacube and Apriori-like algorithms will be further discussed later in Section 4.3.

Another interesting observation is that for certain patterns, if they hold under a given circumstance c, they necessarily hold in a weaker circumstance c', where weaker means c implies c'. As an example, consider frequent itemset as a pattern. Let c be the circumstance `city` = 'westfield' & `state` = 'nj' & `month` = 'october' and c' be `state` = 'nj'. Clearly, c' is weaker than c in the sense that every transaction that satisfies c necessarily satisfies c'. Equivalently, c is stronger than c'. Let S be any itemset. Suppose S is frequent under c. Then it must be frequent under c'. [1] In the terminology of Ng et al. [NLHP98, LNHP99], such patterns is called *monotone*. For these patterns, there is an intrinsic interest in determining the "strongest" circumstances under which they hold. For example, suppose the set of strongest circumstances C under which an itemset S is frequent is given. Intuitively, this covers the space of all circumstances under which S is frequent, since S is frequent exactly under those circumstances that are weaker than some circumstance in C. Another example of a pattern that has this property occurs in (Q2): if interstate calls totaling over $50 are made under a given circumstance (for example, over a week), then under a weaker circumstance (for example, over a month containing that week), the total cannot be less.

[1] here frequency is the absolute support count. This count can come from a specified fraction, say 0.1%, of the *total* database size, for example.

Some patterns tend to be *anti-monotone* in that whenever they hold in a circumstance, they necessarily hold in a stronger circumstance. A natural example is the pattern "interstate calls made total under $25." Clearly, if this holds for a given circumstance, it must hold for any stronger circumstance[2]. For such patterns, the intrinsic interest is to ask for the weakest circumstances under which they hold.

4.2 A Special Circumstance – Time

In this section, a special circumstance attribute – time is studied. Let's first define the notion of a *basic time window*, which is the smallest granularity of time the user cares about, and assume this is specified by the user. For example, the transactions could be by the minute, while the basic time window might be an hour or a day. So the problem becomes: given an itemset S, find all time intervals under which S is frequent, w.r.t. a user-specified support threshold s. Notice that the support of S over an interval is always greater than or equal to that over any of its subintervals. Thus, if S is frequent in an interval I, obviously it is frequent in all intervals including I. Hence, it is interesting to ask, "what are the *minimal* intervals, in integral multiples of the user-specified basic time window, in which S is frequent?". An efficient algorithm is given for answering this question. The algorithm is applicable to any circumstance attribute whose domain is totally ordered. In the algorithm of Figure 20, it is assumed the transactions are ordered by time. If they are not, they should be sorted first.

Call an interval $I = [\ell, r]$ left-extreme provided for a fixed right endpoint r, ℓ is the closest left endpoint such that S is frequent in $[\ell, r]$. Right extreme intervals are analogously defined. The algorithm consists of 3 passes. In the first pass, the count of S in each interval of unit length u supplied by the user, is computed. In the second pass, a forward sweep is made, from the first transaction to the last, and all left-extreme minimal intervals in which S is frequent are found. In the third pass, a backward sweep is made and all right-extreme minimal intervals is found where S is frequent. The correctness of the algorithm follows these observations, since every minimal interval in which S is frequent must be either left-extreme or right-extreme. It is possible to avoid the first pass of the algorithm, and roll the count computation into each of the subsequent passes. Also, as long as the (set of transactions corresponding

[2]The strongest circumstace is *false*, which verifies this pattern vacuously.

to the) intervals scanned by the algorithm fit in main memory, the number of passes remains the same even for disk resident data.

4.3 Algorithms for General Circumstance Mining

In this section, simple adaptations to *some* existing cube algorithms are discussed, and a new algorithm is proposed to find minimal circumstances under which a given itemset is frequent.

As mentioned in Subsection 4.1.2, due to the similarity of data cube lattice and circumstance lattice, all the datacube algorithms can be used to compute the circumstance lattice. The representative cube algorithms that picked up here are the BUC (bottom-up cube) algorithm proposed by Beyer and Ramakrishnan [BR99] and the chunk array-based algorithm (which is top-down) due to Zhao et al. [ZDN97]. Of these, BUC is also designed to incorporate constraints on groups such as $count(*) > n$, which says the group must have at least n elements. However, the express goal of BUC is to find *all* groups satisfying given constraints (as opposed to minimal ones). On the other hand, the main efficiency of chunk array-based algorithm comes from using a multi-dimensional address space for storing the base table and the cube itself. This affords great compression. It uses the minimum weight spanning tree data structure to optimize cube computation. In addition, it also uses techniques for handling sparse data. Figure 21 shows the adapted chunk array algorithm for finding minimal circumstances in which a given itemset S is frequent. The basic idea is to perform a search in the circumstance lattice from the strongest circumstances to the weaker ones. As soon as S is found to be frequent in a circumstance, it is garanteed to be minimal. Clearly, all minimal circumstances can be found in this way. The correctness follows from these observations.

Figure 22 shows the original BUC algorithm and Figure 23 shows the adaptation to it for finding minimal circumstances. In the original BUC algorithm, "numDims" represents the total number of dimentions. "OutputRec" represents the current output record. And "dataCount[numDims]" stores the size of each partition. There is a minor modification consists in BUC adaptionin in computing the support of S in every group (i.e. circumstance). Just like original BUC, whenever S is found to be

```
Algorithm Stretch-Slide-Shrink;
Input: a set of transactions F, ordered by time, a basic window width u, a min. support threshold s, and a specific
itemset S;
Output: the set of all minimal time intervals I such that S is frequent in I;

1 make one scan of F and compute count(c), the no. of hits for S in c, for each consecutive window c of size u;
  initialize c to the first window of size u; make a forward sweep as follows:

2 repeat {

  2.1 if (count(c) > s)
          add c to output;
  2.2 else { // need to stretch;
    2.2.1 repeat {
              stretch(c);
          until ((count(c) > s) or (c.right = lastTrans.time)); } //stop stretching;
    2.2.2 if (count(c) > s) { //try shrinking;
              c' = shrink(c);
              while (count(c') > s) {
                 c = c';
                 c' = shrink(c); }
              add c to output; } } //can't shrink c any more;
  2.3 slide(c); //go to next window;
      until (c.left = lastTrans.time - u); } //all windows have been examined;

    make a backward sweep from the last transaction to the first, to find remaining minimal intervals;

procedure stretch(c);

    if (c.right != lastTrans.time) {

        let d be the window immediately to the right of c, i.e. d.left = c.right and d.right = c.right + u;
        count(c) = count(c) + count(d);
        c.right = d.right; }

//stretch(c) stretches current c by one window of length u to the //right and adds up the appropriate count;
function shrink(c);
local var c': window

    c'.left = c.left + u;
    return(c);

//shrink(c) shrinks current c by one window of length u from the left;
procedure slide(c);

    c.left = c.left + u;
    if (c.right == c.left) { c.right = c.right + u; }
```

Figure 20: Pseudo Code for Algorithm for Finding Minimal Intervals Where an Itemset is Frequent.

Algorithm Modified Chunk Array;
Input: an itemset S, a support threshold s, and a transaction database $\mathcal{D}$;
Output: the set of minimal circumstances in which S is frequent;

1. run chunk array algorithm, with the following modifications:
   (a) no aggregate computation (other than verifying the support constraint) is needed;
   (b) when processing a circumstance ψ, compute the support of S in ψ; if it exceeds s, add ψ to the output; circumstances that are implied by ψ are pruned;

Figure 21: Adaptation to Chunk Array Algorithm for Finding Minimal Circumstances.

infrequent in a circumstance, all stronger ones are pruned away. A final modification consists in post-processing: BUC (modified as above) finds all circumstances in which S is frequent; for circumstance mining, it is necessary to efficiently detect and eliminate non-minimal ones. Correctness of modified BUC is straightforward.

Post-processing: Circumstances are processed in increasing order of their length (for example, $A = a_1 \wedge B = b_1$ is of length 2) in BUC. In addition, within each length, make sure they are processed in lexicographical order (by assuming some arbitrary, but fixed order on circumstance attributes and their domains). Let n be the number of circumstance attributes. Then all circumstances of length n found by the modified BUC algorithm are minimal, by definition. For each circumstance of length $k < n$, we check if it is implied by any circumstance of length $k + 1$. The implication test is syntactic and checking length $(k + 1)$-circumstances for implication can be done efficiently using binary search.

The last algorithm, *minCirc* which proposed for circumstance mining is in some sense inspired from Apriori. The basic idea is that since frequency is c-monotone, it is possible to scan the transactions repeatedly and compute the support of S corresponding to all circumstances of a given length in each iteration, starting from length 1. When S is infrequent in a circumstance, prune away all circumstances of which it is a "prefix". For example, when S is infrequent in $A = a_1 \wedge B = b_1$, ignore $A = a_1 \wedge B = b_1 \wedge C = c_1$, etc. A literal implementation of this idea, however, can lead to a poor performance. The reason is when several circumstance attributes are presented, each with a domain of reasonable cardinality, the effort required for candidate generation (even with this pruning) is appreciable. This algorithm instead imposes an order on the dimensions and exploits this order in quickly pruning away many candidate circumstances early. As an example, if S is found to be frequent in

Algorithm BUC(input, dim);
Input: the relation to aggregate, dim: the starting dimension for this iteration
Output: One record that is the aggregation of input. Recursively, outputs CUBE(dim,..., numDims) on input (with minimum support).

```
1  Aggregate(input); //Place result in outputRec

2  if input.count() == 1 then //Optimization
   WriteAncestors(input[0], dim); return;

   2.1 write outputRec;
   2.2 for d = dim; d ¡ numDims; d++ do
      2.2.1 let C = cardinality[d];
      2.2.2 Partition(input, d, C, dataCount[d]);
      2.2.3 let k = 0;
      2.2.4 for i = 0; i < C; i + + do //For each partition
         2.2.4.1 let c = dataCount[d][I]
         2.2.4.2 if c >= minsup then //The BUC stops here
         2.2.4.3 outputRec.dim[d] = input[k].dim[d];
         2.2.4.4 BUC(input[k...k+c], d+1);
         2.2.4.5 end if
         2.2.4.6 k+ = c;
      2.2.5 end for
      2.2.6 outputRec.dimd = All;
   2.3 end for
```

Figure 22: Algorithm BottomUpCube(BUC).

Algorithm Modified BUC;
Input: an itemset S, a support threshold s, and a transaction database db;
Output: the set of minimal circumstances in which S is frequent;

1. run BUC algorithm, with the following modifications:
   (a) no aggregate computation (other than verifying the support constraint) is needed;
   (b) when processing a circumstance ψ, compute the support of S in ψ; if it is below s, circumstances that are imply ψ, including ψ, are pruned; otherwise, ψ is retained;
2. do post-processing to eliminate non-minimal circumstances;

Figure 23: Adaptation to BUC Algorithm for Finding Minimal Circumstances.

$A = a_1 \wedge D = d_1$, where D is the last circumstance attribute, there is no need to check "suffixes" of this circumstance. Figure 24 shows the algorithm. It uses the following notions.

Suppose $D_1, ..., D_n$ is a chosen order on the circumstance attributes and and let c and d be values of any circumstance attributes. Then $c \prec d$ iff c is a D_i-value and d is a D_j-value, for some $i < j$; c and d are said to be incomparable otherwise. Sometimes it is convenient to write circumstances such as $A = a_1 \wedge B = b_1 \wedge C = c_1$, simply as $a_1b_1c_1$, relying on the fixed chosen order of circumstance attributes. Linked lists are used with heads and nodes (head is not a node). For a linked list, its length represents the number of nodes it has. Each list represents a set of candidate circumstances of a given length. For example, a list with head a_1b_1 and nodes $c_1, \ldots, c_5, d_2, d_6, d_7$ (where $c_i \in dom(C)$ and $d_j \in dom(D)$) represents the set of circumstances $a_1b_1c_1d_2, a_1b_1c_1d_6, \ldots, a_1b_1c_5d_7$. A circumstance $x_1 \cdots x_k$ appears in a linked list provided its head contains $x_1 \cdots x_{k-1}$ and one of its nodes contains the circ x_k. In the algorithm, a circumstance is frequent in which S is frequent.

The last step of the algorithm invokes post-processing, as discussed earlier. In particular, since the circumstances are processed in increasing order of length, the same post-processing as used for BUC can be used. The correctness of the algorithm follows from the following facts: (i) a circumstance is never explicitly pruned unless S is infrequent in it; (ii) circumstances that are suffixes of those ending in the last circumstance attribute value are never considered for counting, for example, if circumstance attributes are ordered as $ABCD$, then there is no need to consider suffixes of b_1d_1; every suffix of such circumstances are covered by some circumstances that already considered somewhere else; for example, the suffix $b_1d_1c_1 \equiv b_1c_1d_1$, and one of its (not necessarily proper) prefixes, for example, b_1, is always considered by the algorithm.

4.3.1 Relevance to CUBE

A significant feature of the chunk array and BUC algorithms (with modifications) is that they can be used to compute not only (minimal) circumstances where a pattern holds for an itemset, but also any required aggregate measure value at each such circumstance. Then what about algorithm minCirc, inspired by Apriori-style intuition? It turns out that by associating a field for computing the aggregate measure with

Algorithm minCirc;
Input: an itemset S, a support threshold s, and a transaction database $\mathcal{D}$;
Output: the set of minimal circumstances in which S is frequent;

1 choose some circ attribute order, say $D_1, ..., D_n$;

2 scan $\mathcal{D}$ once and obtain counts of S in all atomic circumstances, i.e. in all D_1-circumstances, ..., D_n-circumstances;

3 let F_1 be the set of all atomic circumstances in which S is frequent;

4 hash the circumstances and write the counts;

5 create a linked list of candidates as follows:

5.1 for ($1 <= i <= n; i++$) {
5.1.1 let $x_i \in F_1$ be any D_i-value;
5.1.2 create a linked list with head x_i;
5.1.3 for ($i < j <= n; j++$) {
let $x_j \in F_1$ be any D_j-value;
append a node containing x_j to the linked list with head x_i; } }

5.2 $k = 2$;

5.3 while the current set of linked lists is non-empty {
5.3.1 scan $\mathcal{D}$ and obtain counts of all k-circumstances appearing in the linked lists;
5.3.2 purge each node corresponding to a circumstance in which S is infrequent;
5.3.3 construct linked lists for $(k+1)$-circumstances (candidates) as follows:
5.3.4 for each linked list L with head H of size k {
if ($length(L) >= 2$) {
for each node (containing a circ) x in L {
temporarily create a new list with head Hx, which contains all nodes of L after node x, that correspond to later circumstance attributes;
//since length(L) ¿= 2, there will be at //least one such node;
delete from this list those nodes y such that some k-subset of Hxy does not appear in any linked list with head size k;
//this test can be performed quickly by sorting;
} } } }

6 eliminate non-minimal circumstances via post-processing;

Figure 24: A Linked List Based Mining Algorithm for Finding Minimal Circumstances.

each node, the aggregate can be computed incrementally. This argument holds for all distributive and algebraic aggregate functions, for which the most known fast cube algorithms have been developed. This establishes an interesting tight connection between frequent set mining and cube computation. This framework is applicable not only just to frequent sets but also to more general itemset patterns with monotone or anti-monotone properties w.r.t. itemsets or circumstances.

4.4 Experimental Results

A series of experiments are conducted to evaluate the effectiveness of the algorithms presented for mining minimal circumstances in which an itemset is frequent. More specifically, the three algorithms are compared in order to find out how they perform under different conditions. For the given three algorithms, BUC is a bottom up algorithm and uses a "height first" principle; minCirc is also a bottom up algorithm but uses level wise principle; Chunk Array by nature is a top-down algorithm and it also uses the level wise principle. So these three algorithms represent three different ways in mining minimal circumstance sets from different directions of the circumstance lattice, and using different search strategies.[3] The performance of the algorithms are compared not only by total cpu time, but also by the number of candidate sets that each algorithm generated and tested.

4.4.1 Test data

A Pentium II 200 processor with 64 KB memory is used for all the experiments. All the algorithms are implemented in C++ running on a Windows NT platform. Test material is synthetic sets of transactions generated for various cases. A Zipf distribution is used to control the skewness of the data, as commonly used in experiments of this kind, see e. g. [BR99]. To be fair to all the algorithms, both cases where the transaction database fits in main memory and resides on disk are tested. 1 to 7 circumstance attributes are used, and each attribute had a domain of cardinality 100. The number of transactions varies form 10,000 to 500,000, the support threshold is from 5 to 1000, and skewness is varied from 0, which is the uniform distribution,

[3]The fourth way would be top-down, depth-first, which is BUC applied to an i-anti-monotone pattern.

to 3. In each of the experiments, the number of candidate sets generated, the peak memory allocation, the total time, the I/O time, and the number of passes over the transaction database are recorded for analysis. However, for the sake of space and simplicity, not all of the experiment results are showed.

4.4.2 Experimental evaluation

The first set of experiments is conducted where transaction databases can fit in main memory. All other parameters are varied as described above. The total cpu time is measured as a function of the number of transactions. We have chosen 5 circumstance attributes, threshold 100, and skewness 1, as a typical representative of this set of experiments. The results are shown in Figure 25. It is fairly uniform. With the increase of the number of transactions, there is a linear increase in the total time for all three algorithms. But the slope for minCirc is steeper than the slope for BUC and Chunk Array. This is due to the fact that BUC and Chunk Array make only one pass over the transaction database, whereas minCirc makes up to as many passes as the number of circumstance attributes. Other results that have been observed from this set of experiments are: with the increase of support threshold, the number of candidates generated and the total time are decreasing for both BUC and minCirc. For Chunk Array, on the other hand, with the increase of the support threshold, the number of candidate sets generated is slightly increasing and the total time is almost constant. When skewness is increased, the total time for both BUC and minCirc is increasing while the total time for Chunk Array is decreasing. This is because Chunk Array works top-down, the number of atomic circumstances (the ones directly above *false*) will dominate the number of candidate sets generated, and the total time will mainly be affected by the number of circumstance attributes and their cardinalities, once the number of transactions is fixed. As the data becomes more skewed, the number of atomic circumstances that need to be considered decreases. The overall performance nevertheless only increases slightly..

The second set of experiments conducted is on disk resident transaction databases. When the data is uniformly distributed (skewness = 0), minCirc gives the best performance among the three algorithms. Figure 26 displays the results for support threshold at 100, and number of circumstance attributes at 5. With the number of transaction increases, the total time increases rapidly for BUC. When the number

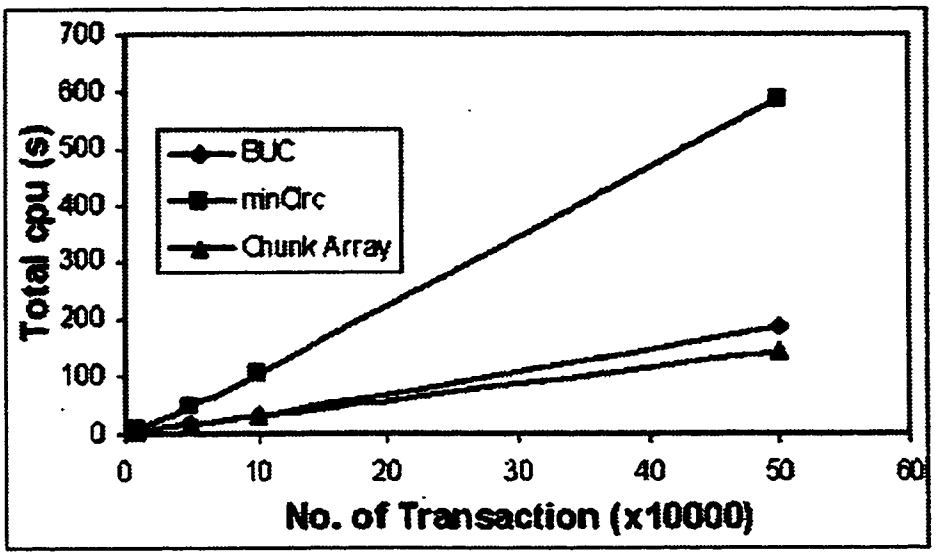

Figure 25: Algorithms Comparison (*memoryresident*, *skewness* = 1, *threshold* = 100, $No.of Circ. = 5$)

of transaction is equal to 500,000, the total time for BUC is 50 times larger than minCirc. Even for Chunk Array the total time is 5 times bigger than minCirc. For minCirc, the upper bound of the number of passes over the transaction database equals to the number of circumstance attributes. On the other hand, for BUC the only upper bound is the size of the circumstance lattice. Furthermore, as the data is uniformly distributed, Chunk Array has to do much more computations for each of the atomic circumstance sets, so minCirc also performs better than Chunk Array.

However, with the increase of the skewness, as the data is not uniformly distributed, the performance of Chunk Array will improve quite drastically since some of the atomic circumstance sets will not appear in the transaction database. Figure 27 gives the results for the same condition as Figure 26, except that the skewness is equal to 1. Similar performance can be observed for BUC and minCirc. Under these conditions, Chunk Array is the algorithm of choice. Note that the time scale is different in Figure 26 and Figure 27.

Another factor that affects the performance of these algorithms, especially Chunk Array, is the number of circumstance attributes. When the number is 5, and the domain of each circumstance attribute has cardinality of 100, there is potentially 10^{10} atomic circumstance sets appearing in the transaction database. If increasing the number of circumstance attributes to 6, there might be up to 10^{12} atomic circumstance

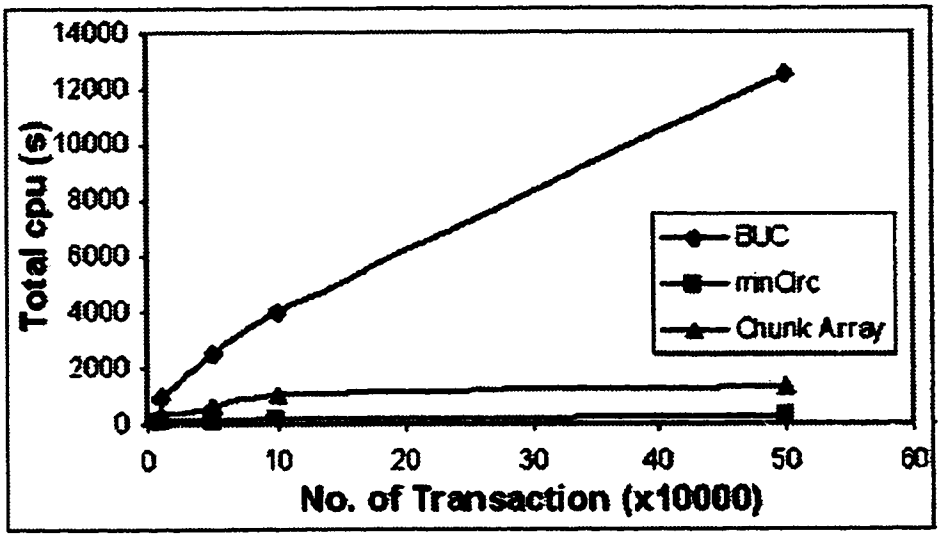

Figure 26: Algorithms Comparison (*diskresident*, *skewness* = 0, *threshold* = 100, $No.ofCirc. = 5$)

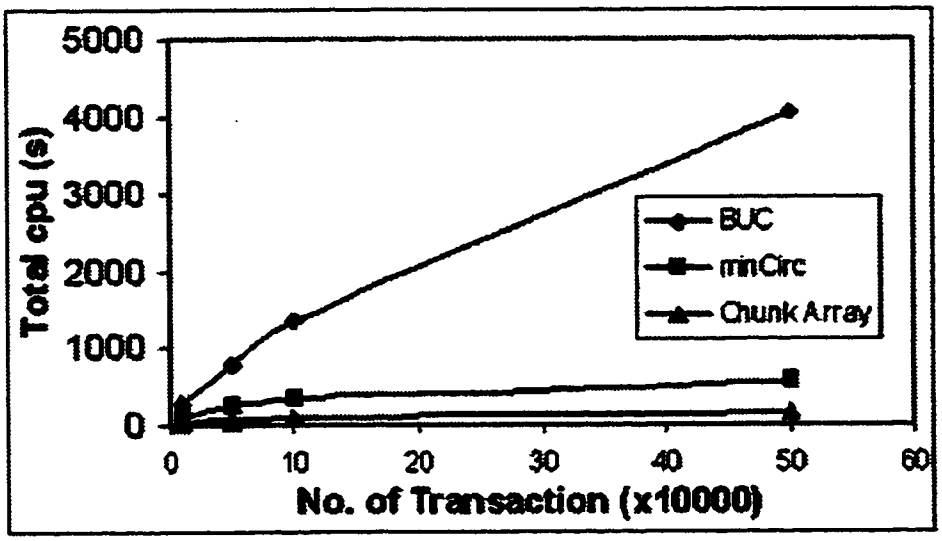

Figure 27: Algorithms Comparison (*diskresident*, *skewness* = 1, *threshold* = 100, $No.ofCirc. = 5$)

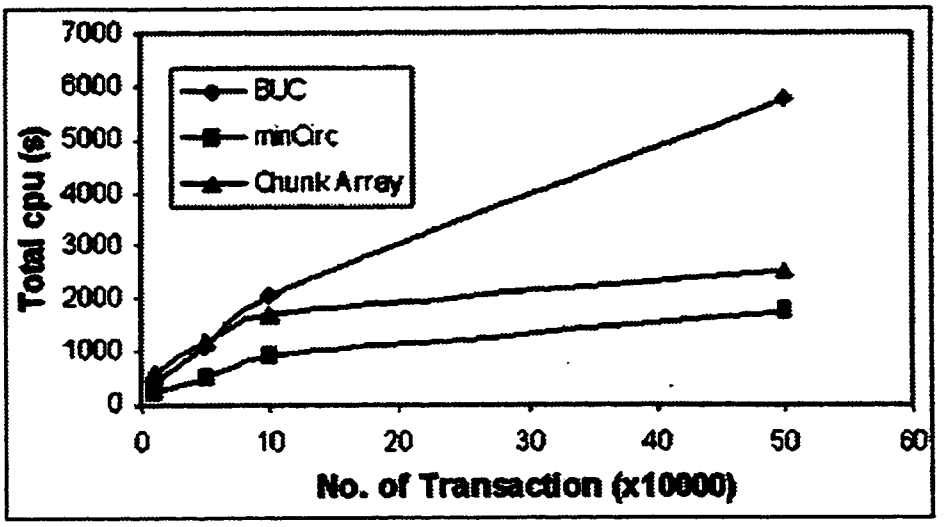

Figure 28: Algorithms Comparison ($diskresident, skewness = 1$, $threshold = 100, No.ofCirc. = 6$)

sets. This is a large increase for Chunk Array, but for minCirc this just implies at most one more pass over the transaction database. For these conditions minCirc becomes the best choice. Figure 28 shows the results for 6 circumstance attributes. When the number of transactions is smaller than 300,000, the total time for BUC and Chunk Array is similar, and both about twice the total time of minCirc. When the number of transactions increases to 500,000, the total time for minCirc is 5 times smaller than the other two algorithms.

So it is interesting to observe the effect of varying number of circumstance attributes. Figure 29 shows such an experiment. When the number is smaller than 5, Chunk Array has the best performance and the slope of the curve is very gradual. However, once the number of circumstance attributes is greater than 6, there is a dramatic decrease in the performance of Chunk Array and it becomes the worst among the three algorithms. With the increase of the number of circumstance attributes, the total time for both BUC and minCirc increase much more slowly compare to Chunk Array.

In the third and last set of experiments the effect of the skewness of the data is tested. In Figure 30, when the data is uniformly distributed and is disk resident, minCirc has the best performance and BUC gives the worst performance. When skewness is greater than 1, the curve for all three algorithms become flat. Chunk Array

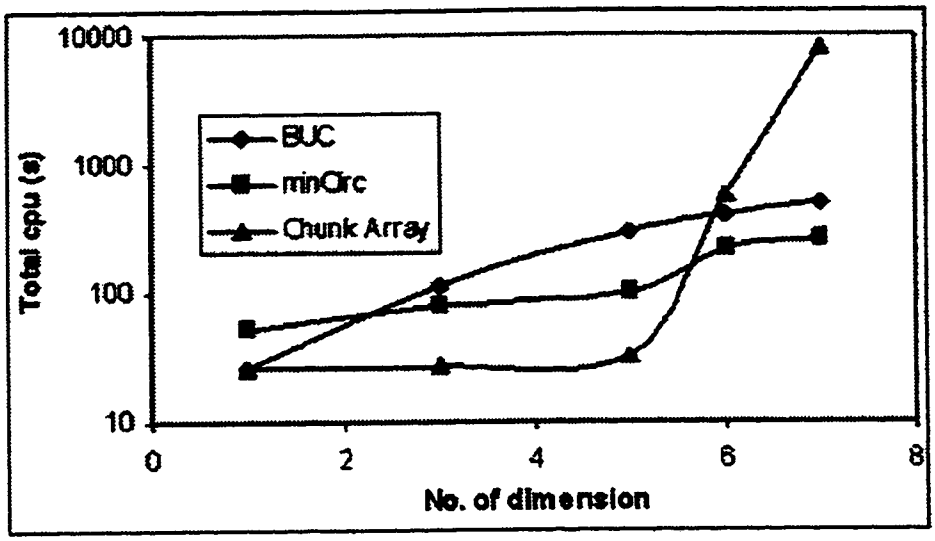

Figure 29: Algorithms Comparison ($diskresident, skewness = 1$, $threshold = 100, No.of Trans. = 100000$)

shows the best performance while BUC is still the worst one. However, in another experiment, as showed in Figure 31, main memory resident transaction databases is tested, while keeping the other parameters the same as in Figure 30, it is observed that BUC is almost unaffected by the skewness of the data, and gives the best performance among the three algorithms.

In other experiments in this set, the number of candidate sets generated is tested. This measure is not affected by the residency of the transaction database. The skewness of the data does affect the performance, but slightly. Two experiments, namely Figure 32 and Figure 33 are showed for the number of candidate sets generated vs. the number of transactions. Skewness is at 1, and the number of circumstance attributes is 5. Figure 32 shows the result for the threshold at 1000, and Figure 33 shows the result for the threshold at 5. When the threshold is high, the pruning strategies for BUC and minCirc are very effective, so at 500,000 transactions the number of candidate sets minCirc is over 500 times smaller than that of Chunk Array, and BUC generates 80 times fewer candidate sets than Chunk Array. However, when the threshold is small, the pruning strategy for BUC loses its efficiency, and the number of candidate sets becomes the biggest. minCirc also lose some of its pruning efficiency, but it still comparable to Chunk Array.

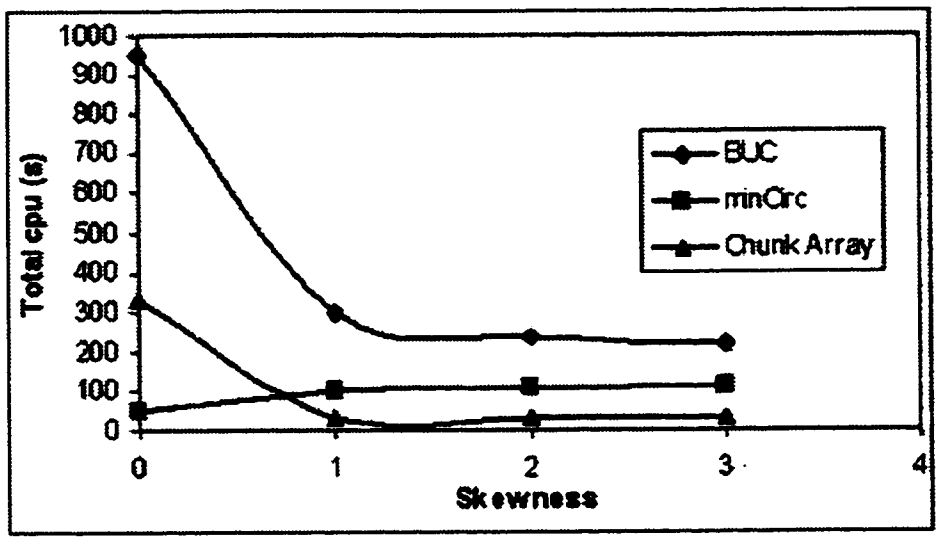

Figure 30: Algorithms Comparison (*diskresident*, $No.ofCirc. = 5$, *threshold* = 100, $No.ofTrans = 100000$)

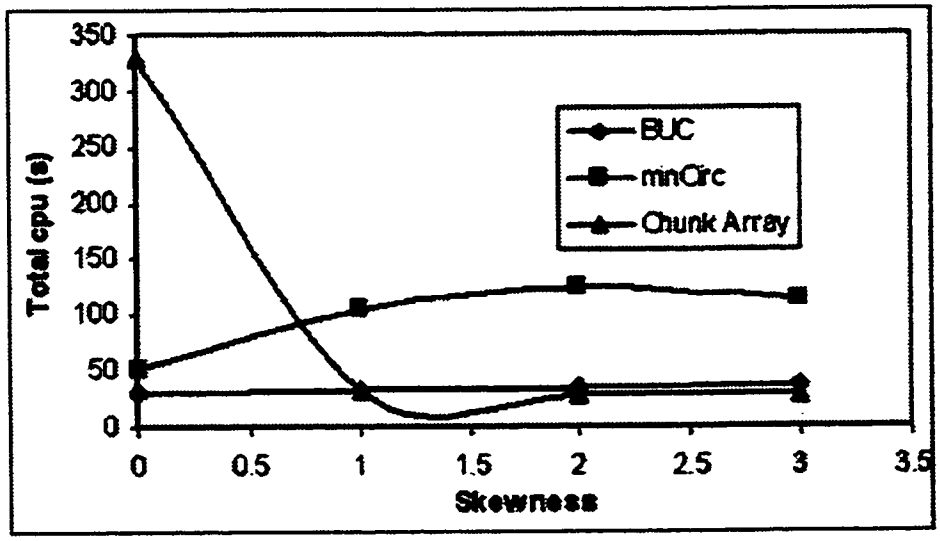

Figure 31: Algorithms Comparison (*memoryresident*, $No.ofCirc. = 5$, *threshold* = 100, $No.ofTrans = 100000$)

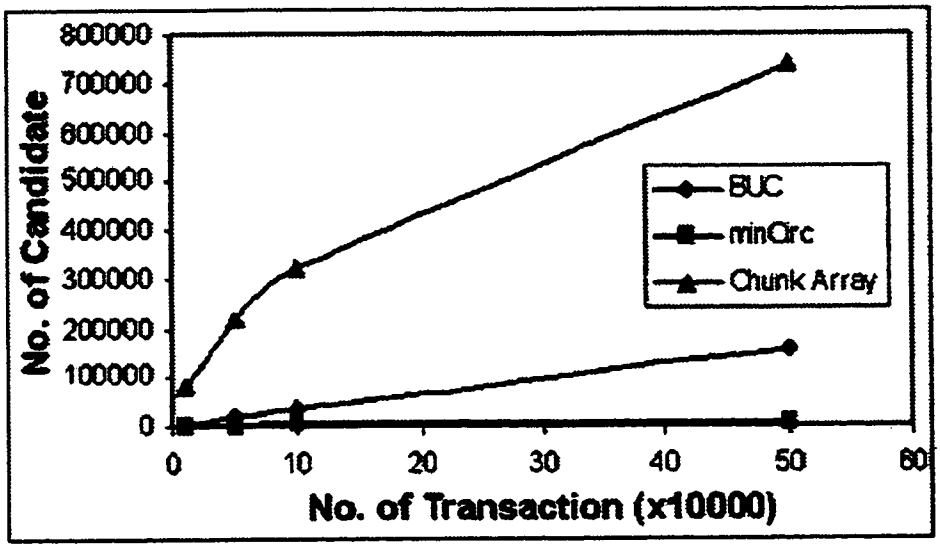

Figure 32: Algorithms Comparison (*memoryresident*, *skewness* = 1, *threshold* = 1000, $No.ofCirc. = 5$)

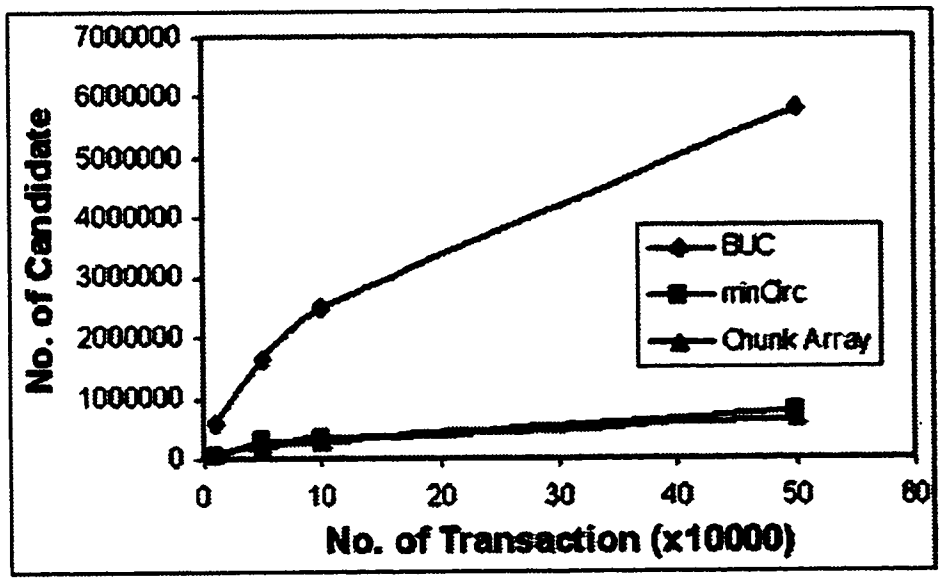

Figure 33: Algorithms Comparison (*memoryresident*, *skewness* = 1, *threshold* = 5, $No.ofCirc. = 5$)

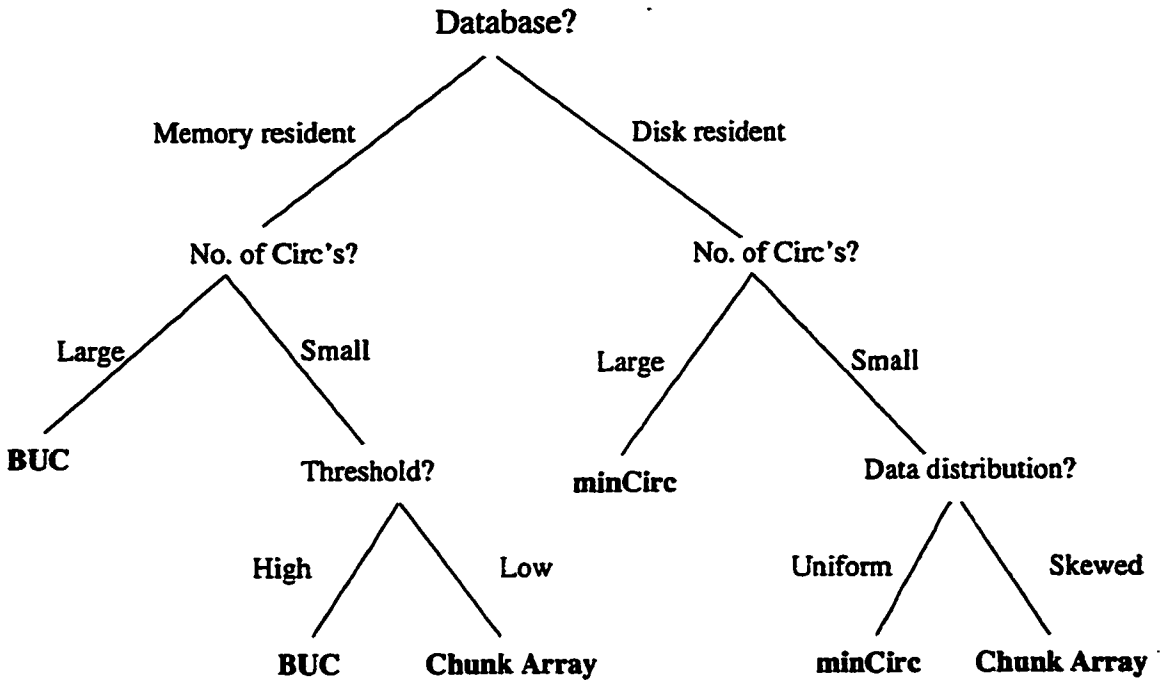

Figure 34: Decision Tree for Choosing a Mining Algorithm.

4.4.3 Summary and conclusions

The prescriptions emerging from our experiments are described in the decision tree of Figure 34.

4.5 Related Work

As previously noted, mining for circumstances in which a given pattern (like frequent itemset) holds has a close relationship with cube computation, with or without constraints, and hence the huge body of techniques and algorithms developed for cube is relevant. Among all these works, the recent paper [BR99] surveys many of them, and also describes the BUC algorithm.

Chapter 5

Data Mining on Bilattice

So far, the work that has been done only concerns with on one lattice. In Chapter 3, constraints are introduced on correlations for itemset lattice mining. In Chapter 4, algorithms are provided to find circumstances where a pattern holds for a given itemset. This chapter will introduce data mining on bilattice.

5.1 Introduction and Motivation

Mining is exploratory in nature. An analyst might wish to find the circumstances under which a pattern holds for an itemset and then find out which *other* sets satisfy the pattern under those circumstances (Query Q_3 in Chapter 4 Section 4.1.1 is an example involving such exploration). Or s/he might start with a circumstance (say, fall in northeast), find the itemsets that satisfy a pattern, and then attempt to characterize the circumstances under which these latter sets satisfy the pattern (e.g., see query Q_5 from the same section as above). Queries of this kind require to be able to freely move between the worlds of itemsets and circumstances. The bilattice of itemsets and circumstances developed in Chapter 2 is ideal for supporting such "migration". In this chapter, the notion of an *Armstrong Basis* is proposed as a basis for supporting a variety of queries involving such migration.

5.2 Theory of Armstrong Basis

Recall the bilattice $(\mathcal{C} \times \mathcal{I}, \leq^{\text{up}}_{\text{up}}, \leq^{\text{up}}_{\text{down}})$ defined in Chapter 2, where for any two elements $(\psi, S), (\phi, T) \in (\mathcal{C} \times \mathcal{I})$, $(\psi, S) \leq^{\text{up}}_{\text{up}} (\phi, T)$ exactly when ψ logically implies ϕ and $S \subseteq T$, and $(\psi, S) \leq^{\text{up}}_{\text{up}} (\phi, T)$ exactly when ψ logically implies ϕ and $T \subseteq S$. An element (ψ, S) in the bilattice is $\leq^{\text{up}}_{\text{up}}$-*minimal* w.r.t. a property p provided it satisfies p and whenever (ϕ, T) satisfies p and $(\phi, T) \leq^{\text{up}}_{\text{up}} (\psi, S)$, we have $(\psi, S) = (\phi, T)$. Minimality w.r.t. the other order and maximality w.r.t. either order are defined analogously.

Definition 3 (Armstrong Basis) Let p be a pattern over circumstances and itemsets, and $\mathcal{D}$ be a transaction database. Suppose p is c-monotone and i-anti-monotone. Then the *Armstrong Basis* of p w.r.t. $\mathcal{D}$ is defined as $\mathcal{AB}(p_{\mathcal{D}}) = \{(\psi, S) \in (\mathcal{C} \times \mathcal{I}) \mid . (\psi, S)$ is $\leq^{\text{up}}_{\text{down}} -$ minimal w.r.t. $p\}$.

The significance of Armstrong Basis is clear from the following facts.

Proposition 2 *Let D be a transaction database, p be a c-monotone and i-anti-monotone pattern, and $\mathcal{AB}(p_{\mathcal{D}})$ be the Armstrong Basis of p w.r.t. D. Then the following statements are true.*

1. *For every element $(\psi, S) \in \mathcal{AB}(p_{\mathcal{D}})$, pattern p does not hold for any proper superset of S at circumstance ψ; similarly, p does not hold for S at any circumstance that is strictly stronger (i.e. it implies, but is not equivalent to, ψ) than ψ.*

2. *Every element of the bilattice $(\mathcal{C} \times \mathcal{I})$ which satisfies the above statement belongs to $\mathcal{AB}(p_{\mathcal{D}})$.*

3. *For every element (ϕ, T) in the bilattice, $p_D(\phi, T)$ holds if and only if there is an element $(\psi, S) \in \mathcal{AB}(p_{\mathcal{D}})$, such that $T \subseteq S$, and ϕ is logically implied by ψ.*

Proof Sketch: We will just show the proof for part 1 as an example. According to the definition of Armstrong Basis, ψ is the strongest circumstance in which $p(S)$ holds and S is the maximal itemset that $p(\psi)$ holds. If there exists a S' that is a superset of S, and $p(\psi, S')$ holds, then $p(\psi, S)$ is not an element in the Armstrong Basis, instead, $p(\psi, S')$ should be in it.

The proposition above also explains the reason behind the terminology used for this notion. Indeed, Armstrong Basis has a structure that is very similar to the notion of Armstrong couples introduced for families of functional dependencies by Armstrong, almost three decades ago [W74].

The notion of Armstrong Basis is applicable to any pattern that satisfies some monotonicity properties w.r.t. itemsets and circumstances, either by defining minimality or maximality w.r.t. the appropriate order in the bilattice. For example, for *minpurchase*, a c-monotone and i-monotone pattern, the Armstrong Basis should be defined as the set of $\leq_{\text{up}}^{\text{up}}$-minimal elements satisfying *minpurchase*. Note that for such a pattern the Armstrong Basis has the complete information about the space of circumstance-itemset pairs valid w.r.t. this pattern, as stated in part (3) of the proposition.

5.3 Algorithms for Computing Armstrong Basis

For a typical transaction database and a pattern such as frequency, the size of the Armstrong Basis can be substantial. So why should anyone want to compute it? The analogy that is given is cube. For large databases, cube is a time- and space-expensive operation. Indeed, a user may not be interested in seeing a cube in its entirety. Its utility comes from its ability to service a variety of aggregation queries involving multiple group-bys. In a similar manner, if the Armstrong Basis for a pattern is available, queries (such as those in Chapter 4 Section 4.1.1) involving migration between itemsets and circumstances can be answered efficiently. Thus, it is worth investigating efficient algorithms for its computation. Here, the c-monotone and i-anti-monotone patterns p are used. Similar notions exist for other patterns.

Let p be a c-monotone and i-anti-monotone pattern. Define a function $p_i : \mathcal{C} \rightarrow \mathcal{I}$ as $p_i(\psi) = \{S \mid p(\psi, S) \wedge \forall S' : S' \supset S \Rightarrow \neg p(\psi, S')\}$, and a function $p_c : \mathcal{I} \rightarrow \mathcal{C}$ as $p_c(S) = \{\psi \mid p(\psi, S) \wedge \forall \phi : \phi < \psi \Rightarrow \neg p(\phi, S)\}$. These functions have obvious extensions to collections of circumstances or sets: for a set of circumstances Ψ, $p_i(\Psi) = \bigcup_{\psi \in \Psi} p_i(\psi)$, and for a collection of itemsets $\mathbf{S}$, $p_c(\mathbf{S}) = \bigcup_{S \in \mathbf{S}} p_c(S)$. Note that with such extensions, the functions p_i and p_c can be composed. For example, let p be frequency. Then for a collection of circumstances Ψ, the composite function $p_c \circ p_i$ does the following: first, it finds all maximal itemsets frequent under any circumstance in Ψ; secondly for each

Algorithm ArmBase;
Input: a transaction database D, a c-monotone and i-anti-monotone pattern p.
Output: $\mathcal{AB}(p_{\mathcal{D}})$

1. the Armstrong Basis is initially empty;
2. do a level wise sweep of the itemset lattice and do the following:
3. for each set S at the current level, if S is not a subset of S' for some (ψ, S') in the current Armstrong Basis, then find the collection of minimal circumstances CIRC_S at which the pattern holds for S;
4. for each circumstance $\psi \in \text{CIRC}_S$, find the collection of maximal sets ITSETS_ψ for which the pattern holds under ψ; let ITSETS be the union of ITSETS_ψ, over all $\psi \in \text{CIRCS}_S$;
5. each pair in $\text{CIRC}_S \times \text{ITSETS}$ is an element of $\mathcal{AB}(p_{\mathcal{D}})$;
6. terminate when all levels are processed, or when no itemset at a given level makes it to $\mathcal{AB}(p_{\mathcal{D}})$;

Figure 35: Algorithm for Computing the Armstrong Basis of a Pattern in a Transaction Database

of such itemsets, it finds all minimal circumstances under which it is frequent. Similar remarks hold for other patterns. A set of circumstances Ψ is said to *cover* another set Φ provided $\forall \phi \in \Phi$, $\exists \psi \in \Psi$, such that ψ implies ϕ. Similarly, a collection of itemsets $\mathbf{S}$ *covers* collection $\mathbf{T}$ provided $\forall T \in \mathbf{T}$, $\exists S \in \mathbf{S}$, such that $T \subseteq S$. Notice that the Armstrong Basis $\mathcal{AB}(p_{\mathcal{D}})$ is itself a cover of the set of all pairs (ψ, S) for which the pattern p holds. Here are the results.

Proposition 3 *Let p_i and p_c be functions as defined above, w.r.t. a c-monotone and i-anti-monotone pattern p. Then the following holds:*

1. *for every collection of circumstances Ψ, $p_c(p_i(\Psi))$ covers Ψ; furthermore, $p_c(p_i(p_c(p_i(\Psi)))) = p_c(p_i(\Psi))$.*
2. *for every collection of itemsets $\mathbf{S}$, $p_i(p_c(\mathbf{S}))$ covers $\mathbf{S}$; furthermore, $p_i(p_c(p_i(p_c(\mathbf{S})))) = p_i(p_c(\mathbf{S}))$.*

Proof Sketch: The proposition can be proved by appealing to the definitions of p_i and p_c. It immediately suggests the algorithm in Figure 35 for computing $\mathcal{AB}(p_{\mathcal{D}})$.

This algorithm relies on the following facts: $\text{CIRCS}_S = p_c(S)$ and $\text{ITSETS} = p_i(p_c(S)$. For each pair $(\phi, T) \in (p_c(S) \times p_i(p_c(S)$, if p holds for T at a stronger circumstance, then it would hold there for S too, which contradicts the minimality of ϕ w.r.t. S. Similarly, if p held for a proper superset of T at ϕ, this would contradict

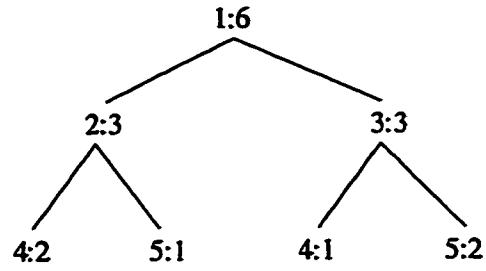

Figure 36: An Example FP-tree (support threshold assumed to be 2)

the maximality of T w.r.t. circumstances in $p_c(S)$. The correctness follows from this. Unfortunately, this algorithm can be prohibitively expensive. In particular, it does not share any effort between successive iterations. There is, of course, a dual algorithm which works off the circumstance lattice, sweeping it level-wise. It is omitted since it is very expensive as well. The question remains how people can efficiently compute the Armstrong Basis.

This problem can be solved by inspiration from the FP-tree algorithm, recently proposed by Han *et al.* [HPY00]. First, let's quickly recall their algorithm.

The FP-tree is a compact data structure for storing all information about frequent itemsets in a database (under the circumstance *true*). The idea is frequent items, obtained from an initial scan of the database are ordered in decreasing order of their frequency. In the next (and last) scan, as each transaction is scanned, the set of frequent items in it are arranged on a tree incrementally, by ensuring that prefixes among sets of frequent items contained in different transactions are maximally shared. In addition, a count is associated with each item i on this tree and it represents the number of transactions containing the itemset corresponding to the prefix terminating at i. Figure 36 shows an example of FP-tree. Note that there may be more than one node corresponding to an item on this tree. Nodes corresponding to the same item are linked together and there is a table of items with a pointer to the first item in the list. The frequency of any one item i is the sum of the count associated with all nodes corresponding to i.

The FP-tree method relies on the following principle: the support of an itemset $\alpha\beta$ in the database is exactly that of β in the restriction of the database to those transactions containing α, called the conditional pattern base of α. The idea is to use an FP-tree to represent all this conditional pattern base in turn efficiently. Once an FP-tree is constructed for the original database, process it bottom-up. For each leaf in

the current FP-tree representing some frequent item i, obtain its support by summing the counts of all nodes associated with i, then construct the conditional pattern base for this item and its FP-tree representation. This representation can be used to find all frequent patterns in this pattern base. Let $\mathcal{F}_i$ be this set, then the set of frequent itemsets in the original database that contains i are exactly $\{\alpha i \mid \alpha \in \mathcal{F}_i\}$. The exit condition for this recursion is an FP-tree containing a single path (i.e. only one branch) α. In this case, every subset of α is frequent (in the appropriate conditional pattern base).

The algorithm for Armstrong Basis is inspired from this algorithm. Before describing it, let's discuss how FP-trees can be used to efficiently extract maximal frequent itemsets. As processing the nodes in an FP-tree bottom-up, consider the FP-tree representation of the conditional pattern base associated with a pattern α as well. If this FP-tree is a single path, say β, it can be directly concluded that $\beta\alpha$ is a frequent itemset and is maximal among those containing α. However, this may or may not be a globally maximal frequent itemset. One way to ensure its maximal is to check that there is no proper superset that generated previously. A second idea, suggested by Jian Pei [P00], is to use the FP-tree algorithm to check that there are no frequent items in the conditional pattern base of $\beta\alpha$. This is clearly more efficient, since it avoids explicit subset checking. However, it incurs the overhead of checking the conditional pattern base of $\beta\alpha$ for possible frequent items.

Now, for computing the Armstrong Basis, first make a scan of the database and determine all atomic circumstances of the form $A = a_1$ (we write simply a_1 when A is understood, as before), and the frequent items they contain. Sort them according to the decreasing number of frequent items they contain. For example, if a_1 contains 5 frequent items while b_2 contains 6, a_1 comes after b_2.[1] The rationale for this order will be clear shortly. Next, scan the database again. For every transaction, incrementally build a *circumstance prefix tree* (CP-tree), using the same principle as the FP-tree, using maximal sharing of prefixes, as follows. Initially, the CP-tree is empty. For each transaction t with circumstance α, if there is a branch in the tree which shares a non-empty prefix β of α (let $\alpha = \beta\gamma$), create a new branch from the node in the tree branch where the maximal shared prefix α ends. Then create a new branch corresponding to β from this node. For example, if the circumstance α is $a_1b_2c_1$ and there is a branch

[1] It is of course possible that a_1 comes after b_2 but before b_3.

$a_1b_3c_1$, a new branch will be created at the node a_1 corresponding to b_2c_1. At each leaf of this tree (which represents a circumstance involving all circumstance attributes), a pointer is made to an (itemset) FP-tree, which represents all sets of frequent items that occur in (transactions satisfying) this circumstance. [2] As scaning the transactions, the latter trees are also incrementally built. Specifically, if the circumstance α of transaction t is fully shared by some branch of the CP-tree, then let T be the FP-tree pointed to by the leaf of this branch. (T is possibly empty.) Then we incorporate the list of frequent items in t incrementally in this tree exactly using the FP-tree method. Once complete the scan, a CP-tree will have been built where each of whose leaves points to an FP-tree, representing all sets of frequent items associated with that circumstance. Recall that the FP-trees themselves maintain count at each node. It is ensured that the counts are local to the circumstance, in that the sum of counts associated with all nodes for item i in a circumstance should be precisely the support count of i in that circumstance.

Once this composite tree is built, the Armstrong Basis can be generated as follows. First, for each leaf of the CP-tree, generate all maximal frequent sets associated with it, as outlined above, using the FP-tree. A technicality here is that the FP-tree at a circumstance leaf may well contain items that are *not* necessarily frequent in that circumstance, although they are frequent (under "tree"). So, an extra detail in generating maximal frequent itemsets at a given circumstance is first checking if the support count for the itemset, as it is being considered, adds up to or above the support threshold. Notice that any maximal frequent itemset associated with a leaf circumstance in the initial CP-tree, is by definition a pair in the Armstrong Basis, since the leaves represent circumstances for which the only stronger circumstance in the lattice is "false".

Next, having processed all leaves of the CP-tree, delete its leaves, and move up the FP-trees pointed by them to their parent. Two or more FP-trees might get merged in so that their roots may get identified in this process. Repeat the process for the leaves of the current FP-tree. Let α be a leaf of the current CP-tree and S be any maximal frequent itemset generated for it. Then (α, S) should be an element of the Armstrong Basis only if the set S was not generated as a maximal frequent set for any stronger circumstance. This check is needed. Repeat these steps iteratively, bottom-up until

[2] An item i in the FP-tree associated with a circumstance α is frequent but need *not* be frequent in α itself.

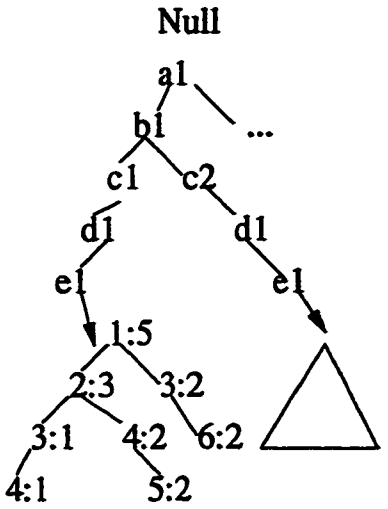

Figure 37: An Example Circumstance Prefix Tree with Leaves Pointing to Itemset FP-trees

the CP-tree becomes empty (in which case, its null root will point to one FP-tree). The algorithm then terminates. The following is drawn:

Proposition 4 *The algorithm for Armstrong Basis outlined above correctly computes precisely those pairs of circumstances and itemsets that belong to the Armstrong Basis.*

Proof Sketch: The Proposition follows from the following observations. For each leaf of the original CP-tree, every pair computed is indeed a pair of the Armstrong Basis, and every pair involving a circumstance involving all circumstance attributes is computed this way. For the remaining circumstances, a simple induction can show that all and exactly those pairs that involve them will be computed by the iterative procedure above.

5.4 Summary and related Work

This chapter has defined Armstrong Basis – a new concept of data mining on bilattices. A couple of algorithms are provided for Armstrong Basis computation. The importance of computing Armstrong Basis is similar to CUBE operation. While the computation is huge, it can provide effective support for queries involving migration between itemsets and circumstances.

Recently, there are interests in mining long patterns. Algorithms for mining maximal frequent sets have been proposed in this context [Bay98, AAP00]. This is relevant

because of the interest in minimal circumstances. However, these algorithms operate in a different lattice and the mechanisms of the algorithms appear different. Given the interest in the Armstrong Basis, these algorithms might offer yet another way to efficiently find maximal frequent itemsets associated with a circumstance. As an example, the FP-tree method [HPY00, PHM00] have already been discussed.

Chapter 6

Conclusions and Future Work

The contributions of this thesis are summarized in this chapter. Then the future research opening related to this topic is discussed.

6.1 Conclusions

With formalizing the notions of item, transaction, and circumstance, the traditional data mining can have a even more wide variety of applications. The setup of itemset lattice and circumstance lattice helps us to distinguish the difference and yet realizes the close relations with itemsets and circumstances.

The introducing of constraints in corrrelations can help users to focus and control the mining task undertaken by the system. Brin *et al.*'s [BMS97] principle of finding minimal sets is extended to two useful semantics of answering constrained correlation queries – (i) all minimal answers that are valid, and (ii) all minimal valid answers. The former are in general a proper subset of the latter, but for one of the important class where all constraints are anti-monotone, the two coincide. A basic algorithm as well as an efficient algorithm are proposed for computing either answer set—Algorithms BMS+ and BMS++ for answer set (i) and BMS* and BMS** for answer set (ii). Analytical articulation is given for the reason why Algorithm BMS++ (resp., BMS**) is more efficient than Algorithm BMS+ (resp., BMS*). Besides, for the case where all constraints in the user query are anti-monotone, Algorithm BMS++ is the most efficient among the four.

A series of experiments is conducted to validate the analysis, using synthetic data

generated by two different methods. These experiments bring out their relative performance and the tradeoffs. From all the experiments, it can be concluded that for BMS+, the constraint type does not influence the overall performance. Algorithm BMS* is slightly affected by the selectivity. On the contrary, the performance of algorithm BMS** is heavily dependent on the constraint for pruning. Generally speaking, BMS++ shows the best performance under all circumstances. When constraint selectivity is low, BMS++ relies on the constraint to prune the candidate itemsets, and when the constraint selectivity is high, it relies on the upward closed property of being correlated to do the pruning.

The new concept of circumstance lattice shows a clear separation between item attributes and circumstance attributes and pays its dividends: the close ties with the cube computation paradigm, and between cube computation and mining of i-anti-monotone patterns opens up new and promising directions for future work. As motivated earlier, finding minimal (for c-monotone patterns) circumstances under which a pattern holds for a specific itemset is useful in its own right. Two existing cube algorithms, BUC and Chunked Array, are modified and a new algorithm, called *minCirc*, is given for determining the strongest (resp., the weakest) circumstances under which a pattern (a frequent itemset) hold. The summary of the decision tree shows that *minCirc* is competitive in that there are conditions under which each algorithm is the algorithm of choice (Section 4.4).

The framework developed in this thesis goes far beyond just support-based mining of frequent itemsets. This is achieved using properties of patterns on the bilattice of circumstances and itemsets. Finally, a useful notion of *Armstrong Basis* is proposed with which people can find (say, minimal) circumstances where a pattern holds for an itemset, and then find which other itemsets satisfy the pattern under these circumstances. This is in the spirit of mining the result of mining, one of the important problems identified by Imielinski and Mannila in their seminal paper [ImMa96]. Just as data cube is an important operator for supporting queries involving multiple group-bys, for queries involving migration between itemsets and circumstances, Armstrong Basis can provide effective support. A couple of algorithms are provided for computing the Armstrong basis.

6.2 Future Work

Several questions remain open. Firstly, it is not clear how some of the constraints such as $avg(S.\texttt{price})\ \theta\ c$, which are neither monotone nor anti-monotone, can be handled for the contrained correlations. The solution space may not be a single region and instead may have holes in it. Under these conditions, blindly returning only minimal valid answers does not make sense. Defining meaningful answer sets and computing them efficiently for such constraints is an interesting issue. Another question is how constraints can help in mining patterns other than associations and correlations?

With the introducing of circumstances and establishment of bilattices, more work needs to be done to search efficient algorithms for computing circumstance lattice, especially for Armstrong Basis. Experiments are needed to evaluate the performance of the algorithms since the efficient computation will be very useful in answering a wide range of mining queries.

Bibliography

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *1993 ACM-SIGMOD Proc. Special Interest Group on Management of Data (ACM-SIGMOD'93)*, pages 207–216.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, Sept. 1994.

[AS95] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, Mar. 1995.

[AAP00] R. Agrawal, C.C. Aggarwal, and V. Prasad. Depth first generation of long patterns. In *Knowledge Discovery and Data Mining 2000 (KDD 2000)*, to appear.

[BMS97] S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. In *1997 ACM-SIGMOD Proc. Special Interest Group on Management of Data (ACM-SIGMOD'97)*, pages 265–276, Tucson, Arizona, May 1997.

[Bay98] R. J. Bayardo. Efficiently mining long patterns from databases. In *1998 ACM-SIGMOD Proc. Special Interest Group on Management of Data (ACM-SIGMOD'98)*, pages 85–93, Seattle, WA, June 1998.

[BR99] K. S. Beyer and R. Ramakrishnan. Bottom-Up Computation of Sparse and Iceberg CUBEs. In *1999 ACM-SIGMOD Proc. Special Interest Group on*

Management of Data (ACM-SIGMOD'99), pages 359–370, Philadelphia, PA, June 1999.

[Chau98] S. Chaudhuri. Data mining and database systems: Where is the intersection? *Data Engineering Bulletin*, 21:4–8, March 1998.

[CF88] J. Cheng, U. M. Fayad, K. B. Irani, and Z. Quian. 1988. Improved Decision Trees: A Generalized Version of ID3. In *Proc. of the Fifth International Conference on Machine Learning.* 1988. Morgan Kaufmann. San Mateo, California. pages 100–107, 1988

[CN89] P. Clark, and T. Niblett. The CN2 Induction Algorithm. *Machine Learning* 3: 261–283, 1989.

[DL99] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)*, pages 43–52, San Diego, CA, Aug. 1999.

[Fit91] M. Fitting Bilattices and the semantics of logic programming. *J. Logic Programming* **11**, 91–116.

[GLW99] G. Grahne, L. V. S. Lakshmanan, Xiaohong Wang. Interactive Mining of Correlations - A Constraints Perspective. ACM SIGMOD workshop on research issues in data mining and knowledge discovery, pages 7-1, Philadelphia, 1999.

[GLW00] G. Grahne, L. V. S. Lakshmanan, Xiaohong Wang. Efficient Mining of Constrained Correlated Sets. In *International Conference on Data Engineering 2000 (ICDE 2000)*, pages 512-521 San Diego, CA, 2000.

[GLW01] G. Grahne, L. V. S. Lakshmanan, Xiaohong Wang, Minghao Xie. On Dual Mining: From Patterns to Circumstances, and Back. Accepted by *International Conference on Data Engineering 2001 (ICDE 2001).*

[G*96] Jim Gray, Adam Bosworth, Andrew Layman, Hamid Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. In *International Conference on Data Engineering 1996 (ICDE'96)*, pages 152-159.

[HPY00] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *2000 ACM-SIGMOD Proc. Special Interest Group on Management of Data (ACM-SIGMOD 2000)*, pages 1–12, Dallas, TX, May 2000.

[HKK97] E.-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In *1997 ACM-SIGMOD Proc. Special Interest Group on Management of Data (ACM-SIGMOD'97)*, pages 277–288.

[HaFu95] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95)*, pages 420–431.

[ImMa96] Tomasz Imielinski, Heikki Mannila A Database Perspective on Knowledge Discovery. In *Communications of the ACM* 39(11): 58-64 (1996).

[Klem+94] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 3rd Int. Conf. Information and Knowledge Management*, pages 401–408, 1994.

[KLKF98] F. Korn, A. Labrinidis, Y. Kotidis, and C. Faloutsos. Ratio rules: A new paradigm for fast, quantifiable data mining. VLDB 1998. In *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*, pages 582–593.

[LNHP99] L. V. S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *1999 ACM-SIGMOD Proc. Special Interest Group on Management of Data (ACM-SIGMOD'99)*, pages 157–168, Philadelphia, PA, June 1999.

[LSW97] B. Lent, A. Swami, and J. Widom. Clustering association rules. In *Proc. 1997 Int. Conf. Data Engineering (ICDE'97)*, pages 220–231, Birmingham, England, Apr. 1997.

[MTV94] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*, pages 181–192, Seattle, WA, July 1994.

[MTV97] H. Mannila, H Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.

[NLHP98] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *1998 ACM-SIGMOD Proc. Special Interest Group on Management of Data (ACM-SIGMOD'98)*, pages 13–24, Seattle, WA, June 1998.

[PCY95] J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. In *1995 ACM-SIGMOD Proc. Special Interest Group on Management of Data (ACM-SIGMOD'95)*, pages 175–186. SIGMOD 1995.

[PCY95b] J.S. Park et al. Efficient parallel mining for association rules. In *Proc. 4th Int. Conf. Information and Knowledge Management*, pages 31–36, 1995.

[PHM00] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD'00)*, pages 11–20, Dallas, TX, May 2000.

[P00] Jian Pei. Personal Communication, August 2000.

[STA98] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. SIGMOD 1998. In *1999 ACM-SIGMOD Proc. Special Interest Group on Management of Data (ACM-SIGMOD'99)*, pages 343–354.

[SON95] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95)*, pages 432–443.

[SiZd97] A. Silberschatz and S. Zdonik. Database systems – breaking out of the box. SIGMOD Record, 26, pages 36–50, 1997.

[SBMU98] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. In *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*, pages 594–605.

[SVA97] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 67–73, Newport Beach, CA, Aug. 1997.

[Toiv96] H. Toivonen. Sampling large databases for association rules. In *Proc. 1996 Int. Conf. Very Large Data Bases (VLDB'96)*, pages 134–145.

[Tsur+98] D. Tsur, J. D. Ullman, S. Abitboul, C. Clifton, R. Motwani, and S. Nestorov. Query flocks: A generalization of association-rule mining. In *1998 ACM-SIGMOD Proc. Special Interest Group on Management of Data (ACM-SIGMOD'98)*, pages 1–12. SIGMOD 1998.

[W74] William Ward Armstrong: Dependency Structures of Data Base Relationships. IFIP Congress 1974: 580-583.

[ZaHs99] M.J. Zaki and C. Hsiao. Charm: an efficient algorithm for closed association rule mining. Tech. Report., RPI, 1999.

[ZDN97] Yihong Zhao, Prasad Deshpande, Jeffrey F. Naughton. An Array-Based Algorithm for Simultaneous Multidimensional Aggregates. In *1997 ACM-SIGMOD Proc. Special Interest Group on Management of Data (ACM-SIGMOD'97)*, pages 159-170.