Investigation of Discovering Rules from Data

By

Ng, King Kwok

Supervised By :

Prof. LAM, Wai

Submitted to the Division of Department of Systems Engineering

& Engineering Management

in partial fulfillment of the requirements for the

degree of Master of Philosophy

at the

Chinese University of Hong Kong

DECEMBER, 1999

The Chinese University of Hong Kong holds the copyright of this thesis. Any person(s) intending to use a part or whole of the material in the thesis in a proposed publication must seek copyright release from the Dean of the Graduate School.

Investigation of Discovering Rules from Data

By

Ng, King Kwok

Supervised By :

Prof. LAM, Wai

Submitted to the Division of Department of Systems Engineering & Engineering Management

in partial fulfillment of the requirements for the degree of Master of Philosophy

at the

Chinese University of Hong Kong

DECEMBER, 1999

香港中文大學
圖書館系統
28 JUN 2001
UNIVERSITY
LIBRARY SYSTEM
THE CHINESE UNIVERSITY OF HONG KONG

Acknowledgments

I would like to thank my supervisor, Dr. Wai Lam, for inspiring me the field of data mining. My gratitude goes to him for his guidance and support throughout my research. Without his help, I could not have my thesis written.

I also thank my examiners, Dr. Kai Pui Lam and Dr. Boon Toh Low, and external examiner Dr. Takao Miura for their suggestions and reading my thesis. My gratitude goes my classmates and colleagues in my department throughout my study in the university.

Investigation of Discovering Rules from Data

summitted by

Ng, King Kwok

for the degree of Master of Philosophy
at the Chinese University of Hong Kong

Abstract

從數據中採集規則之研究

作者:吳敬國

撮要

我們提出一個新的方法, 採用動態計算來處理關聯規則之更新。現有的一種名爲動態項集計算(DIC)的動態計算技術能處理單一數據庫中頻繁出現項集的計算。但是, 直接採用動態項集計算(DIC)於頻繁出現項集的更新, 會浪費了原本儲存的資訊, 因此效率不高。因爲我們仍然需要從新計算舊有的數據庫, 需要的運算時間很長。我們提出的新方法, 名爲IDIC_M, 及其變體IDIC_S, 能有效地使用動態計算技術來處理關聯規則之更新。我們進行了廣泛的實驗去比較我們的新方法和現有的FUP方法。實驗結果顯示我們的新方法比FUP方法有更好的表現。

雖然在多層關聯規則的採集及利用使用者之信念於規則採集方面已有很多的研究, 但是並沒有研究同時處理這兩個問題。在這論文當中, 我們提出一個新的方法來利用使用者信念, 從數據庫中採集多層關聯規則。我們提出一套用名爲確切信念的有興趣規則特性, 使用於用者信念之下的多層關聯規則。我們又提出了一個支援確切信念, 名爲MARUB_E的新方法, 來處理多層關聯規則的採集。同時, 我們又提出了模糊信念, 以及一個支援模糊信念, 名爲MARUB_V的新方法, 來處理多層關聯規則的採集。我們並對新提出的方法作出了評定和分析。

Investigation of Discovering Rules from Data

submitted by

Ng, King Kwok

for the degree of Master of Philosophy
at the Chinese University of Hong Kong

Abstract

We have proposed a new algorithm for incremental mining of association rules based on dynamic counting. A dynamic counting technique known as the Dynamic Itemset Counting (DIC) for large itemset counting in a *single* database was developed previously. Direct application of DIC in incremental updating is not efficient since available information stored in the original database is not useful. We still need to count the original database again and result in long computational time. Our new algorithm, called the IDIC_M algorithm and its variant, the IDIC_S algorithm, can efficiently conduct incremental mining of association rules based on dynamic counting. We have conducted extensive experiments to compare our algorithms with the FUP algorithm. The results illustrate that our two algorithms have better performance over the FUP algorithm.

While much work has been done on multiple-level association rule mining and rule discovery in the presence of user belief, very few research has attempted to tackle both problems at the same time. In this thesis, we have proposed a new approach for mining multiple-level association rules from databases in the presence of user belief. We have introduced a set of properties for rule interestingness on multiple-level association rules under user belief, namely the exact

belief. To tackle this problem, we have developed a multiple-level association rules mining algorithm, called MARUB_E which support the specification of the exact beliefs. In addition, we have also introduced the vague belief and have developed a multiple-level association rules mining algorithm, called MARUB_V to handle the vague beliefs. We have shown the evaluation and analysis done on our new approach.

Contents

Acknowledgments ii

Abstract iii

1 Introduction 1

1.1 Data Mining and Rule Discovery 1

1.1.1 Association Rule 3

1.1.2 Sequential Pattern 4

1.1.3 Dependence Rule 6

1.2 Association Rule Mining 8

1.3 Contributions 9

1.4 Outline of the Thesis 10

2 Related Work on Association Rule Mining 11

2.1 Batch Algorithms 11

2.1.1 The Apriori Algorithm 11

2.1.2 The DIC Algorithm 13

2.1.3 The Partition Algorithm 15

2.1.4 The Sampling Algorithm 15

2.2 Incremental Association Rule Mining 16

2.2.1 The FUP Algorithm 17

2.2.2 The FUP2 Algorithm 18

2.2.3 The FUP* Algorithm 19

2.2.4 The Negative Border Method 20

2.2.5 Limitations of Existing Incremental Association Rule Mining Algorithms 21

3 A New Incremental Association Rule Mining Approach 23

3.1 Outline for the Proposed Approach 23

3.2 Our New Approach 26

3.2.1 The IDIC_M Algorithm 26

3.2.2 A Variant Algorithm: The IDIC_S Algorithm 29

3.3 Performance Evaluation of Our Approach 30

3.3.1 Experimental Results for Algorithm IDIC_M 30

3.3.2 Experimental Results for Algorithm IDIC_S 35

3.4 Discussion 39

4 Related Work on Multiple-Level AR and Belief-Driven Mining 41

4.1 Background on Multiple-Level Association Rules 41

4.2 Related Work on Multiple-Level Association Rules 42

4.2.1 The Basic Algorithm 42

4.2.2 The Cumulate Algorithm 44

4.2.3 The EstMerge Algorithm 44

4.2.4 Using Hierarchy-Information Encoded Transaction Table . 45

4.3 Background on Rule Mining in the Presence of User Belief 46

4.4 Related Work on Rule Mining in the Presence of User Belief . . . 47

4.4.1 Post-Analysis of Learned Rules 47

4.4.2 Using General Impressions to Analyze Discovered Classification Rules 49

4.4.3 A Belief-Driven Method for Discovering Unexpected Patterns 50

4.4.4 Constraint-Based Rule Mining 51

4.5 Limitations of Existing Approaches 52

5 Multiple-Level Association Rules Mining in the Presence of User Belief **54**

5.1 User Belief Under Taxonomy 55

5.2 Formal Definition of Rule Interestingness 57

5.3 The MARUB_E Mining Algorithm 61

6 Experiments on MARUB_E **64**

6.1 Preliminary Experiments 64

6.2 Experiments on Synthetic Data 68

6.3 Experiments on Real Data 71

7 Dealing with Vague Belief of User **76**

7.1 User Belief Under Taxonomy 76

7.2 Relationship with Constraint-Based Rule Mining 79

7.3 Formal Definition of Rule Interestingness 79

7.4 The MARUB_V Mining Algorithm 81

8 Experiments on MARUB_V **84**

8.1 Preliminary Experiments 84

8.1.1 Experiments on Synthetic Data 87

8.1.2 Experiments on Real Data 93

9 Conclusions and Future Work **96**

9.1 Conclusions 96

9.2 Future Work 97

List of Tables

1.1 Original Transaction Database 4

1.2 Database Sorted by Customer Id and Transaction Date 5

1.3 Customer-Sequence Version of the Database 5

1.4 Contingency Table. 6

1.5 A Transaction Database. 8

2.1 Candidate Itemsets and Large Itemsets Found by the Apriori Algorithm. 12

3.1 Parameter Table. 30

6.1 Output Rules for Exact Belief (Set 1). 67

6.2 Output Rules for Exact Belief (Set 1). 70

6.3 Output Rules for Exact Belief (Set 2). 71

6.4 Output Rules for Exact Belief (Set 1). 73

6.5 Output Rules for Exact Belief (Set 2). 75

8.1 Output Rules for Vague Belief (Set 1). 86

8.2 Output Rules for Vague Belief (Set 1). 90

8.3 Output Rules for Vague Belief (Set 2). 92

8.4 Output Rules for Vague Belief (Set 1). 94

8.5 Output Rules for Vague Belief (Set 2). 95

List of Figures

3.1 IDIC_M and FUP for Different Supports 32

3.2 IDIC_M and FUP for Different Incremental Sizes (with the same random seed) 33

3.3 IDIC_M and FUP for Different Incremental Sizes (with different random seeds) 34

3.4 IDIC_M and DIC for Different Incremental Sizes 35

3.5 IDIC_S and FUP for Different Supports 36

3.6 IDIC_S and FUP for Different Incremental Sizes (with the same random seed) 37

3.7 IDIC_S and FUP for Different Incremental Sizes (with different random seeds) 38

3.8 IDIC_S and DIC for Different Incremental Sizes 39

4.1 A Hierarchy of Food Items 41

4.2 Another Hierarchy of Food Items 45

5.1 Another Hierarchy of Food Items 56

5.2 The General-Specific Relationship Among the Rules 57

5.3 The Framework of the MARUB_E Algorithm 62

5.4 Function *expandDatabase*1() 63

5.5 Function $findIntAnc(X)$ 63

6.1 A Hierarchy of Items in the Database 64

6.2 Distribution of Items in the Database 65

6.3 The Ancestor/Descendant Relationships among Discovered Rules (Set 1) 66

6.4 The Ancestor/Descendant Relationships among Discovered Rules (Set 1) 69

6.5 The Ancestor/Descendant Relationships among Discovered Rules (Set 2) 70

6.6 A Hierarchy of Items in the Real Data 71

6.7 The Ancestor/Descendant Relationships among Discovered Rules (Set 1) 73

6.8 The Ancestor/Descendant Relationships among Discovered Rules (Set 2) 74

7.1 The General-Specific Relationship Among the Rules 78

7.2 The Framework of the MARUB_V Algorithm 82

7.3 Function *expandDatabase*2() 83

8.1 The Ancestor/Descendant Relationships among Discovered Rules (Set 1) 85

8.2 The Ancestor/Descendant Relationships among Discovered Rules (Set 1) 93

8.3 The Ancestor/Descendant Relationships among Discovered Rules (Set 2) 95

Chapter 1

Introduction

1.1 Data Mining and Rule Discovery

Knowledge discovery in databases (KDD), or data mining is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data [14]. As information technology is widely used in many different kinds of applications nowaday, the amounts of data generated by such information systems can be very huge in size. If proper techniques are applied to these very large databases, many invaluable and useful information or knowledge can be extracted. The extracted knowledge are used in different aspects such as decision support, prediction and marketing.

Some possible kinds of knowledge that can be mined from data are as follows.

- `Association Rule`, in the form of an implication rule that represents the association among a set of objects, or items, in a database. An association rule is in the form of $X \Rightarrow Y$ where X and Y are non-overlapping sets of items. A rule $X \Rightarrow Y$ means that when the set X of items appears in a record of the database, the set Y of items usually also appears. An association rule usually carries a confidence value and a support value.

An example of an association rule is when people buy bread from the supermarket, they usually also buy milk.

- `Classification Rule`, also in the form of an implication rule that shows the relationship between the values of a set of features with a particular class. It works in the way that if a set of features have some particular values, then we can tell that the tuple belongs to a specific class. For example, if a patient have some particular results in some body tests, then we can classify him/her as suffering from a particular disease.

- `Clusters`, formed by grouping objects into different sets based on their attributes. For example, students can be divided into different groups according to their interests and performance in different subjects.

- `Sequential Pattern`, in the form of an ordered set of objects, or items, that are expected to appear over temporal records in the particular order. For example, customers are expected to buy TV-sets before buying video recorders.

- `Dependence Rule`, which is a set of items, that represents the relationship among items' presence or absence in the records of a database. For example, in the census data, the presence of item *aged under 18* and the absence of item *married* usually appear together.

These are just a few examples of knowledge that can be mined from data. Many other kinds of knowledge can be mined using their respective mining methods. We will look into details three kinds of knowledge, the association rule, the dependence rule and the sequential pattern, which are usually discovered from the transaction databases,

1.1.1 Association Rule

Data mining has recently attracted a large amount of attention due to its wide application. In particular, association rule is an important type of knowledge which can be discovered from large databases. Association rules have been widely used in areas like marketing, decision support and store layout design [7]. An example of association rule is $BREAD \Rightarrow MILK$. If it is discovered from a supermarket transaction database, it may mean that "when people buy bread, they usually also buy milk". It may be a valuable rule for the supermarket manager that he/she may decide to display the bread products and the milk products in neighbouring areas in the supermarket. Another example is that if a company discovers the rules about the associations of some group of customers with some specific products of the company, then marketing campaigns on different products can target at the appropriate customers without wasting resource. For example, if a rule "$YOUNG_CUSTOMER \Rightarrow POP_MUSIC$" is discovered, the company can send mails about new pop concerts to only the young customers instead of all the customers.

Besides binary and categorical data, some databases may also contain quantitative, or numerical data. Much work has been done for discovering association rules from such quantitative databases [33, 37, 18, 26, 6, 22]. In some application domains, a conceptual hierarchy or taxonomy is available. With such additional knowledge, generalized or multiple-level association rules can be discovered [32, 15, 29]. A generalized or multiple-level association rule may contain items from different levels from the conceptual hierarchy. Much work has been done on varies aspects of the association rules [25, 38, 19, 12, 17, 3, 39, 13, 16].

1.1.2 Sequential Pattern

The problem of mining sequential patterns was introduced by Agrawal et al. [2] and some researches have been done on the sequential patterns [34, 28]. An example of sequential pattern is that customers typically rent 'Star Wars', then 'Empire Strikes Back', and the 'Return of the Jedi'. The database considered contains customer-id, transaction time, and the items brought in each transaction. And example of such database is shown as follows. Table 1.1 is the original database. The database sorted by customer-id and transaction date is shown in Table 1.2.

Transaction Date	Customer Id	Items Bought
Jan 10 '99	1	1,2
Jan 12 '99	5	9
Jan 15 '99	1	3
Jan 20 '99	1	4,6,7
Jan 25 '99	4	3
Jan 25 '99	3	3,5,7
Jan 25 '99	2	3
Jan 30 '99	2	9
Jan 30 '99	4	4,7
Feb 25 '99	4	9

Table 1.1: Original Transaction Database

From the sorted database, a customer-sequence version of the database can be constructed from which the sequential patterns are to be mined.

With a minimum support set to 25%, i. e., a minimum support of two customers, the sequences: $< (30)(90) >$ and $< (30)(40, 70) >$ are the desired sequential patterns, where $< (3)(9) >$ is supported by customers 2 and 4 and $< (3)(4, 7) >$ supported by customers 1 and 4.

Customer Id	Transaction Date	Items Bought
1	Jan 10 '99	1,2
1	Jan 15 '99	3
1	Jan 20 '99	4,6,7
2	Jan 25 '99	3
2	Jan 30 '99	9
3	Jan 25 '99	3,5,7
4	Jan 25 '99	3
4	Jan 30 '99	4,7
4	Feb 25 '99	9
5	Jan 12 '99	9

Table 1.2: Database Sorted by Customer Id and Transaction Date

Customer Id	Customer Sequence
1	$< (10, 20)(30)(40, 60, 70) >$
2	$< (30)(90) >$
3	$< (30, 50, 70 >$
4	$< (30)(40, 70)(90) >$
5	$< (90) >$

Table 1.3: Customer-Sequence Version of the Database

1.1.3 Dependence Rule

Traditional association rules cannot capture negative implications in the data. For example, we can have the association rule $BUTTER \Rightarrow CHEESE$, but a fact like, 'when people buy butter, they do not usually buy cheese" cannot be captured directly by an association rule. Association rules have another serious problem as illustrated in the following example. Suppose we have supermarket basket data consisting of 100 transactions, where the relationship between butter and cheese is shown in the contingency table in Table 1.4.

	$BUTTER$	$\overline{BUTTER}$	row-sum
$CHEESE$	30	60	90
$\overline{CHEESE}$	5	5	10
col-sum	35	65	100

Table 1.4: Contingency Table.

From the contingency table, we can discover the association rule $BUTTER \Rightarrow CHEESE$, with a high confidence of $30/35 = 86\%$ and a high support of 30%. However,

$$p(BUTTER \wedge CHEESE)/(p(BUTTER) \times p(CHEESE)) = 0.3/(0.35 \times 0.9) = 0.95 < 1$$

which indicates that butter and cheese are actually negatively dependent. Also,

$$p(BUTTER \wedge \overline{CHEESE})/(p(BUTTER) \times p(\overline{CHEESE})) = 0.05/(0.35 \times 0.1) = 1.43 > 1$$

which indicates a high dependence between the presence of butter and the absence of cheese. The above association rule is therefore misleading if we just look at the association rule itself.

In order to prevent the above problem, Silverstein et al. [31] introduced the dependence rules. In dependence rules mining problem, the input data is similar to the basket data in association rule mining problem. A dependence rule is

defined as a set of dependent attributes (or items). Assume we are given the contingency table of an attribute set, together with a threshold, θ_s. Then, the support, s_{pct}, of the attribute set is defined as the percentage of cells in the contingency table having the counts larger than θ_s. For example, in Table 1.4, if θ_s is set as 25, then two cells in the contingency table of the attribute set $\{BUTTER, CHEESE\}$ have counts larger than θ_s. Assume we are also given a threshold θ_p, then we say an attribute set has the CT-support (i.e., contingency table support) if its support s_{pct} is larger than θ_p. For example, consider Table 1.4 again, and let θ_s and θ_p to be 25 and 25%, respectively. We say the attribute set $\{BUTTER, CHEESE\}$ have the CT-support since it has 50% of the cells having counts larger than 25. If an attribute set has the χ^2 value larger than a predefined cut off value, χ_α, then we say that this attribute set is dependent. Note that the α value here is the chi-squared significance level. For example, if the significance level α is set set to 95%, then we have an associated cutoff value χ_α of 3.84. Now, if the calculated χ^2 value of a set of items is 3.84 or more, then we say the set of items are dependent with a 95% confidence level.

We can easily prove that if a set of attributes are dependent, then all its supersets will also be dependent. Hence, we are only interested in the minimally dependent attribute sets. A minimally dependent attribute set M is defined as a set of dependent attributes where none of its subsets S is dependent, i.e., M is dependent $\wedge\ \forall S \subset M$, S is not dependent. We say an itemset is significant if it is minimally dependent and also has the CT-support. For the problem of mining dependence rules from a set of basket data, we are given the thresholds, θ_s, θ_p and α. The problem of mining dependence rules is to find out (as many as possible) the significant itemsets from the data.

1.2 Association Rule Mining

Assume we have a set of literals called items. Let DB be a database of transactions, where each transaction contains a set of different number of items. An association rule is an implication of the form $X \Rightarrow Y$, where X and Y are itemsets and $X \cap Y = \emptyset$. We call an itemset containing k items a k-itemset. The association rule $X \Rightarrow Y$ holds in DB with *confidence* c if $c\%$ of the transactions in DB that contain X also contain Y. The association rule $X \Rightarrow Y$ has *support* s in DB if $s\%$ of the transactions in DB contain $X \cup Y$. Given a minimum confidence threshold θ_f and a minimum support threshold θ_s, the problem of mining association rules is to find out all the association rules whose confidence and support are larger than the respective thresholds. For an itemset X, its support is defined similarly as the percentage of transactions in DB which contain X. Given a minimum support threshold θ_s, an itemset X is large, or frequent, if its support is no less than θ_s.

For example, if we have a database consisted of 4 transactions as shown in Table 1.5, then the itemset A, B, C is frequent given $\theta_s = 30\%$.

transaction	items contained
1	A, B, C, D
2	A, B, D
3	A, C, E
4	A, B, C, F

Table 1.5: A Transaction Database.

The problem of mining association rules is reduced to the problem of finding all large itemsets for a pre-determined minimum support [7]. This is due to the fact that discovering large itemsets in a database requires most of the computational time used in mining association rules. Association rules can be easily found once a set of large itemsets are available.

1.3 Contributions

We propose a new algorithm to handle the problem of incremental mining of association rules. Recent approaches on this problem usually employ the Apriori algorithm. Some of them are not efficient enough and the others require the storage of extra information. We develop a new algorithm, called the Incremental Dynamic Itemset Counting algorithm. It makes use of the dynamic counting technique to deal with this problem in a more efficient way. We have conducted extensive experiments to compare our new algorithm with a recent incremental mining algorithm, the FUP algorithm. The results illustrate that our algorithm have better performance over the FUP algorithm in terms of the computational time. We also investigate a variant of our algorithm and demonstrate its effectiveness. Note that our algorithms can be easily extended to handle the deletion of transactions from the database, in addition to the insertion of transactions.

In the rule mining process, except the basic constraints we usually used, users may also want to specify some prior beliefs or interests so that these user specified additional information can increase either the speed of mining or the quality of the resulting rules. While much work has been done on multiple-level association rule mining and rule discovery in the presence of user belief, very few research has attempted to tackle both problems at the same time. In this thesis, we propose a new approach for mining multiple-level association rules from databases in the presence of user belief. There are two main aims of incorporating user beliefs in association rule mining. The first one is to reduce the amount of rules mined. Secondly, we can concentrate on the rules that are desired by the users and ignore the rules uninteresting to the users. We introduce a set of properties for rule interestingness on multiple-level association rules under two kinds of user beliefs, namely, the exact belief and the vague belief. In addition, we develop a multiple-level association rules mining algorithm, called MARUB which support

the specification of the two kinds of user beliefs. We also show the evaluation and analysis done on our new approach.

1.4 Outline of the Thesis

The thesis is organized as follows:

In Chapter 2, we will introduce some related previous work in association rules, where emphasis will be placed on the mining and incremental mining of association rules.

In Chapter 3, we will describe our proposed incremental association rule mining algorithm, and a variant algorithm, together with some performance evaluation on the algorithms.

In Chapter 4, we will look at the background and some related work on the mining of multiple-level association rules and the rule mining in the presence of user belief.

We will present our new approach for mining multiple-level association rules in the presence of user belief in Chapter 5 and 7. We will introduce a new definition for rule interestingness on multiple-level association rules under two kinds of user beliefs, namely, the exact belief and the vague belief. Rule mining under exact belief and vague will be described in Chapter 5 and Chapter 7, respectively. Evaluation will be done on our new approach presented for exact belief and vague belief in Chapter 6 and Chapter 8, respectively.

We will have the conclusions and some possible directions of future work in the Chapter 9.

Chapter 2

Related Work on Association Rule Mining

2.1 Batch Algorithms

2.1.1 The Apriori Algorithm

The Apriori algorithm developed by Agrawal et al. [1] uses the idea that 'If an itemset is frequent, so are its subsets" to improve the efficiency of frequent itemset discovery. In this algorithm, after the $k-th$ pass over the database, the set of frequent size k itemsets are found. Then the set of size k frequent itemsets are used to generate the set of size $k+1$ potential frequent itemset candidates by a method called apriori-gen. The apriori-gen generates a superset of the large $(k+1)$-itemsets from the large k-itemsets. These size $k+1$ candidates will then be counted in the next pass over the database.

We give an example to show how the Apriori algorithm works. Let us consider again Table 1.5. Again we assume $\theta_s = 30\%$. Now we have the set of candidate size one itemset $C[1]$ containing itemsets $\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}$. Then

we count the support of itemsets in $C[1]$ to find that only itemsets $\{A\}, \{B\}, \{C\}, \{D\}$ have the enough supports to become frequent itemsets in the database. Now we have the set $L[1]$ of frequent size one itemsets containing $\{A\}, \{B\}, \{C\}, \{D\}$. The next step is to construct the set of size two candidate itemsets $C[2]$ which contains itemsets {AB}, {AC}, {AD}, {BC}, {BD}, {CD}. The candidates in $C[2]$ are formed by combining the itemsets in $L[1]$. For example, candidate itemset $\{AB\}$ is formed by combining itemsets $\{A\}$ and $\{B\}$ in $L[1]$. After counting the candidate itemsets in $C[2]$, we can find the set $L[2]$ of frequent itemsets which contains $\{AB\}, \{AC\}, \{AD\}, \{BC\}, \{BD\}$. Similarly, we can find both $C[3]$ and $L[3]$ to contain itemsets $\{ABC\}$ and $\{ABD\}$. We cannot construct new size four candidate itemset and the algorithm terminates.

C[1]	{A}, {B}, {C}, {D}, {E}, {F}
L[1]	{A}, {B}, {C}, {D}
C[2]	{AB}, {AC}, {AD}, {BC}, {BD}, {CD}
L[2]	{AB}, {AC}, {AD}, {BC}, {BD}
C[3]	{ABC}, {ABD}
L[3]	{ABC}, {ABD}

Table 2.1: Candidate Itemsets and Large Itemsets Found by the Apriori Algorithm.

In the work by Agrawal et al. [1] the method for rule generation from frequent itemsets is also introduced. It uses the idea similar to that used in frequent itemset discovery, namely 'If a rule $(f - C) \Rightarrow C$ has the minimum confidence, then all rules in the form $(f - C) \Rightarrow C$ also have minimum confidence". Here f is a frequent itemset and C is a item such that $C \in f$. For example, we look at the rule $\{A, B\} \Rightarrow \{C, D\}$ which has a confidence $P(ABCD)/P(AB)$ and the rule $\{A, B, C\} \Rightarrow \{D\}$ with confidence $P(ABCD)/P(ABC)$, where $P(ABCD)$ is the percentage count for the itemset $\{ABCD\}$ in the database. Since the confidence of the former rule must be lower than the later rule, if the first rule has the minimum confidence, then the second rule must also have

the minimum confidence. By using this idea, rules can be generated in a way similar to the frequent itemset discovery starting with the generation of rules with single-item consequents. For example, with the frequent itemset $\{A, B, C\}$, 1-item consequent rules $B, C \Rightarrow \{A\}, \{A, C\} \Rightarrow B$ and $\{A, B\} \Rightarrow \{C\}$ can be formed and tested for minimum confidence. If all three rules have the minimum confidence we can then generate 2-item consequent rule candidates as $\{C\} \Rightarrow \{A, B\}, \{B\} \Rightarrow \{A, C\}$ and $\{A\} \Rightarrow \{B, C\}$.

2.1.2 The DIC Algorithm

Brin et al. [5] introduced the dynamic itemset counting algorithm [5] for discovering large itemsets from a database.

In the algorithm, the whole database is divided into intervals each containing M consecutive transactions. The candidate itemsets are counted against every M-transaction interval in the database. At the end of the counting of each M-transaction interval, the counts of the itemsets are checked. If any itemsets are found to have the minimum support count (no matter their counting are finished or not), they are treated as large itemsets and used to generate new candidate itemsets using the apriori-gen method.

Since our new approach will make use of the dynamic counting technique, we will first have a brief description of the DIC algorithm.

In the DIC algorithm each itemset is classified as one of the following categories:

- `confirmed-large` : an itemset we have finished counting that exceeds the support threshold.
- `confirmed-small` : an itemset we have finished counting that is below the

support threshold.

- `suspected-large` : an itemset we are still counting that exceeds the support threshold.
- `suspected-small` : an itemset we are still counting that is below the support threshold.

The DIC algorithm works as follows:

1. All the 1-itemsets are marked `suspected-small`. All other itemsets are unmarked.
2. Read M transactions (where M is a predetermined interval size). For each transaction, increment the respective counters for the itemsets marked `suspected-large` or `suspected-small`.
3. If a `suspected-small` itemset has a count that exceeds the support threshold, turn it into a `suspected-large`. If any immediate superset of it has all of its subsets as `confirmed-large` or `suspected-large`, add a new counter for it and mark it `suspected-small`.
4. If a `suspected-small` (or `suspected-large`) itemset has been counted through all the transactions, mark it `confirmed-small` (or `confirmed-large`) and stop counting it.
5. If we are at the end of the transaction file, rewind to the beginning.
6. If any `suspected-large` or `suspected-small` itemsets remain, go to Step 2.

What DIC does is just to create possible new candidate itemsets at every M-transaction interval. Note that for the Apriori algorithm, new candidate itemsets

can be created only at the end of a pass through the database. In this way, if all the subsets of an itemset are large, this itemset can start to be counted at the end of the interval, without delaying to the end of the pass over the database. Reduced number of passes over the database can hence be achieved. One thing to consider when using DIC is that while decreasing the interval size M may lead to fewer passes over the database and the overhead in checking the itemsets at the end of the intervals may also increase.

2.1.3 The Partition Algorithm

The Partition Algorithm introduced by Savasere et al. ensures two pass over the whole database for the discovery of all large itemsets. [27]. The idea is to divided the whole database into partitions small enough to be handled in the main memory. In the first pass, each partition is read into the memory where the *local* large itemsets in the partition are found. The union of all the local large itemsets is actually a superset of the global large itemsets, since a global large itemset must be globally large in at least one partition. Then in the second pass, the candidates formed by the union of all the local large itemsets are counted in the database to determine the global large itemsets.

2.1.4 The Sampling Algorithm

An algorithm based on sampling was introduced by Toivonen et al. [36]. The algorithm works by first reading in a small fraction of the database into the main memory. The large itemsets in the small fraction are found in the main memory with a support threshold lower than the intended threshold, and the negative border of the set of large itemsets is also computed. Note that the negative border of the set of large itemsets is actually all the candidates itemsets

generated in all levels by an level-wise algorithm (in this case, apriori-gen) which do not have minimum support. The large itemsets in the small fraction and the negative border then become the candidates to be counted in the rest of the database with the actual support threshold. It is expected that these candidates can cover nearly all (if not all) the itemsets that are actually large in the whole database. If some itemsets found to be large after the first pass actually comes from the negative border of the set of expected large itemsets, there may be the possibility of missing large itemsets in the candidates and a second pass over the database is required. This algorithm, therefore, requires at most two passes over the database. If the support threshold is set lower in the count over the small fraction, hopefully there should be fewer occasions of uncounted large itemsets in the first pass. However, more 0 candidates would be counted.

2.2 Incremental Association Rule Mining

Assume we have a database and some association rules discovered from this database. Now, if some new data comes to the database, we will need to update the set of the discovered association rules. One simple way is to run an association rule mining algorithm over the incremented database to get the new set of association rules. However, this approach is not efficient. To address this problem, we explore some incremental association rule mining techniques to avoid the redundant work of the whole mining process over the incremented database.

Let L_{DB} be the set of large itemsets in the database DB, s be the minimum support, and D be the number of transactions in DB. Assume that for each $X \in L_{DB}$, its support count which is the number of transactions in DB containing X, is available. Suppose we gather a set of new additional transactions db. The problem of incremental mining is to find all itemsets having support s in the new database, i.e., $DB \cup db$.

Let d be the number of transactions in db. With respect to the same minimum support s, an itemset X is large in the updated database $DB \cup db$ if the support of X in $DB \cup db$ is no less than s, i.e., $X.support_{UD} \geq s \times (D + d)$, where $X.support_{UD}$ denotes the support counts for itemset X in $DB \cup db$. In the following, we use $X.support_{DB}$ and $X.support_{db}$ to represent the support counts of an itemset X in DB and db, respectively. Similarly, we denote with L_{db} and L_{UD} the set of large itemsets in db and $DB \cup db$ respectively. Thus the essence of incremental mining of association rules is to find the set L_{UD}. Note that an itemset in L_{DB} may not be an itemset in L_{UD}. Also, an itemset not in L_{DB}, may become a large itemset in L_{UD}. We will use the above definitions of symbols in Chapter 3 of this thesis.

Recently, there have been some studies on the incremental mining of association rules [23, 8, 1, 35]. Cheung et al. developed an approach known as the FUP algorithm [8]. The FUP algorithm mainly incorporates the classical level-wise Apriori algorithm [1] to find large itemsets and their support counts in the original as well as incremental databases. It demonstrates good performance over purely Apriori technique. Thomas et al. developed another approach which makes use of the *negative border* of the original database [35]. It incrementally updates the negative borders when a set of new transactions is available. Although this additional information allows more efficient algorithms to be applied, it induces a considerable amount of overhead in calculating the negative border.

2.2.1 The FUP Algorithm

Cheung et al. introduced the fast update algorithm (FUP) for the problem of incremental association rule mining [8]. It is mainly based on the Apriori algorithm and achieves satisfactory performance.

In each iteration of FUP, only candidate itemsets of one size are handled. It

is performed by making one pass over the incremental database and one pass over the original database. In general, FUP finds all the size k large itemsets in the $k-th$ pass over the whole database and generates size $k+1$ candidates at the end of this pass. Therefore, the number of passes over the original database equals the size of the highest order itemsets to be updated.

Note that the major target of the FUP algorithm is to minimize the amount of candidate itemsets to be generated and counted in each pass. This is achieved by pruning the candidates in each pass through the incremental and original databases.

Several important lemmas have been mentioned in the work by Cheung et al. [8] regarding the itemsets to be considered for in the problem of incremental mining. These are the major source of improvement of the FUP algorithm over the apriori algorithm. We will simplify and summarize the ideas of these lemmas when we apply them in our algorithms in Chapter 3.

2.2.2 The FUP2 Algorithm

Cheung at el. also developed an algorithm called FUP2, which can not only handle the insertion of transactions to the database but also the deletion of transactions [9]. The part of FUP2 for handling the insertion of transactions is actually equivalent to the FUP algorithm.

The way the FUP2 algorithm handles deletion of transactions is as follows. It works iteratively, where large k-itemsets in the updated database are found in the k-th iteration. Size k candidate itemsets are generated from large $(k-1)$-itemsets in the updated database, while size 1 itemsets are the set of all items. All the size k candidates are counted against the deleted transactions. If a k-itemset is large in the original database, it's updated count can be found by

simply subtracting its original count with its count in the deleted transactions. On the other hand, if a k-itemset is small in the original database, there exist two possibilities. First, if it is large in the deleted transactions, it must not be large in the updated database and is thus removed from the candidates to be considered. Otherwise, it is small in the deleted transactions, and is added to the remaining set of candidates to be counted in the updated database. After counting the candidates against the updated database, their support can be calculated and all size k large itemsets in the updated database are found. The algorithm then proceeds to finding of size $k+1$ large itemsets in the updated database.

2.2.3 The FUP* Algorithm

Cheung at el. introduced an improved version of the FUP algorithm, called the FUP* algorithm [11]. In the FUP algorithm, the 0 size k itemsets to be counted are generated by the apriori-gen from the set of large $(k-1)$-itemsets in the updated database. In the FUP* algorithm, the candidate itemsets are generated from the set of large $(k-1)$-itemsets whose support counts in the incremental database are larger than or equal to the support threshold. This can be done because an originally small itemset can become large in the updated database only if it is large in the incremental database. By applying this technique, the candidate itemsets to be counted can be reduced. However, note that the number of passes over the original database that FUP* has to perform is still equal to the maximum size of the new large itemsets in the updated database.

In addition to the FUP* algorithm, Cheung at el. also introduced an algorithm for mining multiple-level association rules in the same paper, called the MLUp algorithm [11].

2.2.4 The Negative Border Method

An Efficient Algorithm for the incremental updation of association rules was also proposed by Thomas et al. [35]. The algorithm is based on *negative border* and requires one full scan of the whole database only when the update (insertion or deletion of transactions) causes the negative border of the large itemsets to expand.

To apply this algorithm, the support counts of both the large itemsets in the original database and in the negative border of the large itemsets have to be stored. The negative border is actually the set of all itemsets that were candidates of the level-wise large itemset discovery method which did not have enough support. In other words, the negative border of the set of all large itemsets actually equals all the itemset candidates generated in every levels by the apriori-gen (if Apriori is considered) minus the set of large itemsets in the database.

Let db, DB and $DB+$ be the incremental database, the original database and the updated database, respectively. Also, let L_{db}, L_{DB} and L_{DB+} be the set of large itemsets in the incremental database, the original database and the updated database. Let $NBd(L)$ denotes the negative border of the set L of itemsets. The algorithm works as follows in the case of insertion of transactions. First, all large itemsets in the db are found by any level-wise algorithm, such as the Apriori algorithm [1] or the Partition algorithm [27, 36]. Simultaneously, the support for all the itemsets in L_{DB} and $NBd(L_{DB})$ are counted. If the new support of an itemset in L_{DB} becomes smaller than the threshold, it is removed from L_{DB}. Since this may change the set L_{DB} the negative border may have to be recomputed. On the other hand, some new itemsets may become large in the updated database. According to Thomas et al., an itemset gets added to L_{DB+} must have some subsets of it moved from $NBd(L_{DB})$ to L_{DB+} [35].

Therefore, each itemset in L_{db} is checked if it gets the minimum support to move from the negative border of the original large itemsets $NBd(L_{DB})$ to L_{DB+}. If none of the itemsets in $NBd(L_{DB})$ get the minimum support, no new itemset for L_{DB+} exists. If an itemset in $NBd(L_{DB})$ gets the minimum support, it is added to L_{DB+} and the negative border is recomputed. If $L_{DB+} \cup NBd(L_{DB+}) \neq L_{DB} \cup NBd(L_{DB})$, the whole original database has to be scanned once to find the updated large itemsets and negative border, For this pass, the candidate to be counted will be the negative border closure of L which is found by repeatedly finding $L = L \cup NBd(L)$ until L does not grow.

The deletion of existing transactions from the database can be handled similar to the case of insertion.

While only a single pass over the original database is guaranteed in the worst case, the candidates itemsets to be counted in the pass can be huge. In addition, a lot of calculations have to be done in the computing of the negative borders in the algorithm. These greatly reduce the efficiency of the algorithm. Also, to use the algorithm, the support counts for all the large itemsets and the negative border have to be stored, which is actually a waste of storage.

2.2.5 Limitations of Existing Incremental Association Rule Mining Algorithms

The FUP algorithm introduced by Cheung et al. [8] is mainly based on the Apriori algorithm. In each pass over both the increment and the original database, only candidate itemsets of one size are handled. Therefore, the number of passes over the original database equals the size of the highest order itemsets to be updated. It incurs much I/O time in reading the database and hence is not efficient.

The FUP2 algorithm can handle both the insertion and deletion of transactions from the database [9]. It handles the insertion of transactions in the same way as the FUP algorithm. However, the number of passes over the original database and the incremental database is still equal to the maximum size of the new large itemsets in the updated database. Therefore, the I/O time incurred is similar to the FUP algorithm that the FUP2 algorithm is also not efficient.

The FUP* algorithm introduced by Cheung et al. reduces the number of candidates to be counted compared to the FUP algorithm [11]. However, even though it is better than the FUP algorithm in terms of the execution time, the number of passes over the database that FUP* has to perform is still equal to the maximum size of the new large itemsets in the updated database.

The negative border algorithm introduced by Thomas et al. [35] guarantees a single pass over the original database in the worst case. However, the candidates itemsets to be counted in the pass can be huge. Also, much execution time is consumed by the algorithm in the computing of the negative borders. This greatly reduces the efficiency of the algorithm. 0, the support counts for all the large itemsets and the negative border have to be stored for the use of the algorithm, which requires extra storage.

Chapter 3

A New Incremental Association Rule Mining Approach

3.1 Outline for the Proposed Approach

Dynamic counting technique has been explored by Brin et al. who developed the DIC algorithm for itemset counting in a *single* database [5]. While Apriori requires the same number of passes through the database as the size of the maximum large itemsets, dynamic counting allows candidates of various sizes to be counted at the same time in each pass. This results in the reduced number of passes through the database.

One previous work for incremental association rule mining is the FUP algorithm [8]. It is mainly based on the Apriori algorithm and achieves satisfactory performance. In each iteration of FUP, only candidate itemsets of one size are handled. It is performed by making one pass over the increment db and one pass over the original database DB. Therefore, the number of passes over the original database equals the size of the highest order itemsets to be updated. To reduce the passes over the DB, we developed two new Incremental Dynamic Itemset

Counting algorithms, the IDIC_M and the IDIC_S, which are based on dynamic counting. In both algorithms, we compress the passes over *db* performed in the FUP algorithm by dynamic counting technique. On the other hand, without checking for each k-itemset candidate found immediately after one pass through *db* against DB, a lot more candidates are generated in each of the next pass through db and hence, the total execution time used to scan *db* may increase. Also, a large amount of candidate itemsets are generated after scanning *db*. Two different methods are used respectively to handle this problem. In the IDIC_M algorithm, we compress the passes over DB by apply dynamic counting technique. We allow only the k-itemsets with all its size $k-1$ subsets being large in $DB \cup db$ to be counted in DB. This means that the candidates actually being counted in DB are reduced to only the essential ones and there are not too many of them. However, note that it may need to perform more than one passes over DB. In the other variant algorithm, IDIC_S, we perform exactly one scan over DB to count all the candidate itemsets generated from *db*. While some excessive candidates may be counted, the number of passes over DB is guaranteed to be minimum. By using either of our proposed algorithms, it is expected that the execution time saved in scanning DB should be much larger than the increased time in counting the extra candidates in *db*.

Note that in the IDIC_M algorithm, it is not efficient if we apply the original DIC algorithm directly in counting DB for our incremental counting problem. Since, for the original DIC algorithm, we have to start counting only 1-itemsets in the first interval and cannot start counting k-itemsets until the k-th interval. This limitation is not desirable in our case since we already know the candidates to be checked in DB. Actually, once we have discovered all potential L_{UD} candidates after scanning *db*, we can just make one pass through DB to check the counts of all those candidates. One possible drawback for such approach is that a large amount of unnecessary candidates may be counted in DB, causing considerable

overhead. In contrast, our IDIC_M algorithm will start counting any candidates in DB as soon as it is proven eligible (i.e., it has all its subsets being large in $DB \cup db$). For example, if an 4-itemset has all four of its size-3 subsets being large in L_{UD}, we can start counting it immediately in the first interval of DB instead of counting it in the fourth interval.

Remember that the major target of the FUP algorithm is to minimize the number of candidate itemsets to be generated and counted. This is done by pruning the candidates in each pass through db and DB. On the other hand, our two algorithms allow extra candidate itemsets to be generated from db, while the efficiency is maintained by the reduced passes through DB. Usually, a large amount of candidates are generated from db and we have to be efficient when we count those candidates in DB. Our two variant algorithms try to deal with this problem in two different ways. Also note that we cannot trivially incorporate DIC into the FUP algorithm, since DIC is not a level-wise algorithm and cannot be used directly in place of Apriori in FUP.

Our algorithms are dynamic approaches since they apply the dynamic counting technigues in some ways to increase the efficiency of the mining process. Our algorithms are incremental ones since they are used in the incremental mining of association rules. Or more accurately, they are used to update the set of new large itemsets when the database is incremented with a set of new transactions. Our algorithms are dynamic incremental approaches since we have applied the dynamic counting techniques in some ways in our incremental mining algorithms. To make our incremental algorithms efficient, we try to reduce the large number of passes over the original database, as well as the incremental database, as in the case of FUP algorithms. We try to achieve this by avoiding the level-wise structures of the existing incremental mining algorithms. On the other hand, we try to apply the dynamic counting technique so that the counting process itself can also be efficient.

3.2 Our New Approach

3.2.1 The IDIC_M Algorithm

We propose a new algorithm, the IDIC_M algorithm, for incremental mining based on dynamic itemset counting.

Lemma 1 An itemset X is in L_{UD} only if it is either in L_{DB} or in L_{db} or in both L_{DB} and L_{db}. It means that all large itemsets in the updated database, i.e., L_{UD} must be included in $L_{DB} \cup L_{db}$.

Proof. If X is neither in L_{DB} nor L_{db}, then $X.support_{DB} < s \times D$ and $X.support_{db} < s \times d$. Therefore, $X.support_{DB} + X.support_{db} < s \times (D + d)$. Then X is not in L_{UD}.

The IDIC_M algorithm works as follows:

1. **Scan the** *db*

   Scan the increment *db* and use DIC to find all itemsets which are large in *db*, i.e., L_{db}. For each X in L_{db}, record its support in *db* as $X.support_{db}$.

   Example 1 In this example, we explain why we have the support counts of some itemsets but not the others after Step 1. Note that we only keep the counts of some small itemsets in *db*. Say, in Step 1, when we are counting *db*, we find itemsets $\{2,3\}$, $\{3,4\}$ and $\{2,4\}$ to be large in L_{db} and we start to count itemset $\{2,3,4\}$. However, we finally find that $\{2,3,4\}$ is small in *db*. Then we will have the count for itemset $\{2,3,4\}$ even it is not large in L_{db}. On the other hand, if we find one of the itemsets $\{2,3\}$, $\{3,4\}$ and $\{2,4\}$ to be small in *db*, we will not try to count itemset $\{2,3,4\}$ and so will not have the count for itemset $\{2,3,4\}$.

2. **Consider itemsets in** L_{DB}

For each itemset X in L_{DB}, if $X.support_{DB} + X.support_{db} \geq s \times (D + d)$, it is added to the set of large itemsets in the updated database $DB \cup db$, i.e., W_1, with the updated count. Note that the itemsets we examine in this step already includes all itemsets in $L_{DB} \cap L_{db}$ which can be shown in the following.

Lemma 2 All itemsets in $L_{DB} \cap L_{db}$ are examined in Step 2 of the IDIC_M algorithm.

Proof. From the description, we check each itemsets in L_{DB} if we have its count $X.support_{db}$ (Any itemset in L_{DB} we already have its count $X.support_{DB}$). For all itemsets in L_{db} we should have their counts in db from Step 1. Therefore, any itemset that is both large in DB and large in db has been examined in Step 2.

Example 2 Assume a database DB with size $D = 100000$ is updated with an increment db with size $d = 1000$, where θ_s is 1%. Let I_1, I_2, I_3 and I_4 be four itemsets in L_{DB}, with $I_1.support_{DB} = 1005$, $I_2.support_{DB} = 1007$, $I_3.support_{DB} = 1007$ and $I_4.support_{DB} = 1005$, respectively. Also, we have $I_1.support_{db} = 15$, $I_2.support_{db} = 8$ and $I_3.support_{db} = 2$ after Step 1. However, we do not have the support of I_4 in db after Step 1. Note that I_1 is an large itemset in L_{db} while I_2, I_3 and I_4 are not. In step 2, what we do is to check each of the four itemsets if it should be added into W_1. First, since $I_1.support_{UD} = I_1.support_{DB} + I_1.support_{db} > 101000 \times 1\%$, I_1 will be added into W_1. Similarly, $I_2.support_{UD} = I_2.support_{DB} + I_2.support_{db} > 101000 \times 1\%$, and I_2 will also be added into W_1. However, as $I_3.support_{UD} = I_3.support_{DB} + I_3.support_{db} < 101000 \times 1\%$, I_3 will not be added into W_1. Since we cannot determine whether I_4 is large in UD, it will be handled in Step 3 of the algorithm.

3. **Consider itemsets in $L_{DB} - L_{db}$**

Check all itemsets in $L_{DB} - L_{db} - W_1$, i.e., those itemsets that are large in

DB but not large in db and not already large in $DB \cup db$, by scanning db once to update their counts. For each itemset X in $L_{DB} - L_{db} - W_1$, add it into W_2 with the updated count if $X.support_{DB} + X.support_{db} \geq s \times (D+d)$. Note that this step is necessary as we did not keep all the counts for those itemsets which are not large in db. Set $W_2 = W_1 \cup W_2$.

Example 3 We continue this example from Example 2. Remember we have a database DB with size $D = 100000$ updated with an increment db with size $d = 1000$, where θ_s is 1%. Four itemsets I_1, I_2, I_3 and I_4 are in L_{DB}, with $I_1.support_{DB} = 1005$, $I_2.support_{DB} = 1007$, $I_3.support_{DB} = 1007$ and $I_4.support_{DB} = 1005$, respectively. We found $I_1.support_{db} = 15$, $I_2.support_{db} = 8$ and $I_3.support_{db} = 2$, but we do not have the support $I_4.support_{db}$ after the first two steps. Now, as I_4 is in L_{DB} but not in L_{db} and W_1, so we have to count it once in Step 3. I_1 is not counted in Step 3 since it is handled in Step 2 and is already in W_1. Though I_2 is in L_{DB} and not in L_{db}, it is not counted in Step 3 since is handled in Step 2 and is already in W_1. Note that I_3 is already rejected in Step 2.

4. **Consider itemsets in $L_{db} - L_{DB}$**

In this step, we determine which itemsets in $L_{db} - L_{DB} - W_2$ are large in $DB \cup db$. We do this by applying a dynamic counting technique, similar to the DIC algorithm, to DB. Before counting, each k-itemset in $L_{db} - L_{DB}$, which has all its possible size $(k-1)$-subsets being in W_2, is added to the candidate set to be counted, provided that it is not already in W_2. Note that any k-itemset can contain at most k subsets of size $k-1$. All 1-itemsets in $L_{db} - L_{DB} - W_2$ are also added to the candidate set. The counts of all the candidates are initialized with their supports in db. After the counting of each interval is finished, new large itemsets are identified and added to W_2 with the updated supports. New candidates are generated from W_2 by a function *apriori-gen*, developed by Agrawal et al., which takes as input

the set of all large $(k-1)$-itemsets and returns a superset of the set of all large k-itemsets [1]. These new candidates are then added to the candidate sets if it is not already in W_2. Continue the process until there are no more candidates to be counted. Finally, output the set W_2 of itemsets as L_{UD}.

Example 4 Assume we now have in W_2 1-itemsets $\{1\}$, $\{2\}$ and $\{3\}$, 2-itemsets $\{1,2\}$, $\{2,3\}$, $\{5,6\}$, $\{6,7\}$ and $\{5,7\}$. Also, assume that we have itemsets $\{4\}$, $\{1,3\}$, $\{1,2,3\}$ and $\{5,6,7\}$ in L_{db}. In Step 4, we start to count $\{1,3\}$ in the first interval because it has 2 of its 1-subsets $\{1\}$ and $\{3\}$ in W_2. Similarly, we start to count $\{5,6,7\}$ since its 3 2-subsets $\{5,6\}$, $\{6,7\}$ and $\{5,7\}$ are in W_2. The 1-itemset $\{4\}$ is also counted in the first interval. If we find $\{1,3\}$ to be large after, say, counting the first interval, then we will start to count $\{1,2,3\}$ in the second interval, as all 3 of its 3-subsets $\{1,2\}$, $\{2,3\}$ and $\{1,3\}$ are now large in W_2.

Lemma 3 All potential L_{UD} candidates are examined by the IDIC_M algorithm.

Proof. After the above four steps, we have checked all the itemsets in $L_{DB} \cap L_{db}$, L_{DB} and L_{db}. That is all itemsets in $L_{DB} \cup L_{db}$ have already been examined. We have covered all potential L_{UD} candidates, by Lemma 1.

Note that although the above lemmas are similar to those introduced in [8], they give a more clear view of what can be covered in each steps of our algorithms.

3.2.2 A Variant Algorithm: The IDIC_S Algorithm

We now introduce a variant algorithm to the IDIC_M algorithm, called the IDIC_S algorithm. While there may be more than one passes through DB in the IDIC_M algorithm, the IDIC_S algorithm guarantees a single pass over the DB. The first three steps of the IDIC_S algorithm are exactly the same as the

IDIC_M algorithm. The Step 4 of the IDIC_S algorithm works as follows:

In this step, we determine which itemsets in $L_{db} - L_{DB} - W_2$ are large in $DB \cup db$. Instead of applying a dynamic counting technique as in the IDIC_M algorithm, we count all these candidate itemsets in one pass over DB. After the counting is finished, new large itemsets are identified and added to W_2 with the updated supports. Finally, output the set W_2 of itemsets as L_{UD}.

Note that all the lemmas in the previous subsection also applies in the IDIC_S algorithm, which means that all potential L_{UD} candidates are examined by the IDIC_S algorithm.

3.3 Performance Evaluation of Our Approach

3.3.1 Experimental Results for Algorithm IDIC_M

Several sets of experiments were conducted to compare the IDIC_M algorithm with the FUP algorithm and the pure DIC algorithm. Note that each data point in the experimental results is obtained from the average value over 10 trials. The synthesis data used in all the experiments are generated following the techniques in [8].

D	Number of transactions in database DB
d	Number of transactions in the in increment db
$\|T\|$	Mean size of the transactions
$\|I\|$	Mean size of the maximal potentially large itemsets
$\|L\|$	Number of potentially large itemsets
N	Number of items

Table 3.1: Parameter Table.

We use the notation Tx.Iy.Dm.dn to denote a database of size $D = 1000m$ is

updated with an increment with size $d = 1000n$, while $|T| = x$ and $|I| = y$. In our experiments comparing the IDIC_M algorithm and the FUP algorithm, D and d were chosen in similar way as in [8], and $|T| = 10$, $|I| = 4$, $|L| = 2000$ and $N = 1000$.

We generated $(D + d)$ transactions by one random seed. Then, the first D transactions are stored for DB and the others for db. Note that in the second set of experiments in Section 3.4.2, DB and db were generated separately with different random seeds. For our IDIC_M algorithm, the FUP algorithm and the DIC algorithm, their execution time speedup ratios over pure Apriori algorithm are calculated to evaluate their performance. Note that pure Apriori algorithm here means applying the Apriori algorithm once over the updated database. We first combine the original database and the incremental database into a new updated database. Then, the Apriori algorithm is applied on this new updated database to obtain the new large itemsets we want. Speedup ratio data point on the curve **Apriori/IDIC_M** is calculated by dividing the execution time of the pure Apriori algorithm with the execution time of the IDIC_M algorithm.

IDIC_M versus FUP for Different Supports

The IDIC_M algorithm and the FUP algorithm were tested for different θ_s ranging from 0.01 to 0.05, with updated database T10.I4.D100.d1 which is similar to the one used in [8]. The interval sizes for counting db and DB were fixed at 1000 and 10000, respectively.

The result is shown in Figure 3.1. The IDIC_M algorithm clearly has a better execution time speedup ratio for all supports tested. Note that, our algorithm has a substantial advantage over the FUP algorithm when the minimum support threshold is set at at a relatively higher value. The speedup ratio of the IDIC_M algorithm is more than 1.5 times higher than the FUP algorithm when $\theta_s =$

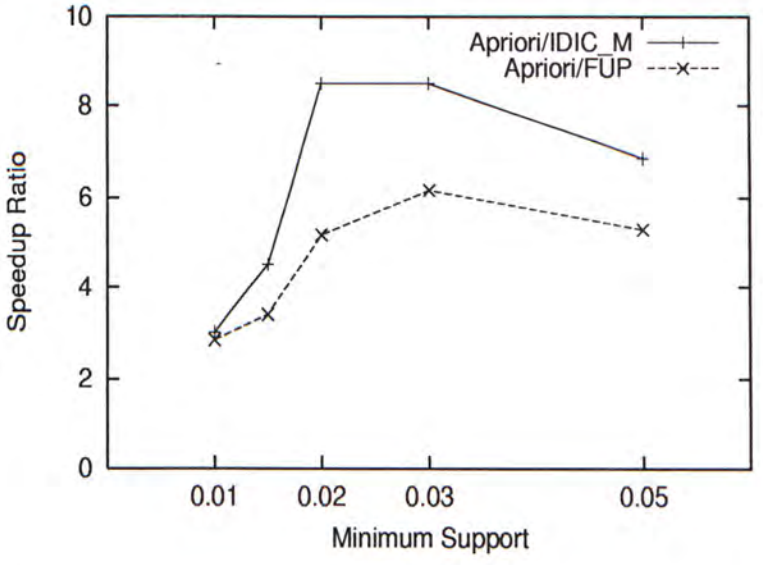

Figure 3.1: IDIC_M and FUP for Different Supports

0.02. The IDIC_M algorithm is more than 8 times faster than the pure Apriori algorithm at this support threshold value. We can see that when the support threshold is low, the IDIC_M algorithm does not have a very significant advantage over the FUP algorithm. Since a very large number of candidate itemsets are expected to be generated from *db* in this case, we can observe that IDIC_M may not be very efficient in dealing with those large amount of candidates.

IDIC_M versus FUP for Different Incremental Sizes

We have done two sets of experiments to compare the performance of the IDIC_M algorithm with the FUP algorithm for different incremental size, d, ranging from 1000 to 50000, with DB fixed at 100000, i.e., T10.I4.D100.dn. θ_s is fixed at 0.015. For the first set of experiments, the DB and db are generated as usual using the same random seed. While for the second set of experiments, the DB and db are generated separately using different random seeds.

The results for first set of experiments are shown in Figure 3.2. The IDIC_M

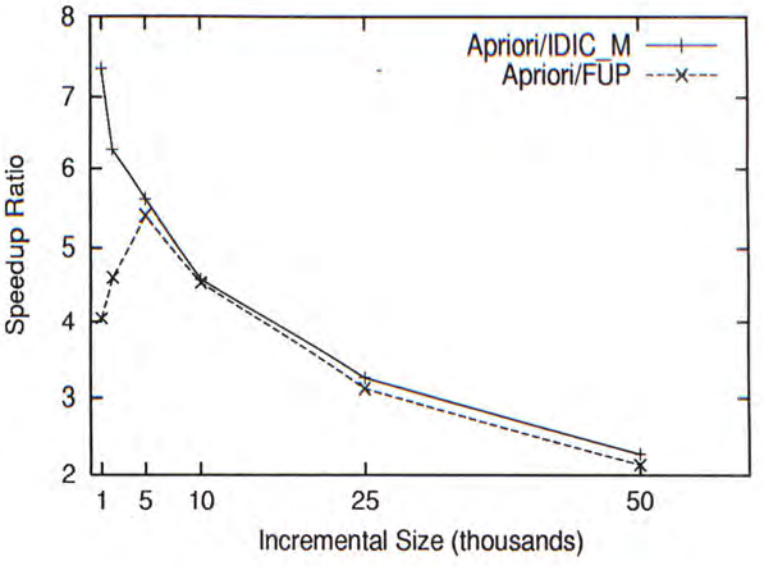

Figure 3.2: IDIC_M and FUP for Different Incremental Sizes (with the same random seed)

algorithm outperforms the FUP algorithm for all incremental sizes tested. It works well especially when the incremental size is small. The greatest advantage of our algorithm over the FUP algorithm was obtained when $d = 1000$, where the speedup ratio of IDIC_M is about 7.3 and the speedup ratio of FUP is about 4. We see the trend that as the incremental size continues to increase, the performances of both the IDIC_M algorithm and the FUP algorithm decrease. This is due to the fact that when the incremental size is large relative to the original database size, the value of the known information about the original database will be lower. This experiment can simulate situations where the incoming data are having similar patterns to the original database.

The results for second set of experiments are shown in Figure 3.3. The IDIC_M algorithm also outperforms the FUP algorithm for all incremental sizes tested. Our algorithm has a considerable advantage over the FUP algorithm except when the incremental size is near to 10000. The greatest advantage of our algorithm over the FUP algorithm was also obtained at $d = 1000$. Note that the curve of the IDIC_M algorithm converges to a higher value than the curve

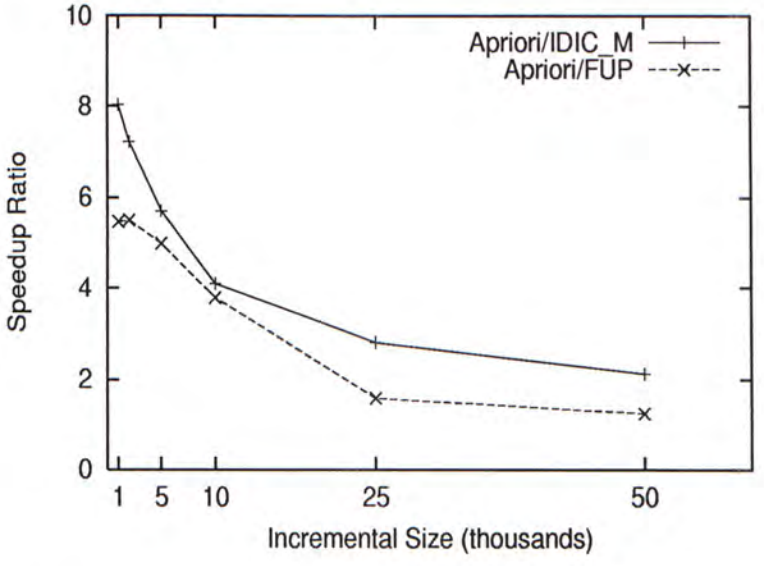

Figure 3.3: IDIC_M and FUP for Different Incremental Sizes (with different random seeds)

of FUP does. Also, note that our IDIC_M algorithm is much better than the FUP algorithm in this situation where the original database and the incremental database were generated by different random seeds. The situation simulated by this experiment does exist in real life when each batch of new incoming data have a different transaction pattern.

IDIC_M versus DIC for Different Incremental Sizes

Experiments were done to compare the performance of the IDIC_M algorithm with the DIC algorithm for different incremental size, d, ranging from 10000 to 50000, with DB fixed at 50000, i.e., T10.I4.D50.dn. θ_s is fixed at 0.015. We are especially interested in the situation that when the incremental size is relative large compared to the original database size. Note that what we mean by DIC here is that we applied the DIC algorithm once over the updated database.

The results for the experiments are shown in Figure 3.4. Although the advantage is decreasing with the incremental size, our IDIC_M algorithm outperforms

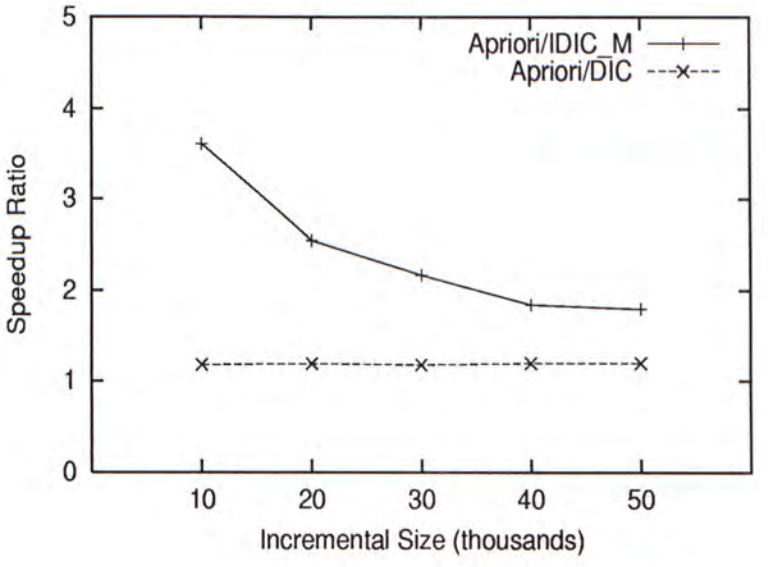

Figure 3.4: IDIC_M and DIC for Different Incremental Sizes

the DIC algorithm for all incremental sizes tested with considerable amount. The decreasing performance of IDIC_M is due to the less significant of known information about the original database as increment becomes larger. Since DIC is not an incremental mining algorithm, it can maintain a relatively steady advantage over the Apriori algorithm. From the results, we can conclude that the advantage of our IDIC_M algorithm over the FUP algorithm is not only contributed by the use of dynamic counting technique, but also the whole design of our IDIC_M algorithm.

3.3.2 Experimental Results for Algorithm IDIC_S

Several sets of experiments were conducted to compare the IDIC_S algorithm with the FUP algorithm and the pure DIC algorithm. The data used in this subsection for testing the IDIC_S algorithm is similar to that used to test the IDIC_M algorithm in Section 3.4.

IDIC_S versus FUP for Different Supports

The IDIC_S algorithm and the FUP algorithm are tested for different θ_s ranged from 0.01 to 0.05, with updated database T10.I4.D100.d1 which is similar to the one used in [8]. The interval sizes for counting *db* and *DB* were fixed at 1000 and 10000, respectively.

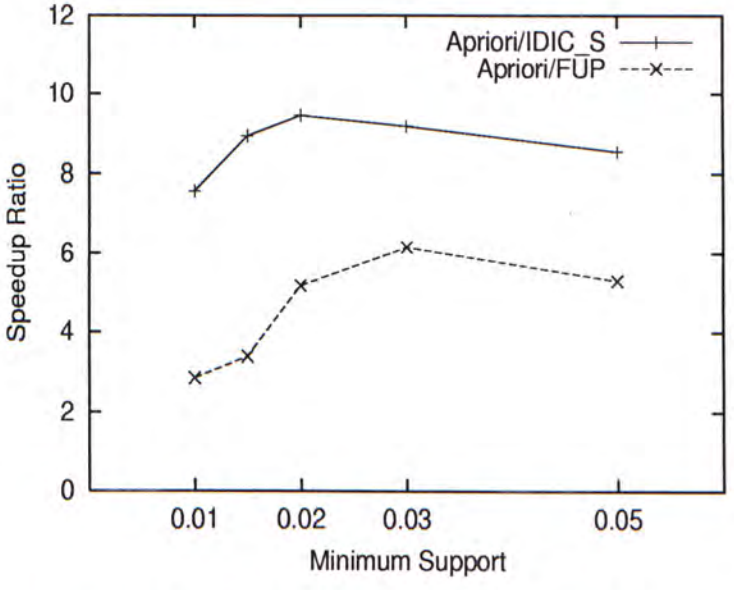

Figure 3.5: IDIC_S and FUP for Different Supports

The result is shown in Figure 3.5. The IDIC_S algorithm clearly has a better execution time speedup ratio for all supports tested. Note that, our algorithm has a steady advantage over the FUP algorithm at different minimum support thresholds. The difference between the speedup ratio of the IDIC_S algorithm and that of the FUP algorithm is more than 3 for all the supports tested. Compared to the IDIC_M algorithm, we can say that the IDIC_S algorithm is much more efficient in dealing with the large number of candidate itemsets generated from *db* when the support threshold is low. It also has a good performance when the support threshold is high.

IDIC_S versus FUP for Different Incremental Sizes

We have done two sets of experiments to compare the performance of the IDIC_S algorithm with the FUP algorithm for different incremental size, d, ranging from 1000 to 50000, with DB fixed at 100000, i.e., T10.I4.D100.dn. θ_s is fixed at 0.015. For the first set of experiments, the DB and db are generated as usual using the same random seed. While for the second set of experiments, the DB and db are generated separately using different random seeds.

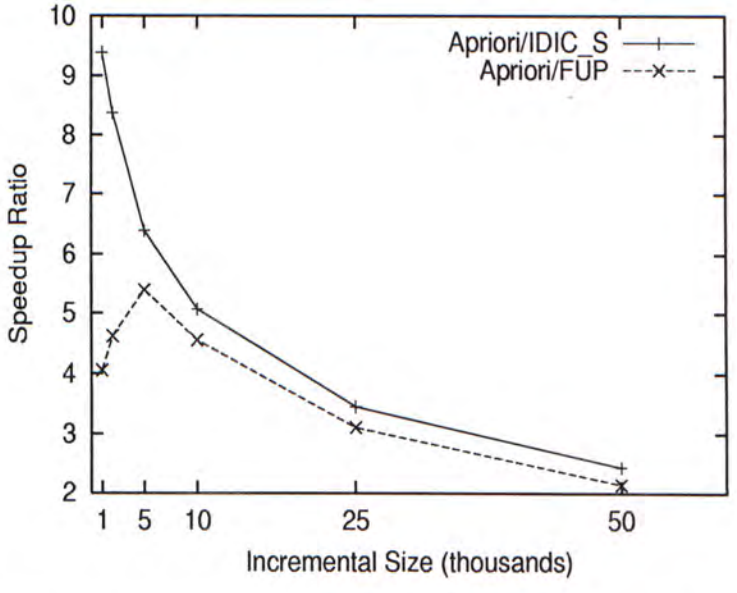

Figure 3.6: IDIC_S and FUP for Different Incremental Sizes (with the same random seed)

The results for first set of experiments are shown in Figure 3.6. The IDIC_S algorithm clearly outperforms the FUP algorithm for all incremental sizes tested, especially when the incremental size is small. The greatest advantage of our algorithm over the FUP algorithm was obtained when $d = 1000$, where the speedup ratio of IDIC_S is more than twice the speedup ratio of FUP. We also see the trend that as the incremental size continues to increase, the performances of both the IDIC_S algorithm and the FUP algorithm decrease.

The results for second set of experiments are shown in Figure 3.7. The IDIC_S

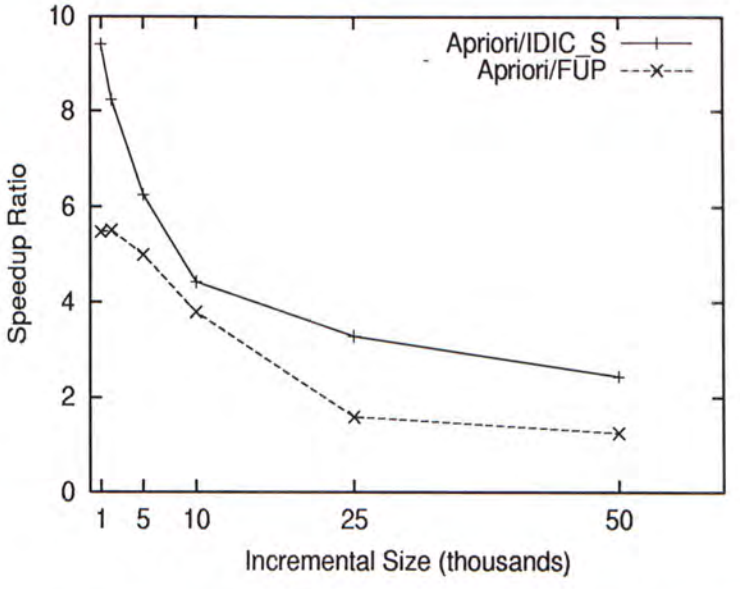

Figure 3.7: IDIC_S and FUP for Different Incremental Sizes (with different random seeds)

algorithm also outperforms the FUP algorithm for all incremental sizes tested. Note that the curve of the IDIC_S algorithm again converges to a higher value than the curve of FUP does.

IDIC_S versus DIC for Different Incremental Sizes

Experiments were done to compare the performance of the IDIC_S algorithm with the DIC algorithm for different incremental size, d, ranging from 10000 to 50000, with DB fixed at 50000, i.e., T10.I4.D50.dn. θ_s is fixed at 0.015.

The results for the experiments are shown in Figure 3.8. Similar to the results for the IDIC_M algorithm, the advantage of the IDIC_S algorithm is decreasing with the incremental size. Our IDIC_S algorithm shows significant advantage over the DIC algorithm. We can have the conclusion similar to the IDIC_M algorithm that dynamic counting technique is not the only cause of the efficiency of our IDIC_S algorithm.

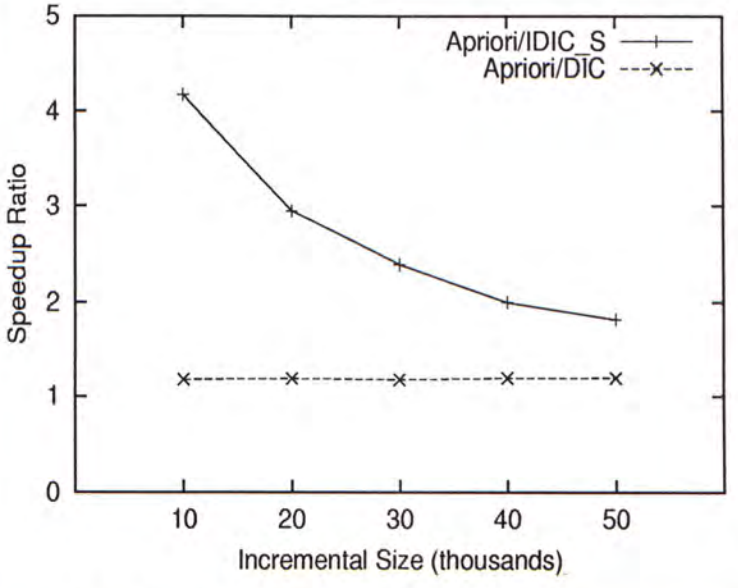

Figure 3.8: IDIC_S and DIC for Different Incremental Sizes

3.4 Discussion

Although we did not include in the thesis the direct comparison on the performances between the algorithms IDIC_M and IDIC_S, it has been observed that the IDIC_S algorithm actually has better performance over the IDIC_M algorithm in most of the experiments that we have done. However, we cannot come up with any conclusions on at what particular occasions does the IDIC_M algorithm work better.

The main objective for doing the experiments comparing the IDIC_M and IDIC_S algorithms with the DIC algorithm is that we want to show our algorithms really work well as incremental mining algorithms. The advantage of our proposed algorithms over the FUP algorithm is not only contributed by the use of dynamic counting technique, but also the whole design of our algorithms.

In this chapter, we have proposed new incremental updating algorithms, the IDIC_M and the IDIC_S, to handle the problem of incremental association rule mining using dynamic counting technique. We describe in details our new algo-

rithms and illustrates how dynamic counting works efficiently in this problem. We have fully implemented our algorithms. Experimental results show that our new algorithms have superior performance in comparison with another recent incremental updating algorithm, the FUP algorithm.

Chapter 4

Related Work on Multiple-Level AR and Belief-Driven Mining

4.1 Background on Multiple-Level Association Rules

Taxonomies over the items are often available in many application domains. Taxononies are in form of $is-a$ hierarchy. For example, Figure 4.1 is a taxonomy on food items in the supermarket domain.

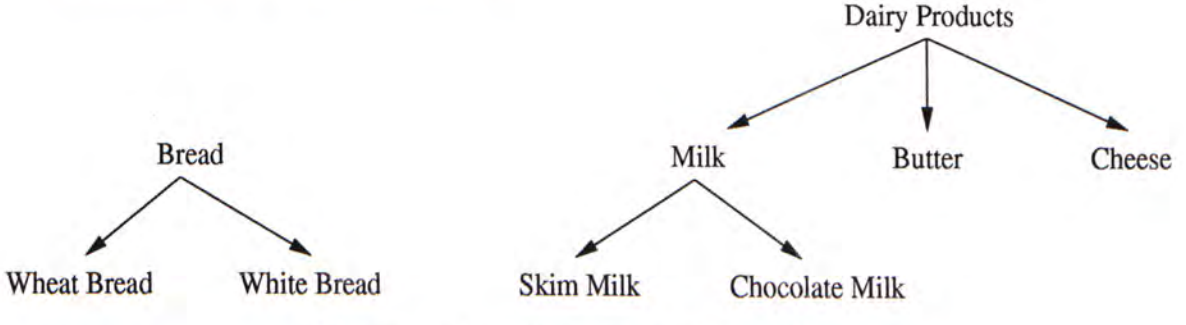

Figure 4.1: A Hierarchy of Food Items

In most situations, only the items in the leaf nodes of the taxonomy appear in the database. Therefore, if the taxonomy is not considered, we may be able

to find rules such as $WheatBread \Rightarrow SkimMilk$ or $WhiteBread, Cheese \Rightarrow ChocolateMilk$. However, we are not able to discover rules like $WheatBread \Rightarrow Milk$ since the item $Milk$ does not appear in the database originally.

One intuitive way to incorporate taxonomy in the mining of association rules is to expand each item in each transaction with their ancestor items in the hierarchy, as proposed in the work by Srikant et al. [32]. For example, assume a transaction originally contains the items $WheatBread$, $SkimMilk$ and $Cheese$. Then after expansion, the transaction will contain items $WheatBread$, $Bread$, $SkimMilk$, $Milk$, $Cheese$ and $DairyProducts$. After 0 the database with the hierarchy, we can now discover rules like $WheatBread \Rightarrow Milk$, which we can call multi-level association rules or generalized association rules.

4.2 Related Work on Multiple-Level Association Rules

Related research on the mining of multiple-level association rules were done by Srikant et al. [32] and Han et al. [15]. The former group introduced a measure of interestingness of rules to allow efficient mining, while the later group encoded the database in a special way and then mining is performed by extension of some existing association rule mining techniques.

4.2.1 The Basic Algorithm

In the work by Srikant et al., methods for mining multiple-level association rules, together with some definition regarding the interestingness of multiple-level association rules, are introduced [32]. The main ideas are described as follows.

An itemset $\hat{Z}$ is an ancestor of another itemset Z if we can get $\hat{Z}$ from Z by replacing one or more items in Z with their ancestors and Z and $\hat{Z}$ have the same number of items. $\hat{X} \Rightarrow Y$, $X \Rightarrow \hat{Y}$ and $\hat{X} \Rightarrow \hat{Y}$ are called the ancestor of the rule $X \Rightarrow Y$. Given a set of rules, $\hat{X} \Rightarrow \hat{Y}$ is a *close ancestor* of $X \Rightarrow Y$ if there is no rule $X' \Rightarrow Y'$ such that $X' \Rightarrow Y'$ is an ancestor of $X \Rightarrow Y$ and $\hat{X} \Rightarrow \hat{Y}$ is an ancestor of $X' \Rightarrow Y'$. Similar definitions apply to $X \Rightarrow \hat{Y}$ and $\hat{X} \Rightarrow Y$. Consider a rule $X \Rightarrow Y$ where $Z = X \cup Y$. The expected value of $Pr(Z)$ given $Pr(\hat{Z})$ is denoted by $E_{\hat{Z}}[Pr(Z)]$ where $Pr(Z)$ is the support of Z. $\hat{Z}$ is an ancestor of Z and $Z = \{z_1, ..., z_n\}$ and $\hat{Z} = \{\hat{z}_1, ..., \hat{z}_j, y_{j+1}, y_n\}$ where $\hat{z}_i$ is an ancestor of z_i. It is defined that

$$E_{\hat{Z}}[Pr(Z)] = \frac{Pr(z_1)}{Pr(\hat{z}_1)} \times \cdots \times \frac{Pr(z_j)}{Pr(\hat{z}_j)} \times Pr(\hat{Z})$$

to be the expected value of $Pr(Z)$ given $\hat{Z}$. Similarly, $E_{\hat{X} \Rightarrow \hat{Y}}[Pr(Y|X)]$ denotes the expected confidence of the rule $X \Rightarrow Y$ given rule $\hat{X} \Rightarrow \hat{Y}$. $\hat{Y}$ is an ancestor of Y and $Y = \{y_1, ..., y_n\}$ and $\hat{Y} = \{\hat{y}_1, ..., \hat{y}_j, y_{j+1}, y_n\}$. It is defined that

$$E_{\hat{X} \Rightarrow \hat{Y}}[Pr(Y|X)] = \frac{Pr(y_1)}{Pr(\hat{y}_1)} \times \cdots \times \frac{Pr(y_j)}{Pr(\hat{y}_j)} \times Pr(\hat{Y}|\hat{X}).$$

A rule $X \Rightarrow Y$ is R-interesting w.r.t. an ancestor $\hat{X} \Rightarrow \hat{Y}$ if the support of $X \Rightarrow Y$ is R times the expected support based on $\hat{X} \Rightarrow \hat{Y}$, or the confidence is R times the expected confidence based on $\hat{X} \Rightarrow \hat{Y}$. Given a set of rules S and a minimum interest R, a rule $X \Rightarrow Y$ is interesting (in S) if it has no ancestors or it is R-interesting with respect to its close ancestors among its interesting ancestors.

The Basic algorithm by Srikant et al. is to simply add to each transaction all the ancestors of each item in the transaction, and then run any algorithm for large itemset discovery over the expanded database.

4.2.2 The Cumulate Algorithm

The Cumulate algorithm is an improved the Basic algorithm with three optimizations [32]. First, ancestors of the items that do not exist in the candidate itemsets need not be added to the transaction. Only the ancestors that exist in one or more of the candidate itemsets being counted need to be added to the transaction. For example, let us consider the hierarchy in Figure 4.1. Assume a transaction originally contains items $WheatBread$, $SkimMilk$ and $Butter$, and the only itemset we are counting contains $WheatBread$ and $DairyProducts$. We just need to add the ancestor $DairyProducts$ into the transaction. The ancestors $Bread$ and $Milk$ need not be added to the transaction. Second, the ancestors of each item are pre-computed instead of being found by traversing the taxonomy graph. Third, any itemsets that contain both an item and its ancestor can be pruned since its count will be equal to the count of the itemset without that ancestor item. For example, if the item $Bread$ is the ancestor of $WheatBread$. Then, in each transaction, $Bread$ must appear whenever $WheatBread$ appears. Therefore, the itemset $\{Bread, WheatBread\}$ must have the same counts as itemset $\{WheatBread\}$. These three techniques greatly improve the efficiency of the Cumulate algorithm compared to the Basic algorithm.

4.2.3 The EstMerge Algorithm

Further improvement to the Cumulate algorithm makes the EstMerge algorithm [32]. It is achieved by two techniques called *stratification* and *estimate* Stratification is based on the simple idea that if the ancestor item does not have the minimum support, then we do not need to count its descendants. Estimate means that we only count those candidate itemsets that are expected to have minimum support or itemsets that are not expected to have minimum support but have all of whose parents having minimum support. The latter itemsets are

expected not to have minimum support, so their descendants will not be counted. If some the itemsets that are not expected to have minimum support turn out to be large itemsets, then an additional pass is required to count their supports. Note that the estimated support of itemsets are found by sampling. Experiments show that the EstMerge algorithm works better that the Cumulate algorithm, which in turn works better than the Basic algorithm, in terms of the efficiency.

4.2.4 Using Hierarchy-Information Encoded Transaction Table

Another method for mining multiple-level association rules was introduced by Han et al. [15]. It is a top-down progressive deepening method developed by extension of some existing association rule mining techniques.

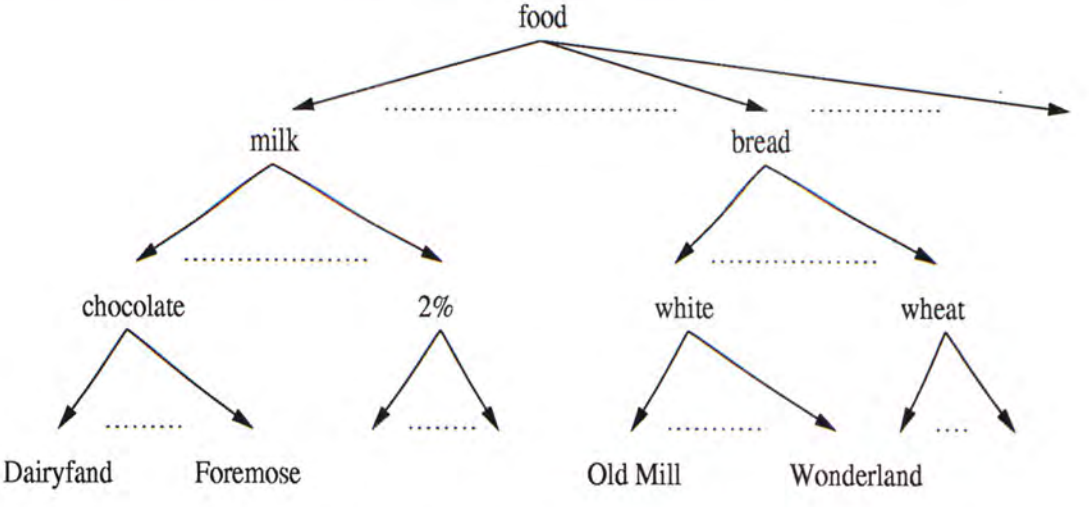

Figure 4.2: Another Hierarchy of Food Items

In the algorithm, all items in the transactions are encoded in a way such that the hierarchical information is stored. For example, let consider the food hierarchy in Figure 4.2. In the hierarchy, the root node is considered level-0, the level just below level-0 is level-1, two levels down the root node is level-2 and the lowest level is level-3. An item 'chocolate Foremose milk' is encoded as

'112' in which the first digit '1' represents 'milk' at level-1, the second digit '1' represents 'chocolate' at level-2 and the last digit '2' represents 'Foremose' at the lowest level. The algorithm then try to find large itemsets at level-1 items by considering items in the form '1**', '2**', etc. in the database. If we find that, say, only items '1**' and '2**' are large, then, the database will be filtered by retaining only items in the form of '1**' and '2**' for the second pass counting. For example, items such as '3**' or '4**' will be removed from the database. As mentioned in the work, *flexible association rules* (i.e., an association rule containing items from different abstraction level) can also be generated from this algorithm by relaxing the level-by-level restriction. For example, instead of only finding large itemsets like '12*','23*', itemsets in form of '1*1','12*' can also be found.

4.3 Background on Rule Mining in the Presence of User Belief

In the rule mining process, except the basic constraints we usually used (for examples, the support and confidence thresholds in the case of association rule mining), users may also want to specify some prior beliefs or interests. Hopefully this user specified additional information can increase either the speed of mining or the quality of the resulting rules. One approach to handling this problem is the belief-driven rule discovery. Unlike classical association rule mining, it performs rule discovery in the presence of user beliefs or interests. One difference between this kind of user beliefs and prior domain knowledge is that such kind of user beliefs represent some interests or expectation of the user. The discovered rules may, in fact very often, contradict with the user beliefs. Due to the unexpectedness, these rules are interesting to the user. Some work on the belief-driven discovery of rules has been done [20, 21, 24, 4].

In the work by Silberschatz et al. [30], measures of interestingness of discovered patterns are divided into two types, the objective measure and the subjective measure. While the objective measure depends only on the structure of the discovered patterns, the subjective measure depends also on the interests of users who are extracting the patterns. Therefore, the user beliefs we have been mentioning are actually examples of subjective measures of interestingness.

4.4 Related Work on Rule Mining in the Presence of User Belief

For the belief-driven rule discovery, some related researches are described as follows.

4.4.1 Post-Analysis of Learned Rules

Liu et al. [20] proposed a technique to perform the post analysis of classification rules generated by systems such as C4.5. It aims at identifying the new rules that have changed with respect to the knowledge represented by the existing rules. In his work, existing rules are treated as fuzzy rules, based on fuzzy set theory [40]. Newly generated rules are then matched against the existing fuzzy rules using the fuzzy matching technique.

The fuzzy matching method consists of two steps. In the first step, the user converts each rule in the existing set of rules E to a fuzzy rule, which has the same syntax as the original rule. However, the attribute values of the fuzzy must be described using some fuzzy linguistic variables. In the second step, the system matches each new rule $B_i \in B$ against each fuzzy rule $E_j \in E$ using different matching algorithms to obtain the degree of match for B_i against E. The new

set of rules in B are ranked by their degrees of match with E.

An example similar to those given in the work by Liu et al. [20] is stated as follows. Say the following set of three classification rules are discovered from an accident database.

- IF $Age > 55, Loc = straight$ THEN $Class = slight$
- IF $Age > 70, Loc = bend$ THEN $Class = killed$
- IF $Age > 55, Loc = T - junct$ THEN $Class = slight$

Assume the user believes that an old person crossing at a location with obstructed view is likely to result in a serious accident. The user then specifies the hypothesis as:

IF $Age = OLD, Loc = OBSTRUCT$ THEN $Class = SERIOUS$

Before doing the matching, the user is required to provide the semantic rules associated with the terms 'OLD', 'OBSTRUCT' and 'SERIOUS'. After the matching, the rules are ranked. For example, if confirmation of the hypothesis is desired, the rules that are similar to the hypothesis will be ranked higher, resulting at:

1. IF $Age > 70, Loc = bend$ THEN $Class = killed$
2. IF $Age > 55, Loc = T - junct$ THEN $Class = slight$
3. IF $Age > 55, Loc = straight$ THEN $Class = slight$

On the other hand, if the rules contradictory to the hypothesis are desired, the following ranking will be resulted.

1. IF $Age > 55, Loc = T - junct$ THEN $Class = slight$
2. IF $Age > 70, Loc = bend$ THEN $Class = killed$
3. IF $Age > 55, Loc = straight$ THEN $Class = slight$

4.4.2 Using General Impressions to Analyze Discovered Classification Rules

Another research by Liu et al. [21] performs analysis on discovered classification rules using general impressions. A language is defined for the specification of general impressions by the users. By allowing the 0 between the newly discovered rules and the user-specified general impressions, the conforming rules or the unexpected rules can be identified.

Two type of general impressions are allowed. The first type (or called T1 GI) is in the form of $a_1 ID_1, ..., a_w ID_w \Rightarrow C_j$ where a_i are the attributes and $ID \in \{<, >, <<, |, [aset]\}$. The meanings of T1s can be illustrated as follows.

- $a <\Rightarrow C_j$: A smaller value of a will result in a higher likelihood of being in class C_j.
- $a >\Rightarrow C_j$: A larger value of a will result in a higher likelihood of being in class C_j.
- $a <<\Rightarrow C_j$: If the value of a is within some range, it will result in a higher likelihood of being in class C_j.
- $a| \Rightarrow C_{sub}$: There exist some relationships between attribute a and some subset C_{sub} of all possible classes.
- $a[S] \Rightarrow C_j$: If the value of a is an element in set S, it will result in a higher likelihood of being in class C_j.

The other type of general impressions is T2 GI in the form of

$$a_1 ID_1, ..., a_k ID_k \& a_{k+1} ID_{k+1}, ..., a_w ID_w \Rightarrow C_j.$$

The first part $a_1 ID_1, ..., a_k ID_k$ is the *core* which must be non-empty. The second part, $a_{k+1} ID_{k+1}, ..., a_w ID_w$, is the *supplement*. If the *supplement* is not empty, it means the user believes that the *core* and any subset of the *supplement* may lead to C_j. If the *supplement* is empty, the user believes that the *core* should lead to C_j.

Algorithms were introduced by Liu et al. [21] to match the classification rules to T1 and T2 GI. Conforming rules, unexpected conclusion rules and unexpected condition rules can then be found.

4.4.3 A Belief-Driven Method for Discovering Unexpected Patterns

Padmanabhan et al. have also developed a belief-driven algorithm for discovering unexpected rules [24]. It is done by discovering rules that have consequents contradicting to the consequent of a user-specified belief.

Given a rule $A \Rightarrow B$ and a belief $X \Rightarrow Y$, the rule is unexpected to the belief if the following conditions hold:

- B *AND* Y logically contradict each other.
- A *AND* X holds on a large subset of tuples in the database.
- $A, X \Rightarrow B$ holds.

An example given in the work by Padmanabhan is stated in the following [24]. If a belief: *Professionals tend to shop more on weekends than on weekdays*

(presumably because they are busier during the week). It turns out that the belief itself holds with high confidence. Then the rules such as the following two will be discovered.

- *In December, professionals tend to shop more on weekdays than on weekends.*
- *Professionals in large households tend to shop more on weekdays than on weekends.*

The reasons for these may be the holiday season in December makes professionals shop more often on weekdays and large households may have shopping demands more often than small households.

4.4.4 Constraint-Based Rule Mining

In the recent work of Bayardo et al., a constraint-based mining algorithm is introduced [4]. This set-enumeration search algorithm restricts the amount of rules discovered from dense databases by introducing the consequent constraint and the minimum improvement constraint. The consequent constraint allows the user to specify the consequent of the rules to be discovered, while the minimum improvement constraint prunes any rule with small predictive power improvement over its sub-rules.

The consequent constraint introduced is implemented by allowing the user to specify a single-item consequent that all the discovered rules must have.

The objective of the improvement constraint is to prevent the discovering of many insignificant rules. For example, let the rule $X \Rightarrow Y$ to have a confidence of 90%. Then, many rules in the form of $X\&I_1\&_2\&I_n \Rightarrow Y$ may exist with confidence around 90%, say 90.1%. However, it is obvious that such rules are not de-

sired by the user. To solve this problem, the *minimumimprovementconstraint* was introduced, which requires each discovered rule to have an *improvement* higher than a predetermined threshold. The *improvement* of a rule $A \Rightarrow C$ is defined as $imp(A \Rightarrow C) = min(\forall A' \subset A, conf(A \Rightarrow C) - conf(A' \Rightarrow C))$ where $conf(A \Rightarrow C)$ denotes the confidence value of rule $A \Rightarrow C$. The *improvement* constraint guarantees that the addition of each item to the condition can contribute a certain improvement to the confidence value of the rule.

Instead of using the traditional association rule mining algorithm, an algorithm based on the set-enumeration search was introduced. It stored the possible combinations of the rule conditions in a set-enumeration tree. The upper-bound for the confidence, improvement or support of any rule in certain groups of rules are calculated. Then, if a group of rules failed to satisfy any of the three constraints, the whole group of rules can be pruned from the set-enumeration tree. The desired rules can be discovered after pruning of the tree and some postprocessing.

4.5 Limitations of Existing Approaches

In Srikant's work [32], no prior knowledge or belief is allowed to be supplied by the user. In our work, we intend to incorporate the user belief in our rule mining. Also, the definition that we introduced for rule interestingness are more general than those introduced by Srikant.

Among the above mentioned existing belief-driven algorithms, most of them only deal with rules having single-item consequents [20, 21, 24, 4], and concentrate on classification rules only [20, 21, 4]. Also, while it is reasonable to treat some previously accepted rules as part of our belief, in most of the existing work, unexpectedness of discovered rules are usually measured only with respect to the

user belief but not previously discovered rules [20, 21, 24]. In particular, [4] only allows the user to specify a consequent constraint, instead of a complete belief.

Note that our approach is based on probability while the fuzzy approach is based on fuzzy set theory. Therefore, our approaches need not have the user to specify the membership functions for the values of the attributes. Moreover, our approarch can handle rules with multiple-item conditions and consequents while most of the existing work cannot.

Chapter 5

Multiple-Level Association Rules Mining in the Presence of User Belief

We propose a new algorithm for mining multiple-level association rules from databases in the presence of user belief. It is quite often that a large number of rules are discovered in the mining of association rules. Unfortunately, many rules may be useless or not interesting to the user. To concentrate on what the user intends to discover, one can make use of belief-driven mining. The idea of belief-driven mining is to allow the user to specify some beliefs or interests which are then used in the rule mining process, in order to limit the discovered rules to only those desirable to the user. Different methods have been developed for the users to specify their beliefs or interests. For example, in the work by Bayardo et al., only the rules having a consequent the same as that specified by the user are discovered [4]. In this case, users can find rules conforming to their beliefs. On the other hand, some algorithms target at discovering rules that are unexpected with respect to the user belief. For example, the algorithm developed by Padmanabhan et al. discovered rules that have consequents contradicting to

that of the user belief [24].

5.1 User Belief Under Taxonomy

Our approach allows users to specify a belief in the form of an association rule $X \Rightarrow Y$ where X and Y are non-0 itemsets. Also, there exists no items $p, q \in X \cup Y$ such that p is an ancestor of q. An example of a user belief is: $WhiteBread, Cheese \Rightarrow Milk$.

Due to the presence of taxonomy, these are two kinds of beliefs which differ in their interpretation. The first kind is called an exact belief and the second kind is called a vague belief. In this chapter, we focus our discussion on exact beliefs.

An exact belief refers to the situation where the user is certain about the specified user belief. The belief has a similar form as a multiple level association rule. Suppose there is a manager of a supermarket selling food items with hierarchy as shown in Figure 4.1. He has the belief that "people buying bread usually also buy milk", or in the rule form $Bread \Rightarrow Milk$. This belief comes from his experience and he is rather sure about it. Now what the manager wants to extract from the supermarket transaction database are some patterns that are out of his expectation, that is, unexpected with respective to his belief. Therefore, a newly discovered rule $WheatBread \Rightarrow Milk$ with a *low* confidence value will be valuable to the manager, since the manager expects all bread-buyers usually also buy milk. Now in the mind of the manager, his belief changes to "people buying bread other than wheat bread usually also buy milk". Now, say if we further discover a new rule $WheatBread \Rightarrow SkimMilk$ with a *high* confidence value, then it again is unexpected to the manager. The manager's belief will now change to "people buying bread other than wheat bread usually also buy milk,

while people buying wheat bread usually also buy skim milk".

The example given above is a simplified one. Instead of comparing the confidence of the rules directly, we consider both the exact confidence and the expected confidence of the rules. To illustrate the exact beliefs more clearly, we will present a detailed example. We say a rule is specific if its items come from the lower levels of the hierarchy. A rule is general if it has items from the higher levels of the hierarchy. Let us consider a taxonomy as shown in Figure 5.1.

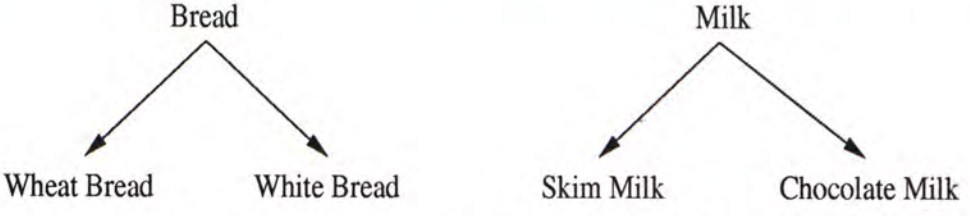

Figure 5.1: Another Hierarchy of Food Items

Suppose a user specifies a belief: $Bread \Rightarrow SkimMilk$ which has a confidence of 25%. Two rules with enough support and confidence values are discovered from the database as shown in the following (with their respective confidence values).

1. $Bread \Rightarrow Milk(50\%)$
2. $WhiteBread \Rightarrow SkimMilk(50\%)$

Now, Rule 1 is a more general rule than the user belief since it contains item $Milk$ which is at a higher level than item $SkimMilk$ contained in the belief. Rule 2 is a more specific rule than the user belief since it contains item $WhiteBread$ which is at a lower level than item $Bread$ in the belief. Their relationships are shown in Figure 5.2 with the more general rules in higher levels.

Now we see how the exact belief interpretation works. Assume after some calculation, we found that the expected confidence of the user belief with respect to Rule 1 is 25%. It means that we would expect the confidence of the user belief

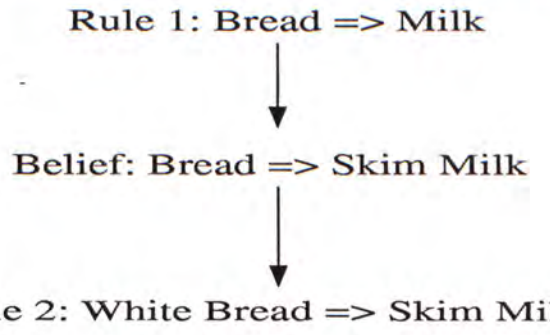

Figure 5.2: The General-Specific Relationship Among the Rules

to be 25% given Rule 1 with a confidence of 50%. Since the actual confidence of the user belief is also 25% which is the same as the expected value, we can say that the confidence of the user belief can actually be induced from that of Rule 1. Rule 1 can be used to generalized the user belief, and the user belief is only a special case of Rule 1. Hence Rule 1 is interesting to the user.

Now, we look at Rule 2. Assume also after some calculation, we found that the expected confidence of Rule 2 with respect to the user belief is 25%. It means that we would expect the confidence of Rule 2 to be 25% given the user belief with a confidence of 25%. However, the actual of confidence value of Rule 2 is 50% with is greatly different from the expected value of 25%. Therefore, we can say that given such a user belief, the user would not be able to get a near guess at the confidence of Rule 2. Hence, Rule 2 is unexpected and is interesting to the user.

5.2 Formal Definition of Rule Interestingness

We will describe the notion of interestingness of rules given exact belief. Before introducing our new formal definition of interestingness of rules, we first introduce the definition of some basic concepts.

Definition An itemset X is a descendant of another itemset $\hat{X}$ (and $\hat{X}$ is an ancestor of X), if $\hat{X}$ can be obtained by dropping some items in X and/or replacing some items in X by their ancestor items, i.e., $X = \{x_1, ..., x_i, x_{i+1}, ..., x_m\}$ and $\hat{X} = \{\hat{x}_1, ..., \hat{x}_u, x_{u+1}, ..., x_i\}$.

For example, itemset X is a descendant of itemset $\hat{X}$ where $X = \{a, b, C, D, e\}$, $\hat{X} = \{A, b, D, E\}$ and item A is an ancestor of item a, B is an ancestor of b and so on.

Definition A rule $\hat{R}$ is an ancestor of another rule R (and R is a descendant of $\hat{R}$), if $R : X \Rightarrow Y$ and; $\hat{R} : \hat{X} \Rightarrow \hat{Y}$ or $\hat{R} : \hat{X} \Rightarrow Y$ or $\hat{R} : X \Rightarrow \hat{Y}$.

From the definition, the relatively more general rules will be treated as ancestor rules and the relatively more specific rules will be treated as descendants rules. For example, rule $\hat{R}$ is an ancestor of itemset R where $R : a, b, C \Rightarrow D, e$ and $\hat{R} : A, b \Rightarrow D, E$, and item A is an ancestor of item a, B is an ancestor of b and so on.

Definition A rule $\hat{R}$ is an immediate ancestor of another rule R (and R is an immediate descent of R) in a set S of rules if there exists no rule R' in S such that $\hat{R}$ is an ancestor of R' and R' is an ancestor of R.

Definition Given $R : X \Rightarrow Y$ where $X = \{x_1, ..., x_i, x_{i+1}, ..., x_m\}$ and $Y = \{y_1, ..., y_j, y_{j+1}, ..., y_n\}$. Also given $\hat{R} : \hat{X} \Rightarrow \hat{Y}$ where $\hat{X} = \{\hat{x}_1, ..., \hat{x}_u, x_{u+1}, ..., x_i\}$ and $\hat{Y} = \{\hat{y}_1, ..., \hat{y}_v, y_{v+1}, ..., y_j\}$. The expected support of R given $\hat{R}$ is defined as

$$E_{\hat{R}}[P(R)] = P(\hat{R}) \times \frac{P(x_1)}{P(\hat{x}_1)} \times \cdots \times \frac{P(x_u)}{P(\hat{x}_u)} \times P(x_{i+1}) \times \cdots \times P(x_m) \times \frac{P(y_1)}{P(\hat{y}_1)} \times \cdots$$

$$\times \frac{P(y_v)}{P(\hat{y}_v)} \times P(y_{j+1}) \times \cdots \times P(y_n)$$

where $P(X)$ denotes the support of itemset X in the database, i.e., the probability that an itemset X appears in a transaction in the database.

Definition The expected confidence of a rule R given $\hat{R}$ is defined as

$$E_{\hat{R}}[P(Y|X)] = P(\hat{Y}|\hat{X}) \times \frac{P(y_1)}{P(\hat{y}_1)} \times \cdots \times \frac{P(y_v)}{P(\hat{y}_v)} \times P(y_{j+1}) \times \cdots \times P(y_n)$$

where $P(Y|X)$ denotes the confidence of rule $X \Rightarrow Y$ in the database, i.e., the percentage of transactions in the database that contain itemset X also contain itemset Y.

Definition The improvement of a rule $R : X \Rightarrow Y$ is defined as

$$imp(R) = min(\forall X' \in X, conf(X \Rightarrow Y) - conf(X' \Rightarrow Y))$$

which was introduced by Bayardo et al. [4]. It measures the improvement of the predictive power of a rule with respect to its proper sub-rules by comparing their confidence values.

If the newly discovered rules are more specific than the belief, they will be interesting if their confidence or support values cannot be predicted from the user belief or some previously discovered rules. It is because the existence of these newly discovered rules is out of the user's previous knowledge. On the other hand, if the discovered rules are more general than the user belief, they will be interesting if the user belief or some previously discovered rules can be predicted from these newly discovered rules. These newly discovered rules can generalize the belief of the user and are therefore useful. Note that the newly discovered rules are compared not only to the user belief but also among other previously discovered rules. In our mining algorithm, all rules we encounter will be composed of all ancestor items and descendant items of the items that exist in the user belief.

Now, we introduce the definition of interestingness of the discovered rules. Assume that we have a belief $B : P \Rightarrow Q$. For the exact user belief interpretation, a discovered rule $R : X \Rightarrow Y$ is interesting if either of the following four sets of conditions is satisfied.

1. Among the set of all ancestor rules of B, R is an immediate ancestor of B

   and $|P(B) - E_R[P(B)]| \leq \theta_{sup}$;

   and $|P(T|Q) - E_R[P(T|Q)]| \leq \theta_{conf}$;

   and $imp(R) \geq \theta_{imp}$.

   A rule is an immediate ancestor of the user belief and can be used to predict the user belief is an interesting rule because it is a more general pattern than the user belief and therefore is useful.

2. Among the set of all ancestor rules of R', R is an immediate ancestor of R', where R' is interesting

   and R' is an ancestor of B;

   and $|P(R') - E_R[P(R')]| \leq \theta_{sup}$;

   and $|P(Y|X) - E_R[P(Y'|X')]| \leq \theta_{conf}$;

   and $imp(R) \geq \theta_{imp}$.

   A newly discovered rule is interesting if it is an immediate ancestor of a previously discovered general rule and can be used to predict that previously discovered rule. This new rule is useful to the user, in the sense that this it can generalize the previously discovered general rules.

3. Among the set of interesting descendant rules of B, R is an immediate descendant of B;

   and $imp(R) \geq \theta_{imp}$;

   and $|P(R) - E_B[P(R)]| \geq \phi_{sup}$

   or $|P(Y|X) - E_B[P(Y|X)]| \geq \phi_{conf}$.

   It means the newly discovered rule has either unexpected support confidence or unexpected confidence value with respect to the user belief.

4. Among the set of interesting descendant rules of R', R is an immediate descendant of R', where R' is interesting;

   and R' is a descendant of B;

   and $imp(R) \geq \theta_{imp}$;

   and $|P(R) - E_{R'}[P(R)]| \geq \phi_{sup}$

   or $|P(Y|X) - E_{R'}[P(Y|X)]| \leq \phi_{conf}$.

   It means the newly discovered rule has either unexpected support confidence or unexpected confidence value with respect to some previously discovered interesting rules.

5.3 The MARUB_E Mining Algorithm

The proposed algorithm for mining multiple-level association rules in the presence of exact user belief is as follows.

In the algorithm, we first call the function *expandDatabase*1() to expand all the items in the database with their ancestors and to remove those items that are not considered with respect to the user belief. We then find all the frequent itemsets from the resulting database. From the discovered frequent itemsets, we can generate rules that satisfy both the minimum support and minimum confidence criteria. All ancestor/descendant relationships among the rules are then discovered, from which we can construct the immediate ancestor/immediate descendant relationships among the rules. These immediate ancestor/immediate descendant relationships are used in the mining of interesting rules. Now, if the user has specified an exact belief, the algorithm will start finding interesting ancestor rules and interesting descendant rules from the user belief. Ignore all rules that are not descendants of user belief The interesting rules that are more general than the belief are discovered by the function $findIntAnc()$. The

Algorithm 5.1 MARUB_E

*expandDatabase*1()
Find set L of all large itemsets
Generate set R of rules from L with minimum confidence
Find ancestor/descendant relationship among all rules in R
Find immediate ancestor/descendant relationship among all rules in R
$findIntAnc(User_Belief)$
Prune rules in R that are not descendants of User_Belief
while there exist unmarked rules in R
    for each rule I
        if I is unmarked
        then if all ancestors of I are marked
            then if I is interesting to any
                immediate ancestor among
                the interesting ancestors
                then Output I
                    Mark I as interesting
                else Mark I as uninteresting
    end for
end while

Figure 5.3: The Framework of the MARUB_E Algorithm

interesting rules that are more specific than the user belief are discovered in the subsequent steps in the algorithm. Each rule is checked whether it can be marked as interesting or uninteresting when all its ancestor rules are already marked as either interesting or uninteresting. The algorithm ends when no more unmarked rules exist. Note that we only need to consider the rules that are descendants of the user belief for mining interesting rules that are more specific than the user belief.

In order to consider only those relevant items, we retain only the items in the transactions that appear in set H, that contains the items in the belief and all ancestors and descendants of the items in the belief, which are required for the generation of the desired rules.

For example, if the user specifies a belief $Milk \Rightarrow Cheese$ with respect to the hierarchy in Figure 4.1, then we will retain in the transactions only the items

that appear in the tree of *DairyProducts* except *Butter*. It is because *Butter* is neither an ancestor nor a descendant item of the items in the belief. We will remove all items in the tree of *Bread*. This processing of items is done by the function *expandDatabase*1().

Algorithm 5.2 FUNCTION expandDatabase1()
Let B be the set of items that appear in the user belief
Find set H of all the ancestor items of the items in B
Add all items in B to set H
Add all descendant items of the items in B to set H
for each transaction T
  for each item $X \in T$
    if $X \in H$
    then Expand X with all its ancestors in H and add to the transaction
  end for
end for

Figure 5.4: Function *expandDatabase*1()

The function $findIntAnc()$ is shown as follows.

Algorithm 5.3 FUNCTION findIntAnc(X)
for each immediate ancestor A of X
  if A is interesting
  then output A
    $findIntAnc(A)$
end for

Figure 5.5: Function $findIntAnc(X)$

After the function $findIntAnc()$ found an interesting immediate ancestor rule, it will output that rule and then continue to search up the immediate ancestor rule hierarchy to look for other interesting immediate ancestors.

In the algorithm, the finding of immediate ancestor and descendant relationship among all rules are essential for the rule mining process to be carried out. The most difficult part is in the mining of interesting descendant rules that we have to ensure that the new rule is interesting to its "immediate ancestor among interesting ancestors".

Chapter 6

Experiments on MARUB_E

6.1 Preliminary Experiments

A small scale synthetic database is used in this experiment in order to check the correctness of the rules discovered by the MARUB_E algorithm. We have actually expanded all possible combinations of rules that can be formed from this small scale synthetic database and checked the correctness and completeness of our MARUB_E algorithm and the discovered rules.

The transaction data are generated according to the hierarchy as shown in the Figure 6.1. Note that only items 1 to 6 exist in the original database.

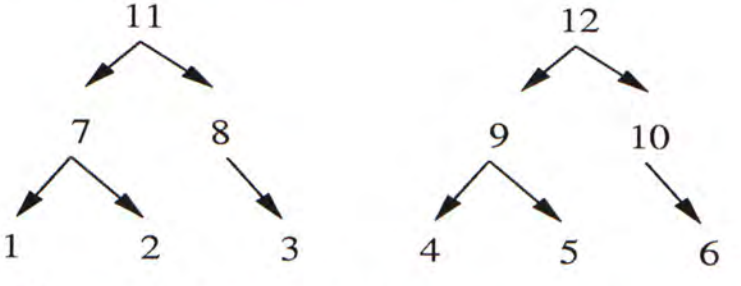

Figure 6.1: A Hierarchy of Items in the Database

The distributions of the items in the transactions in the expanded database are shown in Figure 6.2. The figure is a two dimensional view of the database

where the numbers at the top indicate the items and the numbers on the right show the transaction numbers. For example, it is indicated in the figure that the first two thousand transactions contain items 3, 8 and 11, and in transactions 5501 to 8000, items 1, 4, 7, 9, 11 and 12 are contained.

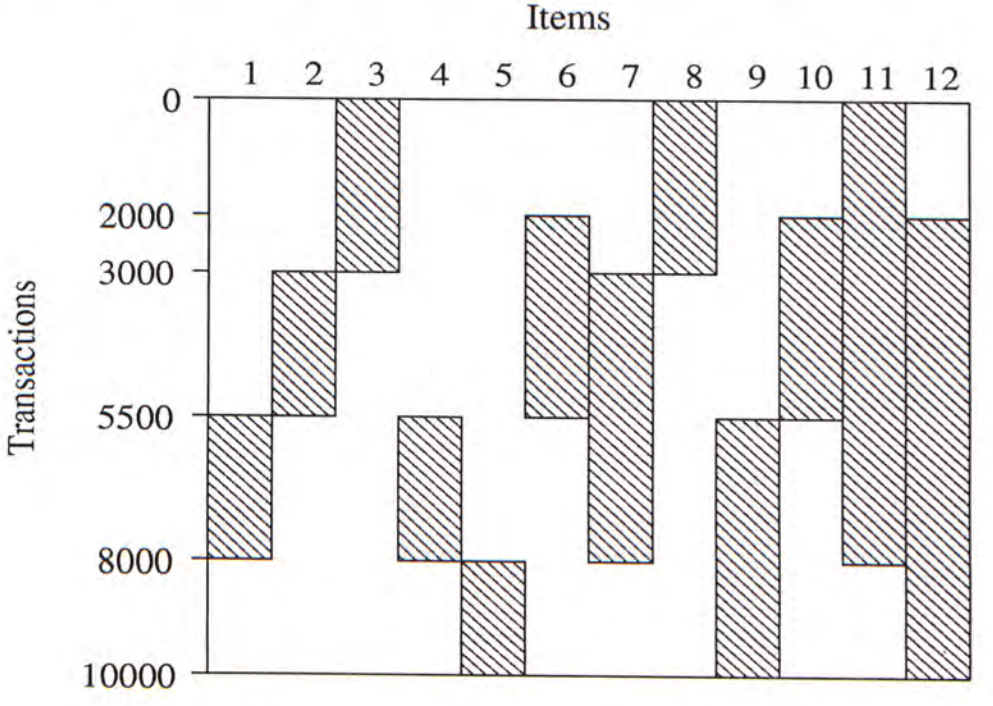

Figure 6.2: Distribution of Items in the Database

Two sets of experiment are done on this synthetic data with different sets of parameters. The user belief

$$item7 \Rightarrow item9$$

which has a confidence 0.5 and a support 0.25 in the database, is used for both sets of experiment.

- $Rule2 : item7 \Rightarrow item12$
- $Rule7 : item7 \Rightarrow item4$
- $Rule13 : item1 \Rightarrow item9$
- $Rule14 : item1 \Rightarrow item4$

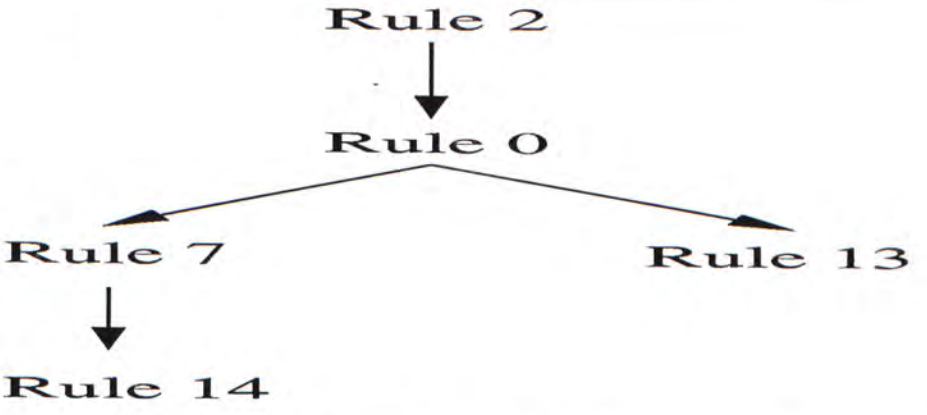

Figure 6.3: The Ancestor/Descendant Relationships among Discovered Rules (Set 1)

Rule 2 is discovered as an interesting rule that can be used to generalize the user belief. The user belief has both its real support and real confidence similar to the expected support and expected confidence with respect to Rule 2. Rules 7 and 13 are also interesting because their real support and real confidence deviate a lot from their expected support and expected confidence given the user belief. Similarly, Rule 14 is also found to be interesting with respect to Rule 7.

Let us look at the discovered rules again in the user's view. The user believes that *item*7 implies *item*9 originally. However, it is discovered in the rule that *item*7 implies *item*12. It means to the user that the belief of *item*7 implies *item*9 is actually only a special case of the fact that *item*7 implies *item*12. Therefore, *item*7 implies *item*12 has a great impact on the user's belief. On the other hand, a more specific rule than the user belief is discovered, which states that *item*7 implies *item*4 with very high confidence and support than expected. It again affects the user's belief by providing an exception to the user belief. The other two rules are also interesting to the user because they provide exceptions to the user. Note that without applying the interesting measures to the approach results in more than twenty discovered rules.

Table 6.1 shows the source of interestingness of each rule. The first column

is the number of the rule under consideration, the second and third columns tell us the source of interestingness for the rule. A rule can either be an ancestor of (indicated as 'anc of') another interesting rule (indicated in 'w. r. t.'), or a descendant of (indicated as 'des of') another interesting rule (indicated in 'w. r. t.'), or is itself the most general rule (indicated as 'topmost'). The last five columns show the expected support, expected confidence, improvement value, real support and real confidence of each rule.

Rule		w. r. t.	Exp Sup	Exp Conf	Imp	Re Sup	Re Conf
2	anc of	0	0.2813	0.5625	1.0000	0.2500	0.5000
7	des of	0	0.1389	0.2778	1.0000	0.2500	0.5000
13	des of	0	0.1250	0.5000	1.0000	0.2500	1.0000
14	des of	7	0.1250	0.5000	1.0000	0.2500	1.0000

Table 6.1: Output Rules for Exact Belief (Set 1).

For the second set of experiment, we fix the parameters as $\theta_s = 0.2$, $\theta_c = 0.2$, $\theta_i = 0.1$, $\phi_s = 0.1$ and $\phi_c = 0.1$. Using the MARUB_E algorithm, 4 rules are discovered which are the same as those discovered in the first set of experiment.

In this experiment, we increase θ_s and θ_c in order to allow more general rules to be discovered. This adjustment of parameters Increasing of parameters θ_s and θ_c should be done when the user is interested in the rules that are more general than the user belief. On the other hand, ϕ_s and ϕ_c are decreased that we expect more specific rules can be found also. Decreasing of parameters ϕ_s and ϕ_c should be done when the user is interested in the rules that are more specific than the user belief. After running MARUB_E over the data, it turns out that the amount of specific rules and general rules found remains the same as those in the first set of experiment.

We have actually expanded all possible combinations of rules that can be formed from this small scale synthetic database and checked the correctness and

completeness of our MARUB_E algorithm and the discovered rules.

6.2 Experiments on Synthetic Data

We have conducted experiments on a large scale synthetic dataset. The synthetic data used in all the experiments are generated following the techniques in [8]. In our experiments, the number of transactions is 10,000; the number of items is 200; the mean size of transactions is 10; the mean size of the maximal potentially large itemsets is 4; the number of potentially large itemsets is 1000. Apart from generating the raw transactions, we need to generate the item hierarchy. The hierarchy of the items are generated using an algorithm described as follows. Nodes in the hierarchy are generated in a bottom-up level-by-level manner. First, the items that appear in the transactions are treated as the leaf nodes in the trees of the hierarchy. Then, nodes in the second lowest level of the trees are created one by one, with groups of leaf nodes assigned as their descendents. The number of leaf nodes to be grouped under a node in the second lowest level is determined by a normal distribution with predetermined mean and variance. The nodes in the second lowest level are then grouped similarly to form the descendants of nodes in the higher level. The number of trees formed in the hierarchy decreases as the process proceeds. The generation process will stop when the number of trees formed in the hierarchy is reduced to a predetermined threshold. By altering the number of trees in the hierarchy and the values of mean and variance for the fan-out of nodes, hierarchy of different desired types can be generated. In our case, the number of leaf nodes are 200 and the total number of nodes in the hierarchy are 261. The mean and variance for the fan-out are both 5. There are totally 13 trees of three level high in the hierarchy.

Two sets of experiment are done on this synthetic data with different sets of

parameters. The user belief

$$item203, item249 \Rightarrow item250$$

which has a confidence 0.5472 and a support 0.0893 in the database, is used for both sets of experiment.

For the first set of experiment, we fix the parameters as $\theta_s = 0.08$, $\theta_c = 0.05$, $\theta_i = 0.05$, $\phi_s = 0.08$ and $\phi_c = 0.05$. Using the MARUB_E algorithm, three rules are discovered as shown below. Their ancestor/descendant relationships are shown in Figure 6.4.

- $Rule112 : item11, item205 \Rightarrow item250$
- $Rule144 : item24, item203 \Rightarrow item31$
- $Rule172 : item203, item249 \Rightarrow item31$

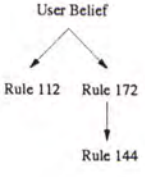

Figure 6.4: The Ancestor/Descendant Relationships among Discovered Rules (Set 1)

According to the user belief, the user expects that items 203, 249 imply item 250. In this experiment, no rules more general than the user belief are discovered. Two rules are discovered to be unexpected with respect to the user belief. Table 6.2 shows to which other rule is a rule unexpected to. The first column shows the rule number. The second and third columns show whether this rule is an ancestor or a descendant of another interesting rule. For example "des of 0" means that the rule is a descendant of interesting Rule 0 ("D" means descendant and "A" means ancestor). Note that we use Rule 0 to denote the User Belief in

Rule		w. r. t.	Exp Sup	Exp Conf	Imp	Re Sup	Re Conf
112	des of	0	0.0302	0.5472	0.0632	0.0453	0.7626
144	des of	172	0.0232	0.3039	0.0691	0.0300	0.3901
172	des of	0	0.0343	0.2104	0.0713	0.0496	0.3039

Table 6.2: Output Rules for Exact Belief (Set 1).

the tables. The following columns show the values of expected support, expected confidence, improvement, actual confidence and actual support of the rule.

For the second set of experiment, we fix the parameters as $\theta_s = 0.1$, $\theta_c = 0.1$, $\theta_i = 0.05$, $\phi_s = 0.1$ and $\phi_c = 0.1$. Using the MARUB_E algorithm, two rule are discovered as shown below. Their ancestor/descendant relationships are shown in Figure 6.5.

- $Rule112 : item11, item205 \Rightarrow item250$
- $Rule171 : item203, item205 \Rightarrow item31$

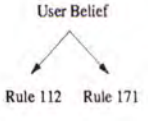

Figure 6.5: The Ancestor/Descendant Relationships among Discovered Rules (Set 2)

If the user is more interested in the general rules rather than the specific rules, the values of θ_s and θ_c should be raised and the values of ϕ_s and ϕ_c should be also be raised, which is what we do in this experiment. Since θ_s and θ_c are increased, we expect the number of general rules discovered to increase. ϕ_s and ϕ_c are increased, so we expect the number of specific rules to decrease. In this experiment, it turns out that the number of specific rules decreases but general rules are still not discovered. Note that Rule 172 is interesting in the first set of experiment. However, when ϕ_c is raised to 0.1, Rule 172 is no longer interesting.

Instead, its descendant Rule 171 becomes interesting. Also, since Rule 172 is not interesting, Rule 144 can no longer be discovered to be interesting with respect to it. Table 6.3 shows the source of interestingness for each rule.

Rule		w. r. t.	Exp Sup	Exp Conf	Imp	Re Sup	Re Conf
112	des of	0	0.0302	0.5472	0.0632	0.0453	0.7626
171	des of	0	0.0184	0.2104	0.0785	0.0339	0.3729

Table 6.3: Output Rules for Exact Belief (Set 2).

6.3 Experiments on Real Data

The real dataset is originally a set of text data obtained by analyzing 91 news articles from the newsgroup clari.world.africa. Only those articles with not less than 200 words were chosen. The dataset was used before in [31] for the mining of dependence rules. In order to study the multiple-level association rule mining under user beliefs, we limit the items in the dataset to those that are specially chosen. We generated a hierarchy consisted of two trees according to the selected items as shown in Figure 6.6.

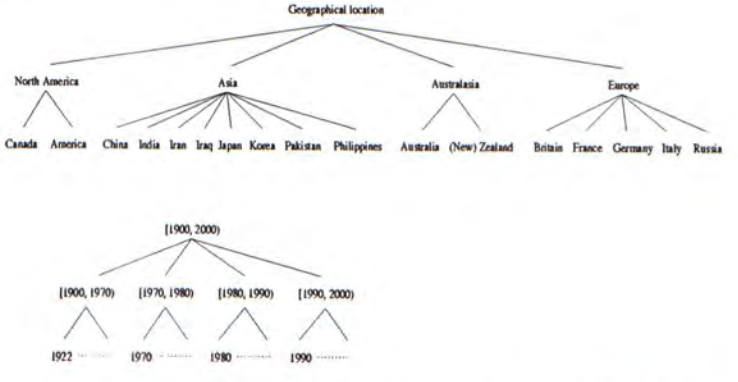

Figure 6.6: A Hierarchy of Items in the Real Data

The first tree consists of items of country names grouped in continents. The second tree consists of items that are numeric numbers in the form of $19XX$

which probably indicates years. Two sets of experiment are done on this synthetic data with different sets of parameters. The user belief

$$Asia \Rightarrow [1990, 2000)$$

which has a confidence 0.4611 and a support 0.172 in the database, is used for both sets of experiment.

For the first set of experiment, we fix the parameters as $\theta_s = 0.2$, $\theta_c = 0.2$, $\theta_i = 0.05$, $\phi_s = 0.2$ and $\phi_c = 0.2$. Using the MARUB_E algorithm, eight rules are discovered as shown below. Their ancestor/descendant relationships are shown in Figure 6.7.

- $Rule1 : France, Iraq \Rightarrow 1990$
- $Rule2 : France, Asia \Rightarrow 1990$
- $Rule4 : Iraq \Rightarrow 1990$
- $Rule7 : Canada, Asia \Rightarrow [1990, 2000)$
- $Rule37 : France, Iraq \Rightarrow [1990, 2000)$
- $Rule78 : GeographicalLocation \Rightarrow [1990, 2000)$
- $Rule84 : Asia \Rightarrow [1900, 2000)$
- $Rule88 : GeographicalLocation \Rightarrow [1900, 2000)$

According to the user belief, we can say that the user expects when Asia is mentioned in an article, the years 1990 to 2000 are usually also mentioned. In this case, Rules 78, 88 and 84 can be used to generalize the user belief. However, since this hierarchy consists of only two trees, the general Rule 88 is not very useful. Rules 1, 2, 4, 7 and 37 have real confidences much higher than their

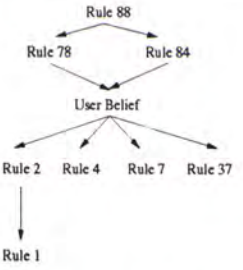

Figure 6.7: The Ancestor/Descendant Relationships among Discovered Rules (Set 1)

expected supports, which means that they are more specific with respect to their ancestor. For example, Rule 7 has a much higher real confidence when compared to its expected confidence with respect to the user belief. It indicates that when *Canada* are mentioned with *Asia*, there will be a higher chance that [1990, 2000) is also mentioned. In other words, although the period of 1990 to 2000 is usually mentioned with Asia, the period of 1990 to 2000 is mentioned especially frequent when both Canada and Asia are mentioned. Another example is that from Rule 37, we can see that the period 1990 to 2000 is usually mentioned with France and Iraq. From a more specific Rule 1, we can see that it is actually 1990 that is usually mentioned with France and Iraq.

Table 6.4 shows the source of interestingness for the discovered rules.

Rule		w. r. t.	Exp Sup	Exp Conf	Imp	Re Sup	Re Conf
1	des of	2	0.0048	0.3250	0.2038	0.0130	0.5493
2	des of	0	0.0026	0.0794	0.1411	0.0130	0.3250
4	des of	0	0.0109	0.0794	1	0.0473	0.3455
7	des of	0	0.0060	0.4611	0.0875	0.0110	0.6875
37	des of	0	0.0055	0.4611	0.1026	0.0157	0.6620
78	anc of	0	0.1809	0.4851	1	0.1720	0.4611
84	anc of	0	0.1799	0.4822	1	0.1720	0.4611
88	anc of	84	0.2202	0.5902	1	0.2183	0.5853

Table 6.4: Output Rules for Exact Belief (Set 1).

For the second set of experiment, we fix the parameters as $\theta_s = 0.3$, $\theta_c = 0.3$,

$\theta_i = 0.05$, $\phi_s = 0.3$ and $\phi_c = 0.3$. Using the MARUB_E algorithm, four rules are discovered as shown below. Their ancestor/descendant relationships are shown in Figure 6.8.

- $Rule5 : Iraq, Europe \Rightarrow 1990$
- $Rule78 : GeographicalLocation \Rightarrow [1990, 2000)$
- $Rule84 : Asia \Rightarrow [1900, 2000)$
- $Rule88 : GeographicalLocation \Rightarrow [1900, 2000)$

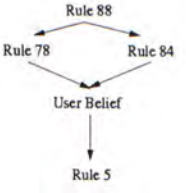

Figure 6.8: The Ancestor/Descendant Relationships among Discovered Rules (Set 2)

Since θ_s and θ_c are increased, we expect the number of general rules discovered to increase. ϕ_s and ϕ_c are increased, so we expect the number of specific rules to decrease. In our case, it turns out that the number of general rules remain unchanged while the number of specific rules decreases.

Note that Rule 4 is interesting with respect to the ancestor user belief for Set 1 experiment. However, in Set 2 experiment, due to the increase in ϕ_c, Rule 4 is no longer interesting with respect to the user belief. Instead, its descendant Rule 5 becomes interesting with respect to the user belief. Table 6.5 shows the source of interestingness for the discovered rules.

Rule		w. r. t.	Exp Sup	Exp Conf	Imp	Re Sup	Re Conf
5	des of	0	0.0026	0.0794	0.0733	0.0163	0.4188
78	anc of	0	0.1809	0.4851	1	0.1720	0.4611
84	anc of	0	0.1799	0.4822	1	0.1720	0.4611
88	anc of	84	0.2202	0.5902	1	0.2183	0.5853

Table 6.5: Output Rules for Exact Belief (Set 2).

Chapter 7

Dealing with Vague Belief of User

7.1 User Belief Under Taxonomy

In this chapter, we look at another kind of interpretation of user belief, called the vague belief. A vague belief has a similar form with the exact belief. It refers to the situation that the user is not certain about the items as specified in the belief. The user has a rough idea about the relationships among the items and wants to know if association rules having similar patterns to the belief can be discovered. Let us consider again the supermarket manager example in a different situation. The manager has the belief that "people buying bread usually also buy milk", or in the rule form $Bread \Rightarrow Milk$. but he is not sure about whether it is true. A supervisor in the supermarket comes to tell the manager that "people buying bread usually also buy skim milk" is a more believable pattern. To the worst, another supervisor in the supermarket tells the manager that, according to his observation, "people buying bread usually also buy dairy products, not only milk". Now, though the manager has an idea about the shopping pattern "people buying bread usually also buy milk", he is not sure about the level of the

items in his belief. Now what the manager wants to extract from the supermarket transaction database are some rules that have similar patterns with his belief.

Suppose from the transaction database, a rule $Bread \Rightarrow Milk$ is discovered with a high confidence value. Therefore the manager now knows that "people buying bread usually also buy milk" is true. However, an additional rule $Bread \Rightarrow SkimMilk$ is also discovered which has a *lower confidence* value than the previous rule. It is a valuable rule to the manager. The manager can now make the conclusion that "people buying bread usually also buy milk other than skim milk".

A detailed example is presented below to illustrate more clearly how the vague beliefs work. Again, we say a rule is specific if its items come from the lower levels of the hierarchy and a rule is general if it has items from the higher levels of the hierarchy. Let us consider the taxonomy as shown in Figure 5.1.

Suppose a user specifies a belief: $Bread \Rightarrow SkimMilk$ which has a confidence of 25%. 6 rules with enough support and confidence values are discovered from the database as shown in the following (with their respective confidence values).

1. $Bread \Rightarrow Milk(50\%)$
2. $WhiteBread \Rightarrow Milk(100\%)$
3. $Bread \Rightarrow SkimMilk(25\%)$
4. $Bread \Rightarrow ChocolateMilk(25\%)$
5. $WhiteBread \Rightarrow SkimMilk(50\%)$
6. $WhiteBread \Rightarrow ChocolateMilk(50\%)$

Here, Rule 1 is the most general rule, Rules 2, 3 and 4 are more specific than

Rule 1, and Rules 5 and 6 are the most specific ones. Their relationships are shown in Figure 7.1 with the more general rules in higher levels.

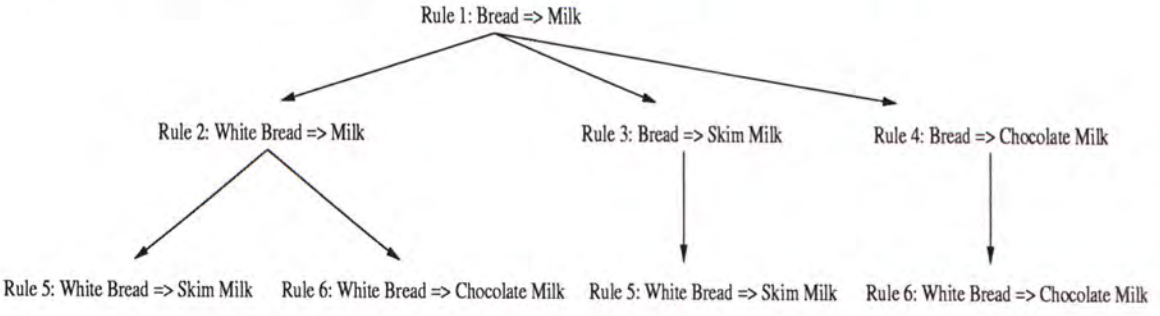

Figure 7.1: The General-Specific Relationship Among the Rules

Now we look at how the vague belief interpretation works. For vague belief, rules are not compared to the user belief but to the previously discovered rules only, starting from the most general ones.

Assume after some calculation, we found that the expected confidence of Rule 2 with respect to Rule 1 is 50%. Rule 2 is regarded as interesting to the user, since its confidence cannot be predicted from Rule 1. Then, we look at the two descendants of Rule 2. Assume expected confidence values of both Rule 5 and Rule 6 with respect to Rule 2 are found to be 50%. As the actual confidence values of both rules are also 50%, their confidence values can be predicted from Rule 2. Therefore, Rule 5 and Rule 6 are not interesting to the user.

Assume we also found that the expected confidence of Rule 3 with respect to Rule 1 is 25%. Then Rule 3 is not interesting since it has an actual confidence of 25%. We then calculate the expected confidence of Rule 5 (which is a descendant of Rule 3) with respect to Rule 1. It is found to be 25%, so Rule 5 is interesting. Rule 5 is unexpected with respect to the previously discovered Rule 1.

We now check Rule 4 with respect to Rule 1. If both its actual confidence and the expected confidence with respect to Rule 1 are 25%, Rule 4 is not interesting. We then calculate the expected confidence of Rule 6 with respect to Rule 1 and

find it equals to 25%. Then, Rule 6 is interesting since its actual confidence is 50% which deviates a lot from the expected value.

7.2 Relationship with Constraint-Based Rule Mining

In the recent work by Bayardo et al., a constraint-based rule mining algorithm is introduced [4], which aims at mining association rules without considering taxonomy. In their algorithm, the rules discovered must have the same consequents as the user-specified single-item consequent. In addition, all those rules must satisfy three constraints, namely, the minimum confidence, the minimum support and the minimum improvement. We can show that our user belief model is a more general approach. If no taxonomy is supplied to our belief model and we include all except one items in the condition part and the remaining item in the consequent part of our belief, then our algorithm for the vague belief will produce the same resulting association rules as that of the constraint-based algorithm by Bayardo et al. if we relax all the thresholds in our algorithm.

7.3 Formal Definition of Rule Interestingness

The definition of basic concepts are restated here for easy reference.

Definition An itemset X is a descendant of another itemset $\hat{X}$ (and $\hat{X}$ is an ancestor of X), if $\hat{X}$ can be obtained by dropping some items in X and/or replacing some items in X by their ancestor items, i.e., $X = \{x_1, ..., x_i, x_{i+1}, ..., x_m\}$ and $\hat{X} = \{\hat{x}_1, ..., \hat{x}_u, x_{u+1}, ..., x_i\}$.

For example, itemset X is a descendant of itemset $\hat{X}$ where $X = \{a, b, C, D, e\}$,

$\hat{X} = \{A, b, D, E\}$ and item A is an ancestor of item a, B is an ancestor of b and so on.

Definition A rule $\hat{R}$ is an ancestor of another rule R (and R is a descendant of $\hat{R}$), if $R : X \Rightarrow Y$ and; $\hat{R} : \hat{X} \Rightarrow \hat{Y}$ or $\hat{R} : \hat{X} \Rightarrow Y$ or $\hat{R} : X \Rightarrow \hat{Y}$.

For example, rule $\hat{R}$ is an ancestor of itemset R where $R : a, b, C \Rightarrow D, e$ and $\hat{R} : A, b \Rightarrow D, E$, and item A is an ancestor of item a, B is an ancestor of b and so on.

Definition A rule $\hat{R}$ is an immediate ancestor of another rule R (and R is an immediate descent of R) in a set S of rules if there exists no rule R' in S such that $\hat{R}$ is an ancestor of R' and R' is an ancestor of R.

Definition Given $R : X \Rightarrow Y$ where $X = \{x_1, ..., x_i, x_{i+1}, ..., x_m\}$ and $Y = \{y_1, ..., y_j, y_{j+1}, ..., y_n\}$. Also given $\hat{R} : \hat{X} \Rightarrow \hat{Y}$ where $\hat{X} = \{\hat{x}_1, ..., \hat{x}_u, x_{u+1}, ..., x_i\}$ and $\hat{Y} = \{\hat{y}_1, ..., \hat{y}_v, y_{v+1}, ..., y_j\}$. The expected support of R given $\hat{R}$ is defined as

$$E_{\hat{R}}[P(R)] = P(\hat{R}) \times \frac{P(x_1)}{P(\hat{x}_1)} \times \cdots \times \frac{P(x_u)}{P(\hat{x}_u)} \times P(x_{i+1}) \times \cdots \times P(x_m) \times \frac{P(y_1)}{P(\hat{y}_1)}$$
$$\times \cdots \times \frac{P(y_v)}{P(\hat{y}_v)} \times P(y_{j+1}) \times \cdots \times P(y_n).$$

Definition The expected confidence of a rule R given $\hat{R}$ is defined as

$$E_{\hat{R}}[P(Y|X)] = P(\hat{Y}|\hat{X}) \times \frac{P(y_1)}{P(\hat{y}_1)} \times \cdots \times \frac{P(y_v)}{P(\hat{y}_v)} \times P(y_{j+1}) \times \cdots \times P(y_n).$$

Definition The improvement of a rule $R : X \Rightarrow Y$ is defined as

$$imp(R) = min(\forall X' \in X, conf(X \Rightarrow Y) - conf(X' \Rightarrow Y))$$

which was introduced by Bayardo et al. [4]. It measures the improvement of the predictive power of a rule with respect to its proper sub-rules by comparing their confidence values.

In the case of vague belief, the belief only provides a rough idea about the relationships about the items. The most general rules conforming to the user belief are first discovered. In addition, rules differing from the previously discovered rules are discovered.

For the vague belief, a discovered rule $R : X \Rightarrow Y$ is interesting if $imp(R) \geq \theta_{imp}$ and either of the following conditions is satisfied, where $\hat{R}$ is an interesting immediate ancestor of R among the interesting ancestor rules of R.

- R has no interesting ancestors.
- $|P(R) - E_{\hat{R}}[P(R)]| \geq \theta_{sup}$.

  It means that rule R has a much greater or much smaller confidence value than its expected confidence value with respect to a given interesting immediate ancestor $\hat{R}$ of R.
- $|P(Y|X) - E_{\hat{R}}[P(Y|X)]| \geq \theta_{conf}$.

  It means that rule R has a much greater or much smaller confidence value than its expected confidence value with respect to a given interesting immediate ancestor $\hat{R}$ of R.

7.4 The MARUB_V Mining Algorithm

The proposed algorithm for mining multiple-level association rules in the presence of vague user-belief is as follows.

In the algorithm, we first call the function *expandDatabase2*() to expand all the items in the database with their ancestors and to remove those items that are not considered with respect to the user belief. Note that the whole trees containing the items in the user belief are considered for the vague belief.

Algorithm 7.1 MARUB_V

```
expandDatabase2()
Find set L of all large itemsets
Generate set R of rules from L with minimum confidence
Find ancestor/descendant relationship among all rules in R
Find immediate ancestor/descendant relationship among all rules in R
Mark all rules in R that do not have ancestors as interesting
while there exist unmarked rules in R
        for each rule I
            if I is unmarked
            then if all ancestors of I are marked
                    then if I is interesting to any
                            immediate ancestor among
                            the interesting ancestors
                            then Output I
                                    Mark I as interesting
                            else Mark I as uninteresting
        end for
end while
```

Figure 7.2: The Framework of the MARUB_V Algorithm

We then find all the frequent itemsets from the resulting database. From the discovered frequent itemsets, we can generate rules that satisfy both the minimum support and minimum confidence criteria. All ancestor/descendant relationships among the rules are then discovered, from which we can construct the immediate ancestor/immediate descendant relationships among the rules. These immediate ancestor/immediate descendant relationships are used in the mining of interesting rules.

Now, if the user has specified an exact belief, the interesting rules that are more specific than the user belief are discovered in the subsequent steps in the algorithm. Each rule is checked whether it can be marked as interesting or uninteresting when all its ancestor rules are already marked as either interesting or uninteresting. The algorithm ends when no more unmarked rules exist. Note that we start with the rules that do not have ancestors. We then discover new interesing rules by comparing them with other interesting rules but not to the

user belief.

In order to consider only those relevant items, we retain only the items in the transactions that appear in set H, that contains the whole trees containing the items in the uer belief, which are required for the generation of the desired rules.

For example, if the user specifies a belief $Milk \Rightarrow Cheese$ with respect to the hierarchy in Figure 4.1, then we will retain in the transactions only the items that appear in the tree of *DairyProducts*. This processing of items is done by the function *expandDatabase*2().

Algorithm 7.2 FUNCTION expandDatabase2()
Let B be the set of items that appear in the user belief
Find set H of all the ancestor items of the items in B
Add all items in B to set H
Add all descendant items of the items in H to set H
for each transaction T
  for each item $X \in T$
    if $X \in H$
    then Expand X with all its ancestors in H and add to the transaction
  end for
end for

Figure 7.3: Function *expandDatabase*2()

Chapter 8

Experiments on MARUB_V

8.1 Preliminary Experiments

The method of the generation of data is the same as that in the experiments for exact belief.

Two experiments are done on this synthetic data with different sets of parameters. The user belief

$$item7 \Rightarrow item9$$

which has a confidence 0.5 and a support 0.25 in the database, is used for both experiments.

For the first set of experiment, we fix the parameters as $\theta_i = 0.1$, $\phi_s = 0.2$ and $\phi_c = 0.2$. Note that the parameters $\theta_s = 0.1$ and $\theta_c = 0.1$ are not used for vague belief according to our definition. Using the MARUB_V algorithm, 12 rules are discovered (including the user belief, i. e., Rule 0) as shown below. Their ancestor/descendant relationships are shown in Figure 8.1. Note that, as in the case of exact belief, a discovered rule can be interesting (i. e., with unexpected confidence or support) with respect to more than one interesting ancestors. Therefore, to prevent confusion, only one possible source of the interestingness is shown for

each rule.

- *Rule*0 : *item*7 $\Rightarrow$ *item*9
- *Rule*2 : *item*7 $\Rightarrow$ *item*12
- *Rule*3 : *item*8 $\Rightarrow$ *item*10
- *Rule*4 : *item*8 $\Rightarrow$ *item*12
- *Rule*5 : *item*11 $\Rightarrow$ *item*9
- *Rule*6 : *item*11 $\Rightarrow$ *item*10
- *Rule*7 : *item*7 $\Rightarrow$ *item*4
- *Rule*8 : *item*11 $\Rightarrow$ *item*4
- *Rule*12 : *item*11 $\Rightarrow$ *item*12
- *Rule*13 : *item*1 $\Rightarrow$ *item*9
- *Rule*14 : *item*1 $\Rightarrow$ *item*4
- *Rule*16 : *item*2 $\Rightarrow$ *item*10

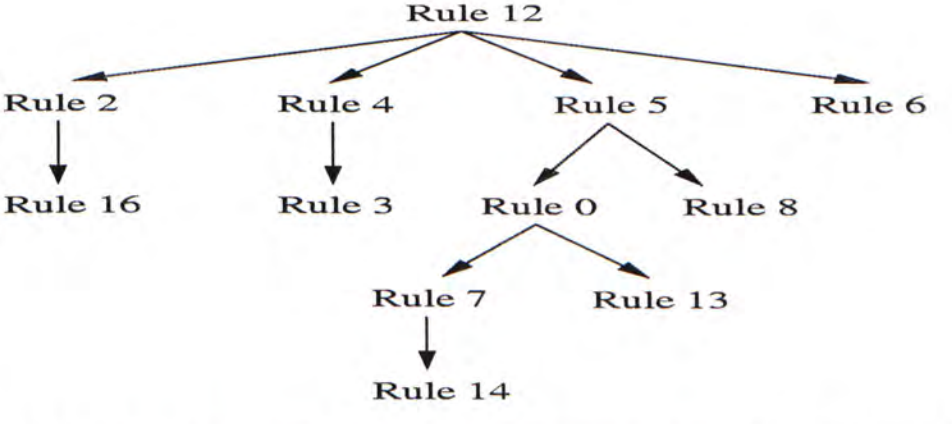

Figure 8.1: The Ancestor/Descendant Relationships among Discovered Rules (Set 1)

The most general rule discovered is Rule 12, which has 4 interesting descendant Rules 2, 4, 5 and 6. Among the 4 rules, 3 have their real confidence larger than the expected confidence, while Rule 4 has its confidence lower than the expected value. Rules 14, 16 and 17 are the most specific rules. Table 8.1 shows the source of interestingness, expected support, expected confidence, improvement, real support and real confidence of each rule.

Rule		w. r. t.	Exp Sup	Exp Conf	Imp	Re Sup	Re Conf
0	des of	5	0.1563	0.3125	1.0000	0.2500	0.5000
2	des of	12	0.3750	0.7500	1.0000	0.5000	1.0000
3	des of	4	0.0438	0.1458	1.0000	0.1000	0.3333
4	des of	12	0.2250	0.7500	1.0000	0.1000	0.3333
5	des of	12	0.3375	0.4219	1.0000	0.2500	0.3125
6	des of	12	0.2625	0.3281	1.0000	0.3500	0.4375
7	des of	0	0.1389	0.2778	1.0000	0.2500	0.5000
8	des of	5	0.1389	0.1736	1.0000	0.2500	0.3125
12	topmost				1.0000	0.6000	0.7500
13	des of	0	0.1250	0.5000	1.0000	0.2500	1.0000
14	des of	7	0.1250	0.5000	1.0000	0.2500	1.0000
16	des of	2	0.1094	0.4375	1.0000	0.2500	1.0000

Table 8.1: Output Rules for Vague Belief (Set 1).

For the second set of experiment, we fix the parameters as $\theta_i = 0.1$, $\phi_s = 0.1$ and $\phi_c = 0.1$. Using the MARUB_V algorithm, 12 rules are discovered that are the same as those discovered in the first set of experiment.

In the second set of experiment, we decrease thresholds ϕ_s and ϕ_c in order to discover more interesting rules. It turns out that the number of discovered rules remains the same.

8.1.1 Experiments on Synthetic Data

The method of the generation of data is the same as that in the experiments for exact belief.

Two sets of experiment are done on this synthetic data with different sets of parameters. The user belief

$$203, 249 \Rightarrow 250$$

which has a confidence 0.5472 and a support 0.0893 in the database, is used for both sets of experiment.

For the first set of experiment, we fix the parameters as $\theta_i = 0.05$, $\phi_s = 0.08$ and $\phi_c = 0.05$. Using the MARUB_V algorithm, 32 rules are discovered.

- $Rule0 : item203, item249 \Rightarrow item250$
- $Rule8 : item202, item205 \Rightarrow item250$
- $Rule11 : item202, item249 \Rightarrow item250$
- $Rule12 : item202 \Rightarrow item250$
- $Rule27 : item204 \Rightarrow item250$
- $Rule31 : item205 \Rightarrow item250$
- $Rule41 : item206 \Rightarrow item250$
- $Rule76 : item9 \Rightarrow item250$
- $Rule84 : item10, item205 \Rightarrow item250$
- $Rule95 : item10 \Rightarrow item33$
- $Rule96 : item10, item205 \Rightarrow item33$

- $Rule97 : item10, item249 \Rightarrow item33$
- $Rule101 : item10, item249 \Rightarrow item250$
- $Rule102 : item10 \Rightarrow item250$
- $Rule110 : item11, item202 \Rightarrow item250$
- $Rule112 : item11, item205 \Rightarrow item250$
- $Rule117 : item11 \Rightarrow item31$
- $Rule119 : item11, item249 \Rightarrow item250$
- $Rule120 : item11 \Rightarrow item250$
- $Rule164 : item205 \Rightarrow item30$
- $Rule166 : item205 \Rightarrow item30, item31$
- $Rule170 : item203 \Rightarrow item31$
- $Rule171 : item203, item205 \Rightarrow item31$
- $Rule172 : item203, item249 \Rightarrow item31$
- $Rule177 : item202 \Rightarrow item33$
- $Rule178 : item202, item205 \Rightarrow item33$
- $Rule179 : item202, item249 \Rightarrow item33$
- $Rule180 : item205 \Rightarrow item33$
- $Rule181 : item205, item248 \Rightarrow item33$
- $Rule182 : item248, item249 \Rightarrow item33$
- $Rule188 : item248 \Rightarrow item250$

- $Rule189 : item249 \Rightarrow item250$

Rule 188 and Rule 189 are found to be one of the most general rules. Table 8.2 shows the source of interestingness, expected support, expected confidence, improvement, real support and real confidence of each rule.

For the second set of experiment, we fix the parameters as $\theta_i = 0.05$, $\phi_s = 0.1$ and $\phi_c = 0.1$. Using the MARUB_V algorithm, 30 rules are discovered.

- $Rule0 : item203, item249 \Rightarrow item250$
- $Rule8 : item202, item205 \Rightarrow item250$
- $Rule11 : item202, item249 \Rightarrow item250$
- $Rule12 : item202 \Rightarrow item250$
- $Rule27 : item204 \Rightarrow item250$
- $Rule31 : item205 \Rightarrow item250$
- $Rule41 : item206 \Rightarrow item250$
- $Rule76 : item9 \Rightarrow item250$
- $Rule92 : item10, item24 \Rightarrow item250$
- $Rule95 : item10 \Rightarrow item33$
- $Rule96 : item10, item205 \Rightarrow item33$
- $Rule97 : item10, item249 \Rightarrow item33$
- $Rule101 : item10, item249 \Rightarrow item250$
- $Rule102 : item10 \Rightarrow item250$

Rule		w. r. t.	Exp Sup	Exp Conf	Imp	Re Sup	Re Conf
0	des of	188	0.0726	0.3866	0.0923	0.0893	0.5472
8	des of	11	0.0667	0.7221	0.1754	0.1167	0.8748
11	des of	12	0.0797	0.5696	0.1525	0.1247	0.7221
12	des of	188	0.0981	0.3866	1.0000	0.1445	0.5696
27	des of	189	0.1218	0.4549	1.0000	0.0793	0.2962
31	des of	189	0.1341	0.4549	1.0000	0.2062	0.6995
41	des of	189	0.0838	0.4549	1.0000	0.0505	0.2743
76	des of	12	0.0554	0.5696	1.0000	0.0353	0.3632
84	des of	101	0.0601	0.8247	0.2219	0.1078	0.9214
95	des of	102	0.0350	0.2011	1.0000	0.0989	0.5674
96	des of	84	0.0310	0.2654	0.2779	0.0989	0.8453
97	des of	95	0.0545	0.5674	0.1582	0.0989	0.7256
101	des of	11	0.0857	0.7221	0.1264	0.1124	0.8247
102	des of	12	0.0993	0.5696	1.0000	0.1217	0.6982
110	des of	12	0.0311	0.5696	0.0910	0.0360	0.6606
112	des of	119	0.0353	0.6064	0.0632	0.0453	0.7626
117	des of	120	0.0414	0.1922	1.0000	0.0704	0.3268
119	des of	120	0.0594	0.5000	0.1064	0.0661	0.6064
120	des of	188	0.0833	0.3866	1.0000	0.1077	0.5000
164	des of	31	0.0518	0.1758	1.0000	0.0865	0.2934
166	des of	164	0.0114	0.0388	1.0000	0.0750	0.2544
170	des of	188	0.0506	0.1486	1.0000	0.0792	0.2326
171	des of	31	0.0270	0.2689	0.0785	0.0339	0.3729
172	des of	0	0.0343	0.2104	0.0713	0.0496	0.3039
177	des of	12	0.0416	0.1640	1.0000	0.0989	0.3898
178	des of	8	0.0336	0.2519	0.3515	0.0989	0.7414
179	des of	11	0.0359	0.2080	0.1828	0.0989	0.5727
180	des of	31	0.0594	0.2014	1.0000	0.0990	0.3358
181	des of	180	0.0735	0.3358	0.0822	0.0989	0.4180
182	des of	188	0.0456	0.1113	0.0531	0.0989	0.2327
188	topmost				1.0000	0.2870	0.3866
189	topmost				1.0000	0.2508	0.4549

Table 8.2: Output Rules for Vague Belief (Set 1).

- $Rule110 : item11, item202 \Rightarrow item250$
- $Rule112 : item11, item205 \Rightarrow item250$
- $Rule117 : item11 \Rightarrow item31$
- $Rule119 : item11, item249 \Rightarrow item250$
- $Rule120 : item11 \Rightarrow item250$
- $Rule164 : item205 \Rightarrow item30$
- $Rule166 : item205 \Rightarrow item30, item31$
- $Rule171 : item203, item205 \Rightarrow item31$
- $Rule177 : item202 \Rightarrow item33$
- $Rule178 : item202, item205 \Rightarrow item33$
- $Rule179 : item202, item249 \Rightarrow item33$
- $Rule180 : item205 \Rightarrow item33$
- $Rule181 : item205, item248 \Rightarrow item33$
- $Rule182 : item248, item249 \Rightarrow item33$
- $Rule188 : item248 \Rightarrow item250$
- $Rule189 : item249 \Rightarrow item250$

Rule 188 and 189 are found to be one of the most general rule. Most of the rules and their relationships are similar to those discovered in the first set of experiment with a few exceptions.

Table 8.3 shows the reason of interestingness, expected support, expected confidence, improvement, real support and real confidence of each rule.

Rule		w. r. t.	Exp Sup	Exp Conf	Imp	Re Sup	Re Conf
0	des of	188	0.0726	0.3866	0.0923	0.0893	0.5472
8	des of	11	0.0667	0.7221	0.1754	0.1167	0.8748
11	des of	12	0.0797	0.5696	0.1525	0.1247	0.7221
12	des of	188	0.0981	0.3866	1.0000	0.1445	0.5696
27	des of	189	0.1218	0.4549	1.0000	0.0793	0.2962
31	des of	189	0.1341	0.4549	1.0000	0.2062	0.6995
41	des of	189	0.0838	0.4549	1.0000	0.0505	0.2743
76	des of	12	0.0554	0.5696	1.0000	0.0353	0.3632
92	des of	101	0.0526	0.8247	0.1799	0.1070	0.9410
95	des of	102	0.0350	0.2011	1.0000	0.0989	0.5674
96	des of	97	0.0529	0.7256	0.2779	0.0989	0.8453
97	des of	95	0.0545	0.5674	0.1582	0.0989	0.7256
101	des of	11	0.0857	0.7221	0.1264	0.1124	0.8247
102	des of	12	0.0993	0.5696	1.0000	0.1217	0.6982
110	des of	120	0.0273	0.5000	0.0910	0.0360	0.6606
112	des of	119	0.0353	0.6064	0.0632	0.0453	0.7626
117	des of	120	0.0414	0.1922	1.0000	0.0704	0.3268
119	des of	120	0.0594	0.5000	0.1064	0.0661	0.6064
120	des of	188	0.0833	0.3866	1.0000	0.1077	0.5000
164	des of	31	0.0518	0.1758	1.0000	0.0865	0.2934
166	des of	164	0.0114	0.0388	1.0000	0.0750	0.2544
171	des of	0	0.0184	0.2104	0.0785	0.0339	0.3729
177	des of	12	0.0416	0.1640	1.0000	0.0989	0.3898
178	des of	8	0.0336	0.2519	0.3515	0.0989	0.7414
179	des of	11	0.0359	0.2080	0.1828	0.0989	0.5727
180	des of	31	0.0594	0.2014	1.0000	0.0990	0.3358
181	des of	182	0.0529	0.2327	0.0822	0.0989	0.4180
182	des of	188	0.0456	0.1113	0.0531	0.0989	0.2327
188	topmost				1.0000	0.2870	0.3866
189	topmost				1.0000	0.2508	0.4549

Table 8.3: Output Rules for Vague Belief (Set 2).

8.1.2 Experiments on Real Data

The data used in the experiments is the same as that used in the experiments for exact belief.

Two sets of experiment are done on this synthetic data with different sets of parameters. The user belief

$$Asia \Rightarrow [1990, 2000)$$

which has a confidence 0.4611 and a support 0.172 in the database, is used for both sets of experiment.

For the first set of experiment, we fix the parameters as $\theta_i = 0.05$, $\phi_s = 0.2$ and $\phi_c = 0.2$. Using the MARUB_V algorithm, 7 rules are discovered as shown below. Their ancestor/descendant relationships are shown in Figure 8.2.

- $Rule1 : France, Iraq \Rightarrow 1990$
- $Rule2 : France, Asia \Rightarrow 1990$
- $Rule4 : Iraq \Rightarrow 1990$
- $Rule8 : America, Asia \Rightarrow [1900, 2000)$
- $Rule88 : GeographicalLocation \Rightarrow [1900, 2000)$

Figure 8.2: The Ancestor/Descendant Relationships among Discovered Rules (Set 1)

For the case of exact user belief, rules are discovered starting at the point of user belief. In the case of vague user belief, it discovers rules starting from the most general ones. The most general rule discovered is Rule 88. The other 3 rules are more specific with respect to Rule 88. Remember that in the real data experiments for the exact user belief,

Also note that since we have only two trees in the hierarchy, all items have been considered according to the user belief $Asia \Rightarrow [1990, 2000)$ where *Asia* is in the Geographical location and $[1990, 2000)$ is in $[1900, 2000)$. Again, Rule 88 is not very useful since our hierarchy consists of only two trees. On the other hand, its descendant Rule 8 has a very high real confidence than the expected confidence with respect to Rule 88 and hence is very useful. We can conclude that, when America and Asia are mentioned, $[1900, 2000)$ is usually also mentioned. Table 8.4 shows the source of interestingness of the discovered rules.

Rule		w. r. t.	Exp Sup	Exp Conf	Imp	Re Sup	Re Conf
1	des of	2	0.0005	0.0838	0.2038	0.0039	0.5493
2	des of	88	0.0047	0.0838	0.1411	0.0130	0.3250
4	des of	88	0.0115	0.0838	1	0.0473	0.3455
8	des of	88	0.0133	0.5902	0.1000	0.0133	0.8333
88	topmost				1.0000	0.3423	0.5902

Table 8.4: Output Rules for Vague Belief (Set 1).

For the second set of experiment, we fix the parameters as $\theta_i = 0.05$, $\phi_s = 0.3$ and $\phi_c = 0.3$. Using the MARUB_V algorithm, 2 rules are discovered as shown below. Their ancestor/descendant relationships are shown in Figure 8.3.

- $Rule5 : Iraq, Europe \Rightarrow 1990$
- $Rule88 : GeographicalLocation \Rightarrow [1900, 2000)$

Since ϕ_s and ϕ_c are increased, we expect the number of rules discovered

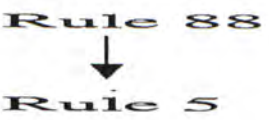

Figure 8.3: The Ancestor/Descendant Relationships among Discovered Rules (Set 2)

to decrease. In this case, it turns out that only 2 rules are discovered in this experiment. Table 8.5 shows the source of interestingness of each discovered rule.

Rule		w. r. t.	Exp Sup	Exp Conf	Imp	Re Sup	Re Conf
5	des of	88	0.0047	0.0838	0.0733	0.0163	0.4188
88	topmost				1.0000	0.3423	0.5902

Table 8.5: Output Rules for Vague Belief (Set 2).

Chapter 9

Conclusions and Future Work

9.1 Conclusions

We have proposed a new algorithm for incremental mining of association rules based on dynamic counting. A dynamic counting technique known as the Dynamic Itemset Counting (DIC) for large itemset counting in a *single* database was developed previously. Direct application of DIC in incremental updating is not efficient since available information stored in the original database is not useful. We still need to count the original database again and result in long computational time. Our new algorithm, called the IDIC_M algorithm and its variant, the IDIC_S algorithm, can efficiently conduct incremental mining of association rules based on dynamic counting. Moreover, it avoids the overhead incurred in the previous algorithm based on negative borders [35]. We have conducted extensive experiments to compare our algorithms with the FUP algorithm. The results illustrate that our two algorithms have better performance over the FUP algorithm.

While much work has been done on multiple-level association rule mining and rule discovery in the presence of user belief, very few research has attempted

to tackle both problems at the same time. In this thesis, we have proposed a new approach for mining multiple-level association rules from databases in the presence of user belief. We introduced a set of properties for rule interestingness on multiple-level association rules under user belief, namely the exact belief. To tackle this problem, we developed a multiple-level association rules mining algorithm, called MARUB_E which support the specification of the exact beliefs. In addition, we also introduced the vague belief and developed a multiple-level association rules mining algorithm, called MARUB_V to handle the vague beliefs. We showed the evaluation and analysis done on our new approach.

9.2 Future Work

Other than binary or categorical association rules, the incremental mining of quantitative association rules is also worth studying. As new incoming data may affect the data distribution pattern of the original database, new discretization may have to be done on the incremented database. Hence, the updating of the discovered association rules will be more complicated than that in the case of binary or categorical data.

Another area is to investigate the adoption of parallel and distributed computing. Some work on the parallel or distributed mining of association rules has been done [10, 29]. One possible research topic is the incremental mining of association rules in a distributed environment.

In this thesis, the proposed rule mining approach under user belief focused mainly on binary association rules. Quantitative association rule mining under user belief in the presence of hierarchy may also be studied.

Other than the two kinds of beliefs introduced in this thesis, many other kinds of beliefs exist in real life. Instead of the user beliefs in the form of association

rules, we can allow users to specify beliefs in other ways. For examples, beliefs can be in form of set of items that are interested to the user. Another possible extension is to allow the user to specify some kinds of measures so that the discovered rules can be ranked to improve the readabiliry of the resulting rules. Example of such measures can be the confidences or supports of the resulting rules.

Bibliography

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of International Conference on Very Large Databases*, pages 487–499, Santiago, Chile, 1994.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995.

[3] K. Ali and S. Manganaris. Partial classification using association rules. In *Proceedings of International Conference on Knowledge Discovery and Data Mining*, pages 115–118, Newport Beach, California, 1997.

[4] R. J. Bayardo Jr., R. Agrawal, and D. Gunopulos. Constraint-based rule mining in large, dense databases. In *Proceedings of International Conference on Data Engineering*, pages 188–197, Sydney, Australia, 1999.

[5] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 255–264, AZ, 1997.

[6] O. Buchter and R. Wirth. Discovery of association rules over ordinal data: A new and faster algorithm and its application to basket analysis. In *Proceed-*

ings of Pacific-Asia Conference on Knowledge Discovery and Data Mining, pages 36–47, Melbourne, Australia, 1998.

[7] M. S. Chen and J. Han. Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, 1996.

[8] D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of International Conference on Data Engineering*, pages 106–114, 1996.

[9] D. W. Cheung, S. D. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In *Proceedings of International Conference on Database Systems for Advanced Applications*, Melbourne, Australia, 1997.

[10] D. W. Cheung, V. T. Ng, A. W. Fu, and Y. Fu. Efficient mining of association rules in distributed databases. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):911–922, 1996.

[11] D. W. Cheung, V. T. Ng, and B. W. Tam. Maintenance of discovered knowledge: A case in multi-level association rules. In *Proceedings of International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, USA, 1996.

[12] D. W. Cheung and Y. Xiao. Effect of data skewness in parallel mining of association rules. In *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 48–60, Melbourne, Australia, 1998.

[13] R. Feldman, Y. Aumann, and W. Kloesgen. Maximal association rules: a new tool for mining for keyword co-occurrences in documents collections. In

Proceedings of International Conference on Knowledge Discovery and Data Mining, pages 167–170, Newport Beach, California, 1997.

[14] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. *Knowledge Discovery in Databases / G. Piatetsky and W. J. Frawley, eds.* AAAI/MIT Press, 1992.

[15] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proceedings of International Conference on Very Large Databases*, Zurich, Swizerland, 1995.

[16] M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. In *Proceedings of International Conference on Knowledge Discovery and Data Mining*, Newport Beach, California, 1997.

[17] M Kryszkiewicz. Representation association rules. In *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 198–209, Melbourne, Australia, 1998.

[18] C. M. Kuok, A. Fu, and M. H. Wong. Mining fuzzy association rules in databases. *ACM SIGMOD Record*, 27(1):41–46, 1998.

[19] D. Li, K. Di, D. Li, and X. Shi. Mining association rules with linguistic cloud models. In *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 392–393, Melbourne, Australia, 1998.

[20] B. Liu and W. Hsu. Post-analysis of learned rules. In *Proceedings of National Conference on Artificial Intelligence*, pages 828–834, Portland, Oregon, 1996.

[21] B. Liu, W. Hsu, and C. Shu. Using general impressions to analyze discovered classification rules. In *Proceedings of International Conference on Knowledge Discovery and Data Mining*, pages 31–36, CA, 1997.

[22] R. J. Miller and Y. Yang. Association rules over interval data. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 452–461, Tucson, Arizona, 1997.

[23] K. K. Ng and W. Lam. Updating of association rules dynamically. In *Proceedings of International Symposium on Database Applications in Non-Traditional Environments*, Kyoto, Japan, 2000. IEEE Computer Society Press.

[24] B. Padmanabhan and A. Tuzhilin. A belief-driven method for discovering unexpected patterns. In *Proceedings of International Conference on Knowledge Discovery and Data Mining*, pages 94–100, NY, 1998.

[25] A Ragel and C. Bruno. Treatment of missing values for association rules. In *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 258–270, Melbourne, Australia, 1998.

[26] R. Rastogi and K. Shim. Mining optimized support rules for numeric attributes. In *Proceedings of International Conference on Data Engineering*, pages 206–215, Sydney, Australia, 1999.

[27] A. Savasere, E. Omiecinski, and S. B. Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of International Conference on Very Large Databases*, pages 432–444, Zurich, Switzerland, 1995.

[28] T. Shintani and M. Kitsuregawa. Mining algorithms for sequential patterns in parallel: Hash based approach. In *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 283–294, Melbourne, Australia, 1998.

[29] T. Shintani and M. Kitsuregawa. Parallel mining algorithms for general-

ized association rules with classification hierarchy. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1998.

[30] A. Silberschatz and A. Tuzhilin. On subjective measures of interestingness in knowledge discovery. In *Proceedings of International Conference on Knowledge Discovery and Data Mining*, pages 275–281, Montreal, Canada, 1995.

[31] C. Silverstein, S. Brin, and R. Motwani. Beyond market baskets: Generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery*, 2, 1998.

[32] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proceedings of International Conference on Very Large Databases*, Zurich, Swizerland, 1995.

[33] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1996.

[34] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of International Conference on Extending Database Technology*, Arignon, France, 1996.

[35] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An efficient algorithm for the incremental updation of association rules in large databases. In *Proceedings of International Conference on Knowledge Discovery and Data Mining*, pages 263–266, Newport Beach, California, 1997.

[36] H. Toivonen. Sampling large databases for association rules. In *Proceedings of International Conference on Very Large Databases*, Mumbai (Bombay), India, 1996.

[37] K. Wang, S. H. W. Tay, and B. Liu. Interestingness-based interval merger for numeric association rules. In *Proceedings of International Conference on Knowledge Discovery and Data Mining*, pages 121–127, 1998.

[38] T. Washio, H. Matsuura, and H. Motoda. Mining association rules for estimation and prediction. In *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 417–419, Melbourne, Australia, 1998.

[39] K. Yoda, T. Fukuda, and Y. Morimoto. Computing optimized rectilinear regions for association rules. In *Proceedings of International Conference on Knowledge Discovery and Data Mining*, pages 96–103, Newport Beach, California, 1997.

[40] H. J. Zimmermann. *Fuzzy Set Theory and Its Applications.* Kluwer Academic Publishers, 1991.

CUHK Libraries

003803881