Maintaining incremental data mining association rules.

Zequn Zhou

University of Windsor
INFORMATION TO USERS

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

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

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

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

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

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

UMI®
Maintaining Incremental Data Mining Association Rules

by

Zequn Zhou

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science in
Partial Fulfillment of the Requirements
for the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2000
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-62311-4
Zequn Zhou 2000

© All Rights Reserved
Abstract

As new transactions update data sources and subsequently the data warehouse, the previously discovered association rules in the old database may no longer be interesting rules in the new database. Furthermore, some new interesting rules may appear in the new database. Generally, the process of generating new association rules using only the updated part of the database and the previously generated association rules is called incremental association rules maintenance.

A straightforward approach for generating association rules in the new database starts from scratch. Obviously, this approach is not efficient and is time consuming since many computations could be repetitive work. In order to save some computation time and reduce maintenance cost, an incremental approach utilizes the previous association rules results to generate new association rules in the updated database. Although existing incremental approaches avoid many recomputations and improve the performance of maintaining incremental association rules, some overheads for getting association rules in the updated database still exist.

This thesis proposes MAAP algorithm, an algorithm for maintaining incremental association rules aimed at making full use of the previous association rules results, reducing repetitive computations and which leads to better performance for incremental association rules maintenance. MAAP algorithm is based on the Apriori property and quickly generates some low level large itemsets from high level large itemsets, thus avoiding many repetitive computations. Meanwhile, MAAP algorithm stores delete part transactions and insert part transaction into an array respectively by scanning delete part database and insert part database one time. Thus it avoids many database scans.

The developed method presents better performance over some existing algorithms and reduces maintenance cost in some situations. Some experiments for conducting comparative analysis between Apriori, FUP2 and MAAP algorithm have been included.

Keywords: Maintaining Incremental Association Rules, Apriori Property, Data Warehouse, MAAP Algorithm
To my parents.
Acknowledgements

I would like to thank my supervisor Dr. Ezeife for her invaluable guidance, continuous encouragement and support during this thesis work. Her insightful thinking as well as her enthusiasm has been inspiring me through the research in data mining. The completion of this thesis would not have been possible without her help.

I would also thank my committee members Dr. Ingo Schmitt, Dr. Myron Hlynka and Dr. Liwu Li for spending their precious time to read this thesis carefully and make thoughtful suggestions.

Finally, I love to thank my parents for their everlasting love and support. Gratitude also goes to my beloved husband for his love and understanding.
Table of Contents

Abstract .................................................................................................................................... IV
Dedication ................................................................................................................................... V
Acknowledgements .................................................................................................................. VI
List of Tables ............................................................................................................................ IX
List of Figures .......................................................................................................................... X
Chapter 1 Introduction ............................................................................................................ 1
  1.1 Association Rules Discovery ......................................................................................... 3
    1.1.1 What is Association Rule? ..................................................................................... 3
    1.1.2 Criteria for Finding the Value of Association Rules .............................................. 3
    1.1.3 Association Rules Mining Process ......................................................................... 6
    1.1.4 The Application Area of Association Rules .......................................................... 7
    1.1.5 General Discussion of Association Rules ............................................................. 9
  1.2 Maintaining Incremental Association Rules ................................................................. 10
    1.2.1 What is Association Rules Maintenance? .............................................................. 10
    1.2.2 The Task of Maintaining Incremental Association Rules .................................... 10
    1.2.3 General Association Rules Mining Method .......................................................... 11
    1.2.4 Apriori Property .................................................................................................... 11
    1.2.5 Methods for Maintaining Incremental Association Rules .................................... 12
  1.3 Motivation of the Thesis ................................................................................................. 12
  1.4 Contributions of the Thesis ........................................................................................... 13
  1.5 Outline of the Thesis ....................................................................................................... 13
Chapter 2 Previous/Related Work ......................................................................................... 15
  2.1 A Review of Association Rules Generation Techniques .............................................. 15
  2.2 More Review on Different Types of Association Rules ............................................... 20
  2.3 Review on Maintaining Incremental Association Rules .............................................. 24
  2.4 Discussion on When to Maintain Incremental Association Rules ............................. 26
Chapter 3 Maintaining Incremental Association Rules ........................................... 28
  3.1 Introduction of MAAP Algorithm ................................................................. 28
    3.1.1 Apriori and FUP2 on Sample Database .............................................. 28
    3.1.2 The Proposed MAAP algorithm on Sample Database ......................... 32
  3.2 The Proposed Association Rule Maintenance Algorithm (MAAP) ............... 34
  3.3 More Discussion on MAAP Algorithm .................................................. 38
Chapter 4 Maintaining Incremental Association Rules on Multi-Dimensional Database 40
  4.1 Introduction to Multi-Dimensional Association Rules Mining .................. 40
    4.1.1 Basic Concepts ................................................................................. 40
    4.1.2 Method for Multi-Dimensional Association Rules generation .......... 42
    4.2 The Procedure for Maintaining Incremental Multi-Dimensional Association
       Rules on Multi-Dimensional Database .................................................... 49
Chapter 5 Performance Analysis ........................................................................... 55
  5.1 Theoretical Analysis ................................................................................. 55
  5.2 Two Case Experiments .............................................................................. 57
  5.3 Experimental Evaluation ........................................................................... 58
Chapter 6 Future Research and Conclusions ...................................................... 65
  6.1 Conclusions .................................................................................................. 65
  6.2 Future Work ................................................................................................. 65
References ............................................................................................................ 67
Appendices ........................................................................................................... 72
  Appendix A ....................................................................................................... 72
  Appendix B ....................................................................................................... 78
  Appendix C ....................................................................................................... 87
List of Figures

1.1 Data Mining Process ................................................................. 1
1.2 Association Rules Mining Process ........................................... 7
2.1 Large Itemsets ....................................................................... 18
2.2 Apriori Algorithm .................................................................. 19
2.3 An Example of Item Taxonomy .................................................. 21
2.4 A Relationship between Old Database and New Database ............ 24
3.1 MAAP Algorithm .................................................................... 37
3.2 Different Situations among Rule Changes in New Database .......... 39
3.3 A Relationship of Large Itemsets between Old Database and New Database 39
4.1 Data Cube Lattice Framework .................................................... 41
4.2 A Cube with Three Dimensions: Location, Product and Profit ........ 43
4.3 Algorithm for Mining Multi-Dimensional Association Rules on Multi-Dimensional Database ................................................. 45
4.4 The Result of An Example for Mining Association Rules on Multi-Dimensional Database ......................................................... 53
4.5 An Algorithm for Maintaining Incremental Multi-dimensional Association Rules on Multi-Dimensional Database ....................... 54
5.1 Different Situations among Large Itemsets Changes in the New Database ................................................................. 56
5.2 Maintenance Time for Different Maintenance Algorithm in Different Support Threshold ............................................................ 59
5.3 Maintenance Time for Different Maintenance Algorithm in Different Percent Change of Old Large Itemsets .................................... 63
List of Tables

Table 1.1: An Example of Customer Purchase Data .................................................. 3
Table 2.1: A Summarized Definition for Support of Itemset in
          Old Database and New Database ......................................................... 26
Table 3.1: Original Database Transaction Table .................................................... 29
Table 3.2: Updated Database Transaction Table ................................................... 29
Table 3.3: Large Itemsets for Apriori Algorithm .................................................. 30
Table 3.4: Candidate Itemsets for Apriori Algorithm ............................................. 30
Table 3.5: Itemsets $P_i$ and $Q_i$ in Each Iteration for FUP2 Algorithm Large
          Itemsets for FUP2 Algorithm ............................................................... 31
Table 3.6: Large Itemsets for MAAP Algorithm After First Step ............................. 33
Table 3.7: Large Itemsets for MAAP Algorithm .................................................... 34
Table 3.8: Candidate Itemsets for MAAP Algorithm ............................................. 34
Table 3.9: A Summary of Symbols ......................................................................... 38
Table 4.1: The Cube Table for Data Cube .............................................................. 44
Table 4.2: Set of items ............................................................................................. 47
Table 4.3: Large 1-itemsets ..................................................................................... 47
Table 4.4: Candidate 2-itemsets ............................................................................. 48
Table 4.5: Large 2-itemsets .................................................................................... 48
Table 4.6: Candidate 3-itemsets ............................................................................. 48
Table 4.7: Large 3-itemsets .................................................................................... 48
Table 4.8: Inserted Cube Table .............................................................................. 50
Table 4.9: Deleted Cube Table .............................................................................. 50
Table 5.1: Parameters ............................................................................................. 58
Table 5.2: Maintenance Time for Different Maintenance Algorithm in
          Different Support Thresholds ....................................................................... 59
Table 5.3: The Benefit Gain of CPU Execution Time between MAAP
          and FUP2 Algorithm in Case Study One .................................................... 60
Table 5.4: The Benefit Gain of CPU Execution Time between MAAP
          and Apriori Algorithm .............................................................................. 61
Table 5.5: The Improvement Ratio of CPU Execution Time between MAAP
          and FUP2 Algorithm .............................................................................. 61
Table 5.6: The Improvement Ratio of CPU Execution Time between MAAP
          and Apriori Algorithm .............................................................................. 61
Table 5.7: CPU Execution Time Between MAAP and FUP2 in Different
          Percentage Change of Old Large Itemsets ............................................... 63
Table 5.8: The Benefit Gain of CPU Execution Time between
          MAAP and FUP2 Algorithm in Case Study Two ......................................... 64
Chapter 1

Introduction

Data mining is a rapidly evolving area of data analysis which attempts to efficiently discover interesting rules or unexpected patterns from large collections of data. It helps end users extract useful business information from large databases. By using data mining techniques, it is possible to strike gold in unexpected places and extract patterns not previously discernable or obvious. Thus, data mining enables businesses to make intelligent business decisions and can serve as front-end to enterprise data warehouses [BS97]. Figure 1.1 illustrates the general idea and process of data mining whereby a data miner(user) runs a data mining application(e.g., MineSet, SAS miner) to generate some useful patterns or rules(e.g., association rules).

Figure 1.1: Data Mining Process
Association rule mining is a data mining technique which discovers strong associations or correlation relationships among data. Given a set of transactions, where each transaction consists of items, an association rule is an expression of the form \( X \rightarrow Y \), where \( X \) and \( Y \) are sets of items. An example of a simple rule is "98% of customers who purchase milk and bread also buy eggs" (this rule is expressed as \( \{\text{milk, bread}\} \rightarrow \{\text{eggs}\} \) and the rule has a confidence or accuracy of 98%). Since discovering all such rules may help market baskets or cross-sales analysis, decision making, and business management, mining association rules from large data sets has attracted a lot of attention. Association rules mining algorithms presented in this research area include [AIS93] [AS94] [BMS97] [HF95] [KMRTV94] [MY97] [NLHP98] [PCY95a] [FU94].

These algorithms mainly focus on how to efficiently generate association rules and how to discover the most interesting rules. However, data stored in the database are often updated making association rules discovered in the previous database possibly no longer valid in the new database. Furthermore, some new interesting rules may have been discovered in the new database. How to generate association rules in the updated database with reduced run time and maintenance cost is also an important research issue. Some works in incremental association rules maintenance include [CHNW96] [CLK97] [LC7] [CLK98].

This thesis proposes a new algorithm maintaining association rules with Apriori property (MAAP) for maintaining incremental association rules. This algorithm utilizes the Apriori property [AS94] to quickly compute the low level large itemsets (which contains few items) from high level large itemsets (which contains many items), thus avoiding many repetitive computation efforts.
1.1 Association Rules Discovery

1.1.1 Definitions

Definition 1.1: An association rule is a rule in the form of \( \{A_1, A_2, \ldots, A_m\} \rightarrow \{B_1, B_2, \ldots, B_n\} \), where \( A_i \) and \( B_j \) are items in the market basket (an item can be viewed as an attribute of a database table or a file). Such rules are usually interpreted as “when items \( A_1, A_2, \ldots, A_m \) occur together, it is often the case that items \( B_1, B_2, \ldots, B_n \) occur together as well in the same transaction”. A transaction in this case is a record of market purchase or simply a tuple in the database table.

Example 1: If breakfast cereal is purchased, then milk will be purchased. It is a rule expressed as \( \{\text{breakfast cereal}\} \rightarrow \{\text{milk}\} \)

Example 2: If pretzels and dry roasted peanuts are purchased, then beer will be purchased. It is a rule expressed as \( \{\text{pretzels, dry roasted peanuts}\} \rightarrow \{\text{beer}\} \)

Table 1.1 presents a database table used to define some terms connected to the topic of association rules discovery.

<table>
<thead>
<tr>
<th>Transaction ID (TID)</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>butter, egg, milk</td>
</tr>
<tr>
<td>2</td>
<td>bread, cheese, milk</td>
</tr>
<tr>
<td>3</td>
<td>bread, cheese, egg, milk</td>
</tr>
<tr>
<td>4</td>
<td>bread, cheese</td>
</tr>
</tbody>
</table>

Table 1.1: Customer Purchase Data
Definition 1.2: A transaction contains items: Assume $I = \{i_1, i_2, \ldots, i_m\}$ is a set of items. $D$ is a set of transactions, where each transaction $T$ is a set of items such that $T \subseteq I$. A transaction $T$ contains set of items $X$ in $I$ if $X \subseteq T$. An association rule is an implication of the form $X \rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \emptyset$. For example, given a simple database table as in Table 1.1, the set of items $I = \{\text{bread}, \text{butter}, \text{cheese}, \text{egg}, \text{milk}\}$. Since \{butter, egg\} is a set of items in $I$ and $\{\text{butter, egg}\} \subseteq \{\text{butter, egg, milk}\}$, transaction $T_1 = \{\text{butter, egg, milk}\}$ contains $\{\text{butter, egg}\}$.

\[ \square \]

Definition 1.3: The support for a set of items is the percentage of transactions in the database table that contain all of these items. The support for a set of items can also be given as the total number of transactions in the database table that contain all of these items. The support for a rule $X \rightarrow Y$ is the support for the set of items $X \cup Y$. For example, using Table 1, assume there is a set of items $\{\text{milk, bread}\}$. Since transaction 2 and 3 contain $\{\text{milk, bread}\}$, the support for the set of items $\{\text{milk, bread}\} = 2/4 * 100\% = 50\%$. The support for $\{\text{milk, bread}\}$ is also 2 transactions.

\[ \square \]

Definition 1.4: The confidence for a rule $X \rightarrow Y$ is the percentage of transactions in the database table that contain both $X$ and $Y$. For example, from Table 1, assume a rule $\{\text{milk}\} \rightarrow \{\text{bread}\}$ exists. Since transaction 2 and 3 contain $\{\text{milk, bread}\}$, transaction 1, 2 and 3 contain $\{\text{milk}\}$, the confidence for the rule $\{\text{milk}\} \rightarrow \{\text{bread}\} = 2/3 * 100\% = 66.7\%$.

\[ \square \]

Definition 1.5: MinSupport is a threshold percentage which is provided by user before doing association rules mining and used to discard less frequent itemset.

\[ \square \]

Definition 1.6: MinConfidence is a threshold percentage which is provided by user before doing association rules mining and used to discard less frequent association rules. User can adjust MinSupport and MinConfidence according to the association rules mining result for getting rules that are most useful in their problem domain.

\[ \square \]
**Definition 1.7:** *Antecedent and Consequent:* The rules themselves consist of two halves. The left side is called the antecedent (body) and the right side is called the consequent (head).

**Definition 1.8:** An itemset is referred to as a set of items, an itemset that contains k items is called a k-itemset. For example, itemset \{abc\} is a set of items which is \{a, b, c\}, since itemset \{abc\} contains 3 items, itemset \{abc\} is called 3-itemset.

**Definition 1.9:** *Large itemset:* An itemset X is large itemset (also called frequent itemset) if support of X is no less than MinSupport, otherwise X is called small itemset. For example, from Table 1.1, assume MinSupport =3 transactions, for itemset \{bread\}, it is included in transactions 2, 3 and 4. Thus, the support of \{bread\} is 3, which is not less than the MinSupport of 3. Therefore, itemset \{bread\} is a large itemset. For itemset \{egg\}, it is included in transactions 1 and 3, the support of \{egg\} is 2, which is less than MinSupport, thus, itemset \{egg\} is a small itemset.

1.1.2 Criteria for Finding the Value of the Association Rules

Association rules mining is the process of finding interesting association or correlation relationships among a large set of data, which means to identify sets of database items that frequently occur together, and then formulate rules with them.

Two measures, which are support and confidence, are normally used for evaluating association rules. Though some papers [KMRTV94] address other measures used for evaluating interesting rules, this thesis focuses on support and confidence.

Generally, support can be used to determine how often the rule is applied while confidence can be used to determine how often the rule is correct.
1.1.3 Association Rules Mining Process

The data that association rules mining deals with can usually be stored in a transaction table or a huge data warehouse table. Initial association rules mining mainly focuses on analysis of market basket data. A market basket data record is considered as a customer transaction record specifying all items purchased by the customer during one market visit and the record consists of transaction id and all items bought together, usually these record of purchases is stored in the transaction database table. If data are stored in a transaction database table, only association rules within one dimension can be generated. These rules are called single dimensional association rules. For example, in Table 1.1, all items in a customer transaction data are regarded as product dimension. The rule \( \{ \text{milk} \} \rightarrow \{ \text{bread} \} \) is a single dimensional rule, since item milk and bread come from the same product dimension. If data are stored in a huge data warehouse table, association rules among many dimensions can be generated. These rules are called multi-dimensional association rules. For example, a rule \( \{ \text{Seattle, backpack} \} \rightarrow \{ \text{goodprofit} \} \) is a multi-dimensional association rule, since all items involved in this rule come from different dimensions, Seattle comes from location dimension, backpack comes from product and goodprofit comes from profit dimension. Figure 1.2 shows the process for mining different types of association rules. This figure shows that the user can generate single dimensional association rules from transaction database table (or transaction file) and user also can generate multi-dimensional association rules from data warehouse. This thesis focuses on association rules mining from transaction database table, which generate single dimensional association rules.
1.1.4 The Application Area of Association Rules

Although association rules mining originated from the problem of market basket analysis, it can also be applied to many other problem domains including business, engineering, medicine, and finance.

* Market Basket Analysis.

Obtaining customers' buying pattern is essential for retailers to make decisions. With an understanding of this information, retailers will know what to put on sale, how to design coupons, how to place merchandise on shelves in order to maximize the profit. Association rules mining can provide such information. An effective mining application in the retail environment is market basket analysis or shopping basket analysis. It analyzes the attributes of customers shopping basket from electronic point of sale data and applies the findings to launch effective
promotions and advertising. For example, all rules that have "Diet Coke" as consequence may help to plan what the store should do to boost the sales of Diet Coke.

• Cross-sale
Strong competitive environment in the service industries and how to retain customers and make better use of the customer have become very important. Most companies are involved in providing more than one service or product. Since they know more about their customers through data analysis, targeting those customers with products that they do not already have is seen as a way of reaping quick profits. Cross-sales is the term given to this problem of attempting to sell a product to existing customers of the company who are not already customers of that particular product. Because customer databases are very large in large organizations and it is not easy to handle, association rules discovery techniques can incorporate domain expertise and discover useful knowledge for the domain expert to help solve cross-sales problems.

• Partial Classification
Partial classification of the data describes the discovery of models that show characteristics of the data classes, but may not cover all classes and all examples of any given class. Conventional classifiers would be ineffective when there are a very large number of attributes, and most of the values of each attribute are missing. This problem can be solved using association rules analysis. For example, the data consists of medical tests given to patients, usually, there are hundreds of medical tests but only a few of them are given to any single patient. Doctors can use rules and models discovered by association analysis to see whether any of the medical test results could be predicted by combinations of other test results. If such tests are found, they can be used to avoid giving patients redundant tests, or complex tests could be replaced with simple tests.
• Financial Services.

Association rules mining is used extensively by financial service industries. Security analysts are using this to analyze massive financial data in order to build trading and risk models for developing investment strategies. These products may be utilized in currency trading, stock selection, credit scoring, identifying fraud patterns, and mortgage screening in the near future.

1.1.5 General Discussion of Association Rules

Since the first introduction of association rules mining [AIS93], mining association rules from large databases has been the subject of numerous studies. These studies cover a broad spectrum of topics including fast algorithms based on the levelwise Apriori framework [AS94][KMRTV94], partitioning [SON95], incremental updating [LCK97] and parallel algorithms [SK98][PCY95b][MY97][HKK97], mining of generalized and multi-level rules [HF95] [SA95], mining of quantitative rules [SA96]. Other topics studied in this area include mining of multi-dimensional rules [FMT96] [KHC97] [LSW97][MY97], mining rules with item constraints and association-rule based query languages [NLHP98][SVA97] as well as mining partial periodicities [CSD98][HDY99]. Today, association rule mining technique has improved and allow users to control the mining process, which means that users can control the result of mining association rules through provision of some constraint association query, dynamic support and confidence. This technique enhances interaction between the computer and users, avoiding a high computational cost that is disproportionate to what the user wants and gets [NLH98][BAG99].
1.2 Maintaining Incremental Association Rules

1.2.1 What is Incremental Association Rules Maintenance?

As data are inserted or deleted from database, the previous association rules may no longer be interesting, new interesting rules could appear in the updated database. Generally, the process of generating new association rules using only the updated part of the database and the previously generated association rules is called maintaining incremental association rules.

1.2.2 The Task of Maintaining Incremental Association Rules

Since the data stored in the database is frequently changed, the previous rules may become stale, while novel rules could appear. If this change is not taken into account and previous rules are used for decision making and business analysis, a heavy loss may be incurred. Thus, maintaining incremental association rules is very important to business management.

Usually, database dealt with contains huge data, sometimes a database table can have one million tuples and one thousand attributes. If we generate our desired rules from such a huge database, it will cost much time and consume many system resources. When maintaining incremental association rules, the task is to make full use of the previous association rules results, reduce run time and use of system resources, therefore achieving lower maintenance cost.

1.2.3 General Association Rules Mining Method

Agrawal and Srikant in [AS94] propose the problem of mining association rules and decompose this problem into two subproblems, namely (1) generating all large itemsets
in the database and (2) using the large itemsets to generate the desired rules. A straightforward algorithm for step 2 is that for every large itemset \( l_u \), find all non-empty subsets of \( l_u \), for every such subset "a", output rules of the form \( a \rightarrow (l_u - a) \) if support \( (l_u) \) divided by support(a) is at least MinConfidence. The basic idea of this approach is to first get all large itemsets based on their support and then generate desired association rules by computing confidence among large itemsets. An important algorithm — Apriori algorithm is presented based on this approach.

Note: The computation cost in first step is much more than the cost in second step since first step needs many database scans, most association rules generations algorithms mainly are concerned with how to quickly generate all large itemsets in the database.

1.2.4 Apriori Property

An important property in association rules mining — Apriori property was introduced and proven in [AS94]. This property is used in Apriori algorithm for pruning many candidate itemsets before generating all large itemsets, thus many computations for the support of some itemsets can be avoided.

**Theorem Apriori property:** all non-empty subsets of a large itemset must be large. We can further induce this property, namely, if any itemset contains non large sub-itemset, then this itemset must be a non large itemset. For example, if itemset \( \{abc\} \) is large, then we can infer \( \{a\}, \{b\}, \{c\}, \{ab\}, \{bc\}, \{ac\} \) must be large; furthermore, if any of \( \{a\}, \{b\}, \{c\}, \{ab\}, \{bc\}, \{ac\} \) is not large, then \( \{abc\} \) must not be large.

1.2.5 Maintaining Incremental Association Rules Methods

- A straight forward method to deal with updating association rules is based on Apriori algorithm [AS94], which generates association rules in the new database
from the scratch. Obviously, this approach is not efficient and is time consuming since many computations could be eliminated if we utilize the previous association rules results in the old database.

- An incremental approach [CHNW96] utilizes the previous association rules results to generate all association rules in the new database. In fact, some large itemsets in the old database could remain large in the new database, for these large itemsets, it is unnecessary to recompute their support from scratch, since we already have their support in the old database. What we need to do is to scan the changed part between old database and new database and get the support from only the changed part of the database. Normally, the changed part of the database is much smaller than the whole database. In this case, much computation time can be saved.

1.3 Motivation of the Thesis

Although the existing incremental approaches improve the performance of maintaining incremental association rules, some overheads for getting association rules in the updated database still remain. The existing approaches compute the new candidate \((n+1)\)-itemsets using the corresponding old large \(n\)-itemsets. A drawback of this method is that computing high level large itemsets requires first scanning the database to compute large 1-itemsets, 2-itemsets, \ldots, \(n\)-itemsets. Computing the lower level large itemsets generally carries the huge overhead of doing several scans of a database or warehouse table since there are long lists of candidate itemsets at the lower level to scan the database for their supports.

An improved algorithm (MAAP) is proposed in this thesis for cutting down the overheads of doing database scan for computing the support of candidate itemsets at the lower level. This algorithm utilizes Apriori property for quickly generating large itemsets in the updated database. It starts to scan the new database from highest level large
itemsets of the old database, if some large itemsets remain large in the new database, their subsets must be inferred large itemsets in the lower level. For example, if a large 3-itemset is \( L_3 = \{123\} \), we can immediately infer that the following itemsets are large as well: \( \{12\}, \{13\}, \{23\}, \{1\}, \{2\}, \{3\} \). Using this principle, we can avoid the overhead of computing low level large itemsets, thereby achieving better response time and maintenance cost.

1.4 Contribution of the Thesis

This thesis presents a new algorithm (MAAP) for efficiently maintaining incremental association rules in the updated database, which starts by computing the high level large itemsets in the new database using the available high level large itemsets in the old database. Some parts of the \( n-1 \), \( n-2 \), ..., 1 level large itemsets can then be quickly generated by applying the Apriori property, thereby avoiding the overhead of calculating many lower level large items that involve many table scans. In addition, MAAP algorithm can also be applied for maintaining multi-dimensional association rules in multi-dimensional databases.

1.5 Outline of the Thesis

The rest of the thesis is organized as follows. Chapter 2 reviews some existing work related to the thesis. Chapter 3 presents a detailed description of the new algorithm (MAAP) for maintaining incremental association rules on transaction data. Chapter 4 discusses on applying MAAP algorithm for maintaining incremental association rules on multi-dimensional database. Chapter 5 presents some experimental results and
performance analysis for proposed MAAP algorithm. Chapter 6 gives conclusion and discusses future work.
Chapter 2

Previous/Related Work

In this chapter, current and previous research on generating and maintaining association rules is reviewed. The overview of related topics focuses on three themes: (1) review of basic approaches for generating association rules, (2) review of general approaches for maintaining association rules, and (3) discussion of when to maintain association rules.

2.1 A General Review of Association Rules Generation Techniques

Agrawal proposes the problem of mining association rules in [AIS93] and decomposes the problem of mining association rules into two subproblems, namely (1) generating all large itemsets in the database and (2) generating association rules in the database according to the large itemsets generated in the first step.

The following is some typical algorithms in generating all large itemsets in the database.

- Template algorithm is presented in [AIS93]. The algorithm makes multiple passes over the database. The frontier set for a pass consists of those itemsets that are extended during the pass. In each pass, the support for certain itemsets are measured. These itemsets called candidate itemsets, are derived from the tuples in the database and the itemsets contained in the frontier set. Associated with each itemset is a counter that stores the number of transactions in which the corresponding itemset has appeared. After one pass, the frontier set is adjusted for the next pass. This algorithm
is terminated when the frontier is empty. The disadvantage of this algorithm is that candidate itemsets are generated on-the-fly during the pass as data is being read, this results in many database scans.

- Apriori algorithm [AS94] is designed for generating association rules. This algorithm is more efficient than Template algorithm since it generates the candidate itemsets to be counted in a pass by using only the itemsets found large in the previous pass without considering the transactions in a database, candidate itemsets can be generated before database scan, therefore reducing many database scans. The basic idea of Apriori algorithm is to find all the large itemsets iteratively. In the first iteration, it finds the large 1-itemsets \( L_1 \) (\( L_1 \) denotes each large itemset in \( L_1 \) contains only one item). To obtain \( L_1 \), it first generates candidate itemsets \( C_1 \) (each candidate itemset in \( C_1 \) contains one item), namely all 1-itemsets of basket data, then the database is scanned for each itemset in the set \( C_1 \) to compute its support. The items with support greater than or equal to minimum support (MinSupport) are chosen as large items \( L_1 \). In the next iteration, apriori_gen function [AS94] is used to generate candidate set \( C_2 \) by performing a conditional join of \( L_1 \) and \( L_1 \). In joining \( L_1 \) and \( L_1 \), which means when one itemset \( X \) is chosen from the first \( L_1 \), another itemset \( Y \) is chosen from the second \( L_1 \), the last item of \( X \) must be less than the last item of \( Y \), the rest of items of \( X \) must be the same as those of \( Y \). Then it inserts those itemsets into \( C_2 \). The last step of apriori_gen function prunes all candidate itemset in \( C_2 \) which have any of their subsets not belonging to \( L_1 \). The large itemsets \( L_2 \) is again computed from the set \( C_2 \) by selecting items that meet the MinSupport requirement. The iterations go on by applying apriori_gen function until either \( L_i \) or \( C_i \) is empty. Finally, the large itemsets \( L \) is obtained as the union of all \( L_1 \) to \( L_{i+1} \).

Table 1.1 re-copied below is used to show how Apriori algorithm works. Assume a MinSupport of 2 transactions.
<table>
<thead>
<tr>
<th>Transaction ID (TID)</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>butter, egg, milk</td>
</tr>
<tr>
<td>2</td>
<td>bread, cheese, milk</td>
</tr>
<tr>
<td>3</td>
<td>bread, cheese, egg, milk</td>
</tr>
<tr>
<td>4</td>
<td>bread, cheese</td>
</tr>
</tbody>
</table>

- In the first iteration, the first step generates all candidate 1-itemsets $C_1 = \{\text{bread, butter, cheese, egg, milk}\}$, the second step scans database table to get the support of each itemset in $C_1$, then inserts all itemsets with support of no less than MinSupport into $L_1$. From Table 1, it can be seen that $\{\text{bread}\}$ appears in 3 transactions, $\{\text{butter}\}$ in 1 transaction, $\{\text{cheese}\}$ in 3 transactions, $\{\text{egg}\}$ in 3 transactions, thus, $L_1 = \{\text{bread, cheese, egg, milk}\}$.

- During the second iteration, the first step generates all candidate 2-itemsets $C_2 = \text{apriori\_gen}(L_1)$. The join of $L_1$ and $L_1$ gives $C_2 = \{\text{bread, cheese}, \{\text{bread, egg}, \{\text{bread, milk}\}, \{\text{cheese, egg}\}, \{\text{cheese, milk}\}, \{\text{egg, milk}\}\}$ (we do not need to prune any candidate itemset from $C_2$ since all non-empty subsets of each candidate itemset belong to $L_1$). Then all itemsets in $C_2$ with support of no less than MinSupport are inserted into $L_2$ to create $L_2 = \{\text{bread, cheese}, \{\text{bread, milk}\}, \{\text{cheese, milk}\}, \{\text{egg, milk}\}\}$. An example of a candidate itemset that would be pruned if it shows in $C_2$ is $\{\text{bread, butter}\}$. This is because a subset butter is not in $L_1$.

- During the third iteration, the first step generates all candidate 3-itemsets $C_3 = \text{apriori\_gen}(L_2)$. The join of $L_2$ and $L_2$ gives $C_3 = \{\text{bread, cheese, milk}\}$, (we do not need to prune any candidate itemset from $C_3$ since all non-empty subsets of each candidate itemset belong to $L_2$). Since the support of $\{\text{bread, cheese, milk}\}$ is 2, insert it into $L_3 = \{\text{bread, cheese, milk}\}$. 
• During the fourth iteration, the first step generates all candidate 4-itemsets $C_4 = \text{apriori\_gen}(L_3) = \emptyset$. Apriori algorithm is terminated. Final result for each level large itemsets is shown in Figure 2.1.

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{bread}</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>{cheese}</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>{egg}</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>{milk}</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$L_2$</th>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{bread, cheese}</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>{bread, milk}</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>{egg, milk}</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>{cheese, milk}</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$L_3$</th>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{bread, cheese, milk}</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 2.1: Large Itemsets

Since Apriori algorithm is very important for consequent work, this algorithm is shown in Figure 2.2.
Apriori algorithm

Begin

$L_1 = \{\text{large } 1\text{-itemsets}\};$

For (k=2; $L_{k-1} \neq \emptyset$; k++) do begin

$C_k = \text{apriori-gen}(L_{k-1}); //\text{new candidates}$

For all transactions $t \in D$ do begin

$C_t = \text{subset}(C_k, t); //\text{candidate itemset contained in } t$

For all candidate itemset $c \in C_t$ do

$c.\text{count}++;$

end

$L_k = \{c \in C_k | c.\text{count} >= \text{minsup}\}$

End

Answer = $\cup_k L_k;$

End

apriori-gen function

\{Insert into $C_k$

Select p.item$_1$, p.item$_2$, ..., p.item$_{k-1}$, q.item$_{k-1}$

From $L_{k-1}$ p, $L_k$: q

Where p.item$_1$ = q.item$_1$, ..., p.item$_{k-2}$ = q.item$_{k-2}$, p.item$_{k-1}$ = q.item$_{k-1}$;

Prune step: delete all itemsets $c \in C_k$ where some $(k-1)$-subset of $c$ is not in $L_{k-1}$\}

Figure 2.2: Apriori Algorithm

- AprioriTid [AS94] algorithm has the additional property that database is not used at all for counting the support of candidate itemsets after the first pass. Rather, the set $C_k^\prime$ is used for this purpose. Each member of the set $C_k^\prime$ is of the form <TID, \{X$_k$\}>, where each X$_k$ is a potentially large k-itemset present in the transaction with identifier TID. In this case, the number of entries in $C_k^\prime$ may be smaller than the number of transactions in the database, especially for large values of $k$.

- DHP algorithm (direct hashing and pruning) is proposed in [PCY95a] for efficient large itemset generation. DHP has two major features: one is efficient generation of large itemsets by utilizing a hashing technique and the other is effective reduction of transaction database size by employing pruning techniques. Generally, given a large database, the initial extraction of useful information from the database is usually the most costly part. DHP presents the technique of hashing to filter out unnecessary
itemsets for next candidate itemsets generation, especially effective for the
generation of candidate set for large 2-itemsets, where the number of candidate 2-
itemsets generated is smaller than that by previous methods [AIS93] [AS94], thus
resolving the performance bottleneck. In addition, the generation of smaller
candidate sets enables us to effectively trim the transaction database at a much
earlier stage of the iterations, thereby reducing the computational cost for later stages
significantly.

2.2 More Review on Different Types of Association Rules

Section 2.1 reviews some association rules generation algorithms on general association
rules (called single level association rules). In fact, for many applications, it is difficult to
find strong and interesting association rules among data items at the primitive levels of
abstraction. On the other hand, strong and interesting association rules are not restricted
from one dimensional database table, it could be from multi-dimensional database table.
Thus, many different kinds of association rules could be generated from different
scenarios. In this section, a review of some other types of association rules is discussed.

- Multi-level Association Rules Mining

Section 2.1 reviews did not consider the presence of taxonomies and restricted the
items in association rules to the leaf-level items in the taxonomy (a taxonomy is a
hierarchy on the items). However, if there are some taxonomies in the basket data,
using taxonomy for mining association rules may give more general rules and cut
some trivial rules. For example, given a taxonomy as shown in Figure 2.3, which
says that a jacket is an outerwear, which is a type of clothes; ski pants is an
outerwear, which is a type of clothes and so on. If we infer a rule that “people who
buy outerwear tend to buy shoes”, this rule may imply that “people who buy
jackets tend to buy shoes”, so we do not need to explicitly generate the implied
rule.
Agrawal and Srikant [SA95] proposes the problem of mining association rules at multi-level. A solution to the problem is to replace each transaction $T$ with an "extended transaction" $T'$, where $T'$ contains all the items in $T$ as well as all the ancestors of each item in $T$. For example if the transaction contains jackets, we would add Outerwear and Clothes to get the extended-transaction. The general method dealt with multi-level association rules adds all ancestors of each leaf level item into itemset of basket data, then apply Apriori algorithm to generate desired rules. Cumulate Algorithm is presented in [SA95]. Cumulate Algorithm is similar to Apriori algorithm, it adds the following optimizations and all itemsets of a certain size are counted in one pass. (i) filtering the ancestors added to transactions: That is, we do not have to add all ancestors of the items in a transaction, just need to add ancestors that are in one or more of the candidate itemsets being counted in the current pass. (ii) pre-computing ancestors: It pre-computes ancestors for each item and drops ancestors that are not present in any of the candidates at the same time. (iii) pruning itemsets containing an item and its ancestor.
• Quantitative Association Rules Mining

In the review above, it is assumed that the database table has an attribute corresponding to each item and a record corresponding to each transaction. The value of an attribute for a given record is "1" if the item corresponding to the attribute is present in the transaction corresponding to the record, otherwise, a given record is "0". This case is referred to as Boolean association rules problem. Relational tables in most business and scientific domains have richer attribute types. For example, attributes can be quantitative or categorical. Quantitative attributes have values of any quantitative number, such as age, income; whereas categorical attributes have values of belonging to a specific category, such as zip code, make of car.

Srikant and Agrawal [SA96] introduces the problem of mining association rules in large relational tables containing both quantitative and categorical attributes and decomposes the problem of discovering quantitative association rules into five steps: (i) determine the number of partitions for each quantitative attribute. (ii) for categorical attributes, map the values of the attributes to a set of consecutive integers. (iii) find the support for each value of both quantitative and categorical attributes. (iv) use the frequent itemsets to generate association rules by applying Apriori algorithm. (v) determine the interesting rules in the output.

• Multi-dimensional Association Rules

The previous review focuses on mining association rules based on transactional databases. This kind of rules are generally referred to as single-dimensional association rules. As the progress of OLAP and data warehousing techniques advance, we want to find association in multi-dimensional data warehouse, these kind of rules are referred to as multi-dimensional association rules. Kamber et al. [KHC97] proposes a data cube model for mining multi-dimensional association rules, multi-dimensional slicing and layered cube search technique is proposed.
Efficient algorithms are developed by either using an existing data cube or constructing a data cube on the fly. The general idea utilizes Apriori algorithm to find the frequent 1-itemsets in each dimension, and then use frequent (k-1)-itemsets to grow frequent k-itemsets by multi-dimensional slicing on the data cube. Since the items in an itemset comes from different dimensions, by using the available summary layers of data cube, the frequency for each itemset can be directly obtained from one cube cell.

• Constraint/Query Based Association Rules
In previous review, before we generate many kinds of rules, users only provide minimum support and minimum confidence, after that, users just wait to get the results. Actually, users can provide additional constraints on the rule pattern to be mined in order to get more interesting and more useful rules. Generally, this kind of mining is referred to as constraint-based association mining. Srikant et al. [SVA97] introduces a solution for mining association rules with item constraints. He considers constraints that are Boolean expressions over the presence or absence of items in the rules and integrate constraints into the association discovery algorithm. Thus, given a set of transactions D, a set of taxonomies G and a Boolean expression B, the problem of mining association rules with item constraints is to discover all rules that satisfy B and have support and confidence greater than or equal to the user-specified minimum support and minimum confidence respectively. Han et al. [NLHP98] proposes constrained association queries as a means of specifying the constraints to be satisfied by the antecedent and consequent of mined association rules and provides an architecture for exploratory association rules mining. He emphasizes user's control during the association rules mining, which means that users can control the mining process after the mining process has begun, and adjust the MinSupport and MinConfidence in order to get desired rules, thus, avoiding a high computational cost that is disproportionate to what the user wants and gets.
2.3 Review on Maintenance of Incremental Association Rules

Cheung et al in [CHNW96] presents the FUP algorithm, which can update the discovered association rules in a database when new tuples are inserted into the database. It utilizes the previous association rules mining result to generate new association rules, thereby cutting down on the amount of repetitive calculations performed scanning the database table to obtain the supports of candidate itemsets in each group. However, the drawback of FUP algorithm is that it only deals with the case when some new transactions are inserted into old database. A more general algorithm FUP2 was proposed in [CLK97]. This algorithm can handle more general cases because new transactions could be inserted into database or some transactions could be deleted from database. A detailed description of FUP2 algorithm is given below:

Assume that $D$ denotes the old database, $D'$ denotes the new database, $\Delta^-$ denotes the deleted part of old database, $\Delta^+$ denotes the newly inserted part to the old database, $D^*$ denotes unchanged part in the old database. The relationships between these data sets are illustrated in Figure 2.4.

![Figure 2.4: A Relationship between Old Database and New Database](image)
Let $S_x$ denote the support of itemset $X$ in the $D$, $S_x^+$ denotes the support of itemset $X$ in $\Delta^+$, $S_x^-$ denotes the support of itemset $X$ in $\Delta^-$, $S_x'$ denotes the support of itemset $X$ in $D'$, $S_x^*$ denotes the support of itemset $X$ in $D^*$. Thus, $D^* = D - \Delta^+ = D' - \Delta^-$, $S_x' = S_x + S_x^- - S_x$

These definitions are summarized in Table 2.1.

<table>
<thead>
<tr>
<th>Database</th>
<th>Support of itemset $X$</th>
<th>Large $k$-itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta^+$</td>
<td>$S_x^+$</td>
<td>---</td>
</tr>
<tr>
<td>$D^*$</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>$\Delta^-$</td>
<td>$S_x^-$</td>
<td>---</td>
</tr>
<tr>
<td>$D = \Delta^+ \cup D^*$</td>
<td>$S_x$</td>
<td>$L_k$</td>
</tr>
<tr>
<td>$D' = D^* \cup \Delta^-$</td>
<td>$S_x'$</td>
<td>$L_k$</td>
</tr>
</tbody>
</table>

Table 2.1: Summarized Definition for Support of Itemset in Old Database and New Database

All large itemsets $L$ and the support $S_x$ for all large itemsets in the old database $D$ are available information, thus, the update problem is to find $L'$ and $S_x', \forall X \in L'$ efficiently, given the knowledge of $D$, $D'$, $\Delta^+$, $\Delta^-$, $L$ and $S_x$ for all $X \in L$.

FUP2 algorithm utilizes the idea of Apriori algorithm, to find the large itemsets iteratively. The difference between FUP2 and Apriori is that FUP2 separates the candidate itemsets in the new database into two subsets in each iteration. That is, in $k$th iteration, candidate itemsets $C_k$ is divided into $P_k$ and $Q_k$, where $P_k$ is the intersection of $C_k$ and $L_k$, where $L_k$ is the previous large itemsets of size $k$ in the old database. $Q_k$ is the remaining part of $C_k$ not included in the large itemsets $L_k$, that is, $Q_k = C_k - (C_k \cap L_k)$.

For all itemsets $x$ in $P_k$, the support of $x$ ($S_x$) in the old database is known. Thus, in order to compute $S_x'$ for each itemsets in $P_k$, it only needs to scan $\Delta^+$, $\Delta^-$ to get $S_x^-$, $S_x^+$ making $S_x' = S_x + S_x^- - S_x^-$. If $A_x' >= |D|^s/\delta$ (s% is MinSupport), then add $x$ into $L_k'$. For each itemset $x$ in $Q_k$, since the support of $x$ ($S_x$) in the old database is unknown, first scan
changed part of database $\Delta^-$ and $\Delta^+$, prune away some itemsets from $Q_k$ whose support in the changed part of the database is less than $(|\Delta^-| - |\Delta^+|) \times s\%$, that is, $S_x^+ - S_x^- \leq (|\Delta^-| - |\Delta^+|) \times s\%$. For the remaining itemsets $x \in Q_k$, scan unchanged part of database $D^+$ to find out their support $S^+$ and then get their support in the updated database $A_x^+$, where $A_x^+ = S^+ + S_x^+ - S_x^-$. If $A_x^+ \geq |D^+| \times s\%$, add $x$ to $L_k^+$.

### 2.4 Discussion on When to Maintain Incremental Association Rules

Although FUP2 algorithm can be used to generate the precise association rules in the updated database, it needs time to scan the database to compute support of each itemset. In particularly, when the change in database does not affect the association rules in the old database, the effort for maintaining association rules to get all large itemsets in the updated database could be wasted. Thus, a new problem comes out, that is when should rule maintenance algorithm be applied to get new rules? DELI algorithm [LCK98] is proposed to solve this problem. This algorithm employs sampling techniques for estimating the differences between the association rules in the database before and after the database is updated. When the differences are large, FUP2 algorithm is recommended. DELI algorithm gives an approximate upper bound on the amount of changes in the set of association rules introduced by the new transaction. If the bound is low, it indicates that the amount of changes in association rules in the new database is small. So, we can keep the old association rules as an approximation of association rules in the new database. If the bound is high, it indicates that the amount of changes in association rules in the new database is large. So, it is necessary to update the association rules in the new database. The basic procedure of DELI algorithm utilizes FUP2 algorithm, first obtain a random sample of the database from the old database, then in $k$th iteration, candidate itemsets $C_k$ is divided into $P_k$ and $Q_k$, where $P_k$ is the intersection of $C_k$ and $L_k$, $L_k$ is the previous large itemsets of size $k$ in the old database. $Q_k$ is the
remaining part of \( C_k \), that is, \( Q_k = C_k - (C_k \cap L_k) \). Find all large itemsets from \( P_k \) and insert them into \( LL_k \), where \( LL_k \) represents itemsets which are large itemset in both of the old database and the new database. Get itemsets that are large in the old database but not large in the new database and insert them into \( SS_k \), e.g. \( SS_k = (L_k - C_k) \cup (P_k - LL_k) \). For each itemset \( x \) in \( Q_k \), which represents the new candidate sets that are not part of the old large itemsets. Find all large itemsets \( x \) from \( Q_k \), then insert them into \( L_k'' \), \( L_k'' \) represents itemsets which contains all large itemsets not being large in the old database. Next obtain the set \( T' \) as \( SS_1 \cup LL_i \) for \( i=1..k \) e.g. \(|T'| = |SS_1| + \ldots + |SS_k| + |LL_1| + \ldots + |LL_k|\), then divide as \(|T'|/|L|\). If \(|T'|/|L|\) \( \geq a \) (\( a \) is the input value of DELI algorithm for evaluating amount of change in the new database), it manifests the need for a rule-update operation, otherwise increase \( k \) and go to next iteration.
Chapter 3

Maintaining Incremental Association Rules on Transaction Data

3.1 Introduction of MAAP Algorithm

In order to clearly illustrate the MAAP algorithm which is presented in this thesis, first an example is used to show how MAAP works. This example is similar to that used in [CLK97]. Since MAAP algorithm is very related to both the Apriori and FUP2 algorithms, both of these algorithms are first applied to this example in section 3.1.1 before applying MAAP to the same example in section 3.1.2.

3.1.1 Apriori and FUP2 on Sample Database

Suppose there is a database D with the set of items, I={A, B, C, D, E, F} which requires MinSupport of 3 transactions. A simple database transaction table is given as Table 3.1. If transaction 400 is deleted, new transaction 700 B D E is added, an updated database which is suitable for FUP2 is obtained. The updated new database table is as given in Table 3.2.
Table 3.1: Original Database Transaction Table

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>A</td>
</tr>
<tr>
<td>200</td>
<td>A</td>
</tr>
<tr>
<td>300</td>
<td>B</td>
</tr>
<tr>
<td>400</td>
<td>A</td>
</tr>
<tr>
<td>500</td>
<td>A</td>
</tr>
<tr>
<td>600</td>
<td>A</td>
</tr>
</tbody>
</table>

Table 3.2: Updated Database Transaction Table

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>A</td>
</tr>
<tr>
<td>200</td>
<td>A</td>
</tr>
<tr>
<td>300</td>
<td>B</td>
</tr>
<tr>
<td>500</td>
<td>A</td>
</tr>
<tr>
<td>600</td>
<td>A</td>
</tr>
<tr>
<td>700</td>
<td>B</td>
</tr>
</tbody>
</table>

To compute the large itemsets in D, Apriori algorithm first generates the candidate set $C_1 = \{A, B, C, D, E, F\}$, then scans D to obtain the support of each itemset in $C_1$. Then, it throws away the item F which has a support that is lower than 3. So, $L_1 = \{A, B, C, D, E\}$. In the second iteration, Apriori computes $C_2 = \text{apriori\_gen}(L_1) = \{AB, AC, AD, AE, BC, BD, BE, CD, CE, DE\}$. Then, scans D to obtain the support of each itemset in $C_2$. This results in an $L_2 = \{AB, AC, AD, AE, BC, BD, CD\}$. During the third iteration, $C_3 = \text{apriori\_gen}(L_3) = \{ABC, ABD, ACD, BCD\}$. Apriori scans D and gets the support of
these itemsets to generate \( L_3 = \{ABC, ACD, BCD\} \). Next step computes \( C_4 = \text{apriori\_gen}(L_3) = \{\}\), causing the algorithm to terminate and resulting in an overall Large itemsets \( L = L_1 \cup L_2 \cup L_3 \), which is, \( \{A, B, C, D, E, AB, AC, AD, AE, BC, BD, CD, ABC, ACD, BCD\} \). A list of all large itemsets is shown as Table 3.3. A list of all candidate itemsets are shown as Table 3.4.

<table>
<thead>
<tr>
<th>Large 1-itemsets</th>
<th>{A, B, C, D, E}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large 2-itemsets</td>
<td>{AB, AC, AD, AE, BC, BD, CD}</td>
</tr>
<tr>
<td>Large 3-itemsets</td>
<td>{ABC, ACD, BCD}</td>
</tr>
</tbody>
</table>

Table 3.3: Large Itemsets for Apriori Algorithm

<table>
<thead>
<tr>
<th>Candidate 1-itemsets</th>
<th>{A, B, C, D, E, F}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Candidate 2-itemsets</td>
<td>{AB, AC, AD, AE, BC, BD, BE, CD, CE, DE}</td>
</tr>
<tr>
<td>Candidate 3-itemsets</td>
<td>{ABC, ACD, BCD}</td>
</tr>
<tr>
<td>Candidate 4-itemsets</td>
<td>{}</td>
</tr>
</tbody>
</table>

Table 3.4: Candidate Itemsets for Apriori Algorithm

Applying the FUP2 algorithm on the same database \( D \) with \( I = \{A, B, C, D, E, F\} \) and with MinSupport of 3 transactions, proceeds as follows. In the first step, we get the new \( C_1 \) as the previous \( C_1 \) in the old database. That is, \( C_1 = \{A, B, C, D, E, F\} \). Then, comparing \( C_1 \) with \( L_1 \) (as computed with the Apriori above), would lead to breaking \( C_1 \) into two parts, (1) the set of elements common to both \( C_1 \) and \( L_1 \), called set \( P_1 = C_1 \cap L_1 = \{A, B, C, D, E\} \) and (2) the set of elements in \( C_1 \), which are not in the first set \( P_1 \), called set \( Q_1 = C_1 - C_1 \cap L_1 = \{F\} \). FUP2 proceeds to compute the support of each itemset in \( P_1 \), to obtain all
large itemsets in \( P_1 \) that are still large itemsets in the new database in order to include them to the new large itemset \( L_1 \). It further computes the support of each itemset in \( Q_1 \) in the new database to see if these previous small items are now large. With this example, \( F \) is still a small itemset in the new database. Thus, the updated new level 1 large itemset, \( L_1' = \{A, B, C, D, E\} \). The next step applies \texttt{apriori-gen} function to generate candidate itemsets \( C_2 \), then goes on with the rest of the iterations to compute \( L_2', \ldots, L_k' \) when \( L_{k+1}' \) or \( C_{k+1}' \) is empty. The result of each step is shown below. The \( C_2 \) generated from \( L_1' \) is \( C_2 = \{AB, AC, AD, AE, BC, BD, BE, CD, CE, DE\} \), which is broken into \( P_2 \) and \( Q_2 \), \( P_2 = \{AB, AC, AD, AE, BC, BD, CD\} \), \( Q_2 = \{BE, CE, DE\} \). \( L_2' = \{AB, AC, AD, BC, BD, BE, CD\} \) is computed from \( P_2 \) and \( Q_2 \). \( C_3 \) is generated from \( L_2' \), \( C_3 = \{ABC, ABD, ACD, BCD\} \) and when broken into \( P_3 \) and \( Q_3 \) gives \( P_3 = \{ABC, ACD, BCD\} \), \( Q_3 = \{ABD\} \). The new level 3 large itemsets, \( L_3' = \{ABC, ACD, BCD\} \). A list of itemsets for \( P_i \) and \( Q_i \) in each iteration is shown as Table 3.5.

### Table 3.5: Itemsets \( P_i \) and \( Q_i \) in Each Iteration for FUP2 Algorithm

<table>
<thead>
<tr>
<th>( P_i )</th>
<th>{A, B, C, D, E}</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q_i )</td>
<td>{F}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( P_2 )</th>
<th>{AB, AC, AD, AE, BC, BD, CD}</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q_2 )</td>
<td>{BE, CE, DE}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( P_3 )</th>
<th>{ABC, ACD, BCD}</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q_3 )</td>
<td>{ABD}</td>
</tr>
</tbody>
</table>
3.1.2 The Proposed MAAP algorithm on Sample Database

Agrawal and Srikant [AS94] propose an Apriori property, which states that "All non-empty subsets of a large itemset must be large". For example, if a large 3-itemset is \( L_3 = \{123\} \), we can immediately infer that the following itemsets are large as well: \( \{12\}, \{13\}, \{23\}, \{1\}, \{2\}, \{3\} \). Based on this principle, when association rules is to be maintained in the updated database, the large itemsets can be computed from the highest level large itemsets, that is, from \( L_k \). If any itemset in \( L_k \) is still large in the new database, its lower level subset itemsets are included to their appropriate level large itemsets in \( L_{k-1}, L_{k-2}, \ldots, L_1 \). For example, since \( L_3 = \{123\} \) is confirmed to be still large in the new database, MAAP includes \( \{12\}, \{13\}, \{23\} \) to \( L_2 \) and \( \{1\}, \{2\}, \{3\} \) to \( L_1 \). By so doing, some computation time is saved. In particular, if many of the old large itemsets in the old database remain large in the new database, the MAAP algorithm should be more efficient in maintaining association rules. For example, utilizing the large itemsets computed in section 3.1.1 as the result in the old database, the MAAP algorithm proceeds by checking if each itemset in \( L_3 = \{ABC, ACD, BCD\} \) is still large in the new database. Since ABC is large in the new database, AB, AC, BC, A, B, C are also large itemsets. Thus, ABC is added to \( L_3' \) (\( L_i' \) denotes the large i-itemsets in the new database), AB, AC, BC are added to \( L_2' \), and A, B, C are added to \( L_1' \). It continues to test the next itemset in \( L_3 \) which is ACD and since ACD is large, AC, AD, CD, A, C, D are also large itemsets. MAAP adds ACD to \( L_3' \), if not already a member of this large itemset. After testing ACD, \( L_3' \) has elements ABC, ACD; \( L_2' \) has AB, AC, BC, AD, CD; \( L_1' \) has A, B, C, D. Next, the last itemset in \( L_3 \) is tested, which is BCD. BC, CD, BD, B, C, D are large itemsets since BCD is a large itemset. These itemsets are included to \( L_3', L_2' \) and \( L_1' \) respectively. The temporary results at this stage are: \( L_3' = \{ABC, ACD, BCD\} \), \( L_2' = \{AB, AC, BC, AD, CD, BD\} \), \( L_1' = \{A, B, C, D\} \). These are not yet the final updated large itemsets in the new database. By using this example, it can be shown that some parts of the new large itemsets do not need to be re-computed in the new database. However,
these large itemsets are only large itemsets in the old database that remain large itemsets in the new database. A list of large itemsets generated by this step is shown as Table 3.6

<table>
<thead>
<tr>
<th>Large 3-itemsets</th>
<th>{ABC, ACD, BCD}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large 2-itemsets</td>
<td>{AB, AC, BC, AD, CD, BD}</td>
</tr>
<tr>
<td>Large 1-itemsets</td>
<td>{A, B, C, D}</td>
</tr>
</tbody>
</table>

Table 3.6: Large Itemsets for MAAP Algorithm After First Step

The next step checks the itemsets in each of the old \( L_1 \) to \( L_{k-1} \) large itemsets not yet in the corresponding new \( L_1' \) to \( L_{k-1}' \) large itemsets to see if they are still large in the new database. Continuing with the running example, \( L_1 - L_1' = \{E\} \) represents large 1-itemsets in the old database not yet in the so far computed \( L_1' = \{A, B, C, D\} \). There is need to scan the new database to obtain the support of item \( E \) which is included in the new large itemset if this support is greater than or equal to minimum support. Since \( E \) is large, the algorithm adds \( E \) to \( L_1' \). The same way \( L_2 - L_2' = \{AE\} \) is also used to compute additional elements of \( L_2' \) as \( \emptyset \) since \( AE \) is not large. Next getting all large itemsets that are not large in the old database and large in the new database. For 1-itemset, subtracts the large 1-itemsets from new candidate itemsets \( C_1 \). Thus, \( Q_1 = C_1 - L_1' = \{F\} \). Since scan \( \Delta^*, \Delta^* \), get \( S_F = 0, (\Delta^* - |\Delta^*|) \cdot s% = 0 \) (in this example, \( \text{MinSupport}=3 \), total number of transactions is 6, thus \( s\%=50\%) \), then \( F \) is still not a large itemset. For 2-itemsets, \( Q_2 = C_2 - L_2' = \{BE, CE, DE\} \). After scanning new database table, only \( BE \) is a large itemset and this is included in the \( L_2' \) to make \( L_2' = \{AB, AC, AD, BC, BD, BE, CD\} \). For 3-itemsets, \( Q_3 = C_3 - L_3' = \{ABD\} \). After scanning new database table, only \( ABD \) is not large itemset. For 4-itemsets, \( C_4 = \text{apriori}_\text{gen}(L_3') = \emptyset \), MAAP is terminated. The final large itemsets for all levels is now \( L = \{A, B, C, D, E, AB, AC, AD, BC, BD, BE, CD, ABD, ACD, BCD\} \). A list of all large itemsets is shown as Table 3.7. A list of all candidate itemsets are shown as Table 3.8.
### Table 3.7: Large Itemsets for MAAP Algorithm

<table>
<thead>
<tr>
<th>Large 1-itemsets</th>
<th>{A, B, C, D, E}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large 2-itemsets</td>
<td>{AB, AC, AD, BC, BD, BE, CD}</td>
</tr>
<tr>
<td>Large 3-itemsets</td>
<td>{ABC, ACD, BCD}</td>
</tr>
</tbody>
</table>

### Table 3.8: Candidate Itemsets for MAAP Algorithm

<table>
<thead>
<tr>
<th>Candidate 1-itemsets</th>
<th>{A, B, C, D, E, F}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Candidate 2-itemsets</td>
<td>{AB, AC, AD, A\E, BC, BD, BE, CD, CE, DE}</td>
</tr>
<tr>
<td>Candidate 3-itemsets</td>
<td>{ABC, ABD, ACD, BCD}</td>
</tr>
<tr>
<td>Candidate 4-itemsets</td>
<td>{}</td>
</tr>
</tbody>
</table>

### 3.2 The Proposed Association Rule Maintenance Algorithm (MAAP)

The steps in the MAAP algorithm being proposed in this thesis are discussed below.

- **Step 1:** Compute Parts of New large itemsets using only itemsets that were large in the old database, and are guaranteed to still be large in the new database because of a superset itemset in a higher level new large itemset.

  Starting with highest level large itemsets in the old database, such as \(L_k\), we test each itemset of \(L_k\) in the new database, by computing their support scanning only the changed part of the database, to identify any large itemsets in the new database, which are added to \(L_k\). \(L_k\) denotes large k-itemsets in the new database and for each element in this new large itemsets, the algorithm computes the union of all its non-empty subsets, which are included in the appropriate level new large itemsets \(L_{k-1}\), \(\ldots\), \(L_1\) respectively. Continue to test next itemsets in \(L_k\) and include in \(L_k\) if it is still
a large itemset in the new database, until all itemsets in \( L_k \) have been tested. This step has computed only part of new large itemsets \( L_k', L_{k-1}', \ldots, L_1' \), which come from large itemsets in the old database that continue to be large itemsets in the new database. If there are no large itemsets in the new database, the next lower level large itemset \( k=k-1 \) in the old database is used to do the computation.

- Step 2: Compute for each new large itemset additional large itemsets that were large in the old database but not computed in the first step because their superset higher level itemset is small in the new database, but these old lower level large itemsets may still be large in the new database.

The MAAP algorithm compares \( L_1' \) with \( L_1 \) to get the set of old large items not yet in the new large items, \( X_1 = L_1 - L_1' \), \( X_k \) denotes the itemsets which are large in the old database, but we are not sure if they are still large in the new database. It tests each itemset \( X \) in \( X_1 \), if \( X \) is large, it is added to \( L_1' \); if not large, it means that itemset is large in the old database, but it becomes small in the new database, in which case MAAP now removes any superset of this new small itemset from the higher level old large and small itemsets \( L_2, L_3, \ldots \) until \( L_{k-1} \). The purpose of this cleaning process is to cut down on the number of old large or candidate itemsets in the old database waiting to be confirmed large in the new database. Since a superset of an itemset cannot be large when the itemset is small, this rule is applied to achieve the reduction. For example, if \( F \) is large in the old database, but it becomes small in the new database, we can cut some itemsets that include \( F \) in the new high level large and candidate itemsets, namely, since \( F \) is large 1-itemsets in the old database, \( AF, CF \) in the large 2-itemsets or candidate 2-itemsets have to be removed as those can no longer be large itemsets at a higher level. The principle is that any large itemset has its subsets also large. On the other hand, if any subset of a higher level itemset is not large, then it cannot be large either.
• Step 3: Compute additional large itemsets that are not large in the old database and large itemsets in the new database. New candidate itemsets $C_k$ substracts $L_k'$ and get $Q_k = C_k - L_k'$ ($Q_k$ denotes all k-itemsets that are not large in the old database and may be large in the new database). Testing each itemset in $Q_k$, prune away some itemsets $x$ from $Q_k$ which their support with $S_x = \langle (|\Delta'| - |\Delta|) * s\% \rangle$ ($s\%$ is MinSupport, $|\Delta'|$ is the total number of inserted database, $|\Delta|$ is the total number of deleted database). For the remaining itemsets $X \in Q_k$, scan unchanged part of database $D^*$ to find out their support and add this support to $S_X$ to get $A_X'$. If $A_X' \geq |D| * s\%$, add $X$ to $L_k'$. Increment $k$ and then go to step 3 until $k$=the highest level $k$ of large itemset in the old database.

• Applying apriori_gen function to adjust the highest level $k$ of large itemset in the new database. When $C_k$ or $L_k'$ is empty, MAAP is terminated.

A formal presentation of MAAP algorithm is shown in Figure 3.1.

```
apriori_gen(L_i)
{Insert into $C_{i-1}$
Select p.item_1, p.item_2, ..., p.item_5, q.item_i
From L_i p, L_i q
Where p.item_1 = q.item_1, ..., p.item_5 = q.item_1, p.item_i < q.item_i;
Prune step: delete all itemsets $c \in C_{i-1}$ such that some $i$-subset of $c$ is not in $C_i$}
```
MAAP Algorithm - Maintenance of Association Rules

Algorithm MAAP()
Input: set of old large itemsets ($L_i$) and old highest level (k)
Output: New large itemsets ($L_i'$) in the new database, new highest level k

Begin
// generate new large itemsets from old large itemsets at level k //
oldk = k; // save highest level //
$L_k = \emptyset$;
while k >= 1 begin //
for all itemsets x in $L_k$ begin
if x is large in the updated database then begin
    generate all non empty subsets of x;
    add x to $L_k$;
    add all subsets of x to appropriate $L_{k-1}, \ldots, L_1$;
end // end of if/\nif $L_k = \emptyset$ then k = k-1 else exit;
end // end of while//
// generate new itemsets from old large itemsets not yet in $L_i$ //
i = 1;
while (i <= oldk) do begin
    $X_i = L_i - L_i$ ; // compare $L_i$ to $L_i$
    for all itemsets x in $X_i$ begin
        if x is large itemset in the new database then
            add x to $L_i$;
        else begin
            delete superset of x from high level large itemsets;
            for all itemsets x in $Q_i (Q_i = C_i - L_i)$ begin
                if x is large itemset in the new database then
                    add x to $L_i$;
            end // end of while//
        end // end of if/\n    end // end of while//
    newk = i - 1;
end // handle the case when new database has a higher level // \\nend() do begin
    $C_{newk+1} = \text{Apriori}_gen(I_{newk+1})$; // join $L_{newk+1}$ with $L_i$
generate more possible large
    itemsets//
    newk = newk + 1;
    if $C_{newk} = \emptyset$ exit // while loop // \;
end // end of while//
$L = \cup L_i$ ;
end // of MAAP //

Figure 3.1: MAAP Algorithm for Maintaining Incremental Association Rules on Transaction
3.3 More Discussion on Proposed MAAP Algorithm

From proposed MAAP algorithm, we notice that although some low level large itemsets can be obtained from high level large itemsets by applying Apriori property, in order to generate precise association rules, the precise support for those low level large itemsets which are obtained from high level large itemsets must be computed. In this situation, in order to avoid computing the support for those low level large itemsets, a tolerance for the support of those low level large itemsets must be considered and does not affect the final association rule generation results.

An induction for tolerance is shown as below:

Assume XY is a large itemset in the new database and X→Y is a rule in the old database. Let XY_{new} denote the support for XY in the new database, X_{new} denote the support for X in the new database, XY_{old} denote the support for XY in the old database, X_{old} denote the support for X in the old database, c denote the confidence for rule X→Y in the old database, a summary of these symbols are shown as Table 3.9.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XY_{new}</td>
<td>the support for XY in the new database</td>
</tr>
<tr>
<td>X_{new}</td>
<td>the support for X in the new database</td>
</tr>
<tr>
<td>XY_{old}</td>
<td>the support for XY in the old database</td>
</tr>
<tr>
<td>X_{old}</td>
<td>the support for X in the old database</td>
</tr>
<tr>
<td>c</td>
<td>denotes the confidence for rule X→Y in the old database</td>
</tr>
</tbody>
</table>

Table 3.9: A Summary of Symbols

In order to keep X→Y as a rule in the new database, XY_{new}/ X_{new} must be greater than MinConfidence(tc%), that is (XY_{new}/ X_{new}) × 100%≥tc%. Now XY_{old}, X_{old} and c is known, give a tolerance t% for XY_{new} and X_{new} to make XY_{new} = XY_{old}×(1-t%), X_{new} = X_{old}×(1+t%), so XY_{new}/ X_{new} = (XY_{old}×(1-t%))/(X_{old}×(1+t%)). Since XY_{old}/
\( X_{\text{old}} = c \geq mc, c \times ((1-t\%)/(1+t\%)) \geq mc \). Thus, tolerance \( t\% \leq (c-mc)/(c+mc) \). In this case, as long as \( XY_{\text{new}} \) and \( X_{\text{new}} \) are changed within \( t\% \), rule \( X \rightarrow Y \) still can be kept a rule in the new database.

Assume \( XY \) is a large itemset in the new database and \( X \rightarrow Y \) is not a rule in the old database. Let \( XY_{\text{new}} \) denote the support for \( XY \) in the new database, \( X_{\text{new}} \) denote the support for \( X \) in the new database, \( XY_{\text{old}} \) denote the support for \( XY \) in the old database, \( X_{\text{old}} \) denote the support for \( X \) in the old database, \( c \) denote the confidence for rule \( X \rightarrow Y \) in the old database, in order to keep \( X \rightarrow Y \) as a rule in the new database, \( XY_{\text{new}} / X_{\text{new}} \) must be less than MinConfidence (mc\%), that is \( (XY_{\text{new}} / X_{\text{new}}) \times 100\% \leq mc\% \). Now \( XY_{\text{old}} \), \( X_{\text{old}} \) and \( c \) is known, give a tolerance \( t\% \) for \( XY_{\text{new}} \) and \( X_{\text{new}} \) to make \( XY_{\text{new}} = XY_{\text{old}} \times (1+t\%), X_{\text{new}} = X_{\text{old}} \times (1-t\%) \), so \( XY_{\text{new}} / X_{\text{new}} = (XY_{\text{old}} \times (1+t\%))/(X_{\text{old}} \times (1-t\%)) \).

Since \( XY_{\text{old}} / X_{\text{old}} = c \leq mc, c \times ((1+t\%)/(1-t\%)) \leq mc \). Thus, tolerance \( t\% \leq (c+mc)/(mc-c) \). In this case, as long as \( XY_{\text{new}} \) and \( X_{\text{new}} \) are changed within \( t\% \), rule \( X \rightarrow Y \) still can be kept not as a rule in the new database.

After analyzing the tolerance for the support of each low level large itemsets, we can say as long as the support of each large itemsets is changed within the tolerance, it doesn’t affect final association rules result.
Chapter 4

Maintaining Incremental Association Rules on Multi-Dimensional Database

A new algorithm (MAAP) for maintaining incremental association rules on transaction data is presented in chapter 3. In this chapter, an extension to apply MAAP algorithm for maintaining incremental association rules on multi-dimensional database is proposed. First, we will introduce multi-dimensional association rules mining and then a description on how MAAP algorithm can be applied for maintaining incremental association rules on multi-dimensional database is presented.

4.1 Introduction to Multi-Dimensional Association Rules Mining

4.1.1 Basic Concepts

Mining association rules can be applied to all kinds of data repositories such as transaction data, data warehouses [CHY96]. A data warehouse is a subject-oriented, integrated, time-variant and nonvolatile collection of data. It consolidates large amounts of data, perhaps collected from multiple sources or over long periods of time, into well-organized summarized data [I92].

OLAP (On-line Analytical Processing) is part of the data warehouse technology that enables users to examine warehouse data interactively. Typical OLAP operations include
Roll-up: to start providing the summarization at a most detailed level before gradually doing it at a highly summarized level.

Drill-down: to start providing the summarization of warehouse data at a highly summarized level before getting to the most detailed level.

Slice: to select and project on one dimension.

Dice: to select and project on two or more dimensions.

Typically the underlying data structure of a data warehouse is a multidimensional data model which is called an OLAP cube or data cube, which can be portions of data warehouses being precomputed and materialized for efficient processing. From the operational point of view, a data cube is referred to as a relational operator which computes group-by aggregations over all possible subsets of the specified dimensions [GCBL97]. Suppose there is a n-dimensional database, since there are $2^n$ combinations with n group-by dimensions, a data cube has $2^n$ data cube views (aggregate views). For example, there is a multi-dimensional database which contains three dimensions: student, course and grade, a data cube lattice framework is shown as Figure 4.1.

![Figure 4.1: Data Cube Lattice Framework](image)

In Figure 4.1, the name of any aggregate views represents the group-by dimensions. For example, SCG represents group-by three dimensions which are student(S), course(C) and
grade(G). NONE is a value which represents total aggregate measure of the data cube, in this example, the measure could be the grade point average(gpa) of a student.

4.1.2 Mining Multi-Dimensional Association Rules

Generally, multi-dimensional association rules are referred to as the association rules which are mined from multi-dimensional data warehouses. The procedure for mining multi-dimensional association rules first generates a required data cube, combines each distinct value of each dimension in the data cube into a set of items, applies Apriori algorithm to generate all large itemsets, finally, generates desired multi-dimensional association rules. In this thesis, we only focus on generating all large itemsets in a multi-dimensional database.

- Generating a required data cube

Given an association mining task involved with dimensions A₁, ..., Aₙ, we precompute and materialize the task-relevant data from data warehouse into an n-dimensional data cube. Each dimension of the data cube contains |Aᵢ|+1 values where |Aᵢ| is the number of distinct values in the dimension Aᵢ. The first |Aᵢ| rows represent the distinct values of Aᵢ. The last row is a special "any" value, in which each cell stores the aggregation value of the previous rows. Those aggregation values represent one of the essential features of the data cube structure. Each cell in these rows stores the "count" value generated from the initial relation (raw data), r(A₁=x₁, ..., Aₙ=xₙ, count). Generally, a data cube can be matched to an (n+1)-attribute table, referred to as cube table, with each attribute representing a dimension and the (n+1)-th attribute representing the count. Then, each cube cell can be matched into one tuple in the table.
Example. Suppose there is a data cube shown as Figure 4.2, which consists of three dimensions: Location, Product and Profit. The cube table corresponding to this data cube is in Table 4.1.

![Cube Diagram]

**Figure 4.2: A Cube with Three Dimensions: Location, Product and Profit**

- **Multi-Dimensional Association Rules Mining**

  Multi-dimensional association rules mining finds the association among a set of dimensions. The whole procedure is also based on the Apriori algorithm with a different support computation method. The union of each distinct value in each dimension is regarded as a set of items and this set will be an input to the Apriori algorithm. Since the items in an itemset come from different dimensions, by using the available summary layers of data cube, the support for each itemset can be
directly obtained from one cube table, making the mining procedure very efficient. The detailed algorithm is shown in Figure 4.3.

<table>
<thead>
<tr>
<th>Location</th>
<th>Product</th>
<th>Profit</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>China</td>
<td>Wagons</td>
<td>Good</td>
<td>30</td>
</tr>
<tr>
<td>China</td>
<td>Wagons</td>
<td>Excellent</td>
<td>10</td>
</tr>
<tr>
<td>China</td>
<td>Vans</td>
<td>Good</td>
<td>30</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Canada</td>
<td>Sports cars</td>
<td>Poor</td>
<td>20</td>
</tr>
<tr>
<td>Canada</td>
<td>Family cars</td>
<td>Poor</td>
<td>65</td>
</tr>
<tr>
<td>Canada</td>
<td>Family cars</td>
<td>Average</td>
<td>25</td>
</tr>
<tr>
<td>Canada</td>
<td>Trucks</td>
<td>Mediocre</td>
<td>25</td>
</tr>
<tr>
<td>Canada</td>
<td>Trucks</td>
<td>Excellent</td>
<td>20</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>USA</td>
<td>Family cars</td>
<td>Good</td>
<td>100</td>
</tr>
<tr>
<td>USA</td>
<td>Wagons</td>
<td>Poor</td>
<td>20</td>
</tr>
<tr>
<td>USA</td>
<td>Wagons</td>
<td>Good</td>
<td>40</td>
</tr>
<tr>
<td>USA</td>
<td>Vans</td>
<td>Average</td>
<td>25</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Japan</td>
<td>Family cars</td>
<td>Mediocre</td>
<td>10</td>
</tr>
<tr>
<td>Japan</td>
<td>Wagons</td>
<td>Average</td>
<td>20</td>
</tr>
<tr>
<td>Japan</td>
<td>Trucks</td>
<td>Mediocre</td>
<td>20</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Table 4.1: The Cube Table for Data Cube

Input: an n-dimensional data cube: CB[d₁,…,dₙ], minimum support (min-sup)  
Output: the set of large multi-dimensional itemsets L among the n dimensions  
K=1; L=∅;  
C₁,d₁=\{all distinct values in dimension d₁\}, C₁=∪₁ⁿ C₁,d₁; //generate candidate 1-itemset for each dimension//  
L₁=gen_large(1,C₁); //generate large 1-itemset//  
Repeat  
k=k+1;  
Cₖ=gen_candidate(k,Lₖ₋₁); //generate candidate k-itemsets//  
Lₖ=gen_large(k,Cₖ); //generate large k-itemsets//  
L=L∪Lₖ;  
Until Lₖ=∅;
The above algorithm is a main algorithm for multi-dimensional association rules generation. It first collects all distinct values from each dimension as candidate 1-itemsets, then calls function gen_large(k,C_k) (k=1) to generate large 1-itemsets L_1, after that enter into a while loop to continue generating high level candidate itemsets and large itemsets by calling function gen_candidate(k,L_{k-1}) and gen_large(k,C_k), when any level large itemsets becomes empty, the algorithm stops.

The following two algorithms are two functions which are called by above algorithm. One function used to generate large itemsets, the other used to generate candidate itemsets.

```
Function gen_large(k, C_k) // generate large k-itemset L_k from candidate C_k
L_k=∅;
for each candidate I={i_1,...,i_k} ∈ C_k do begin
    support = the count in the row(i_1,...,i_k) in the cube table
    if (support>=min_sup) then
        L_k=L_k∪{I};
end// end of for loop//
```

```
Function gen_candidate(k, L_{k-1}) // generate candidate k-itemsets C_k from large (k-1)-itemsets L_{k-1}
C_k=∅;
for each item l_1 ∈ L_{k-1} begin
    for each item l_2 ∈ L_{k-1} begin
        if (l_1 and l_2 has k-2 same items, and the other one from different dimensions) then begin
            c=l_1 ∪ l_2;
            if c has non-large (k-1)-subset then delete c
            else add c to C_k
        end // end of inner for loop//
    end // end of outer if//
end // end of outer for loop//
return C_k
```

Figure 4.3: Algorithm for Mining Multi-Dimensional Association Rules on Multi-Dimensional Database
Procedure: Based on the Apriori algorithm, the algorithm generates all large itemset iteratively. In the first iteration, it starts with finding large 1-itemsets. In the following iteration, it generates the set of large k-itemsets \( L_k \), the algorithm uses \( L_{k-1} \times L_{k-1} \) to generate the candidate k-itemsets \( C_k \). For every candidate k-itemset \( X \in C_k \), it checks the corresponding cell in the k-D cube view. According to the definition of cube, the count value saved in each cell is the aggregation from the initial database, it is exactly the support for the itemset represented by the cell. Then the algorithm compares the count value which is saved in corresponding cell with the threshold, and chooses those whose support is no less than MinSupport as large k-itemset.

Example. Based on the data cube shown in Figure 4.2. A multi-dimensional association rules mining task involve three dimensions: Location, production and profit. A concrete procedure based on algorithm is given below:

- First, collect all the distinct values in three dimensions as the initial set of items, it can also be regarded as the candidate 1-itemsets \( C_1 \) shown as Table 4.2. Then, for each item in \( C_1 \), find the count value which belongs to the corresponding row from the cube table, the count value will be the support of the item. For example, itemset “China” in Location dimension, we find the corresponding count value from the row (“China”, any, any) in cube table, suppose the count value is 82, then the support for itemset \{“China”\} is 82. Using this approach, we test each itemset in \( C_1 \) to get its support.

- According to the minimum support threshold (i.e., MinSupport=85 transactions), choose any itemset whose support is no less than MinSupport and insert them into the large 1-itemsets \( L_1 \). The results are shown in Table 4.3.

- Joining large 1-itemsets \( L_1 \) itself from different dimensions, generate candidate 2-itemsets \( C_2 \) and then doing prune step, e.g. prune all candidate 2-itemsets which have any of their subsets not belonging to \( L_1 \). The results are shown in Table 4.4.
• For each itemset in C₂, check the row from the cube table and get the count value saved in this row, namely, the support this itemset. In the same way, get the support for all 2-itemset in C₂.

• For each candidate 2-itemsets in C₂, compare its support with MinSupport, choose all 2-itemsets whose support are no less than MinSupport and insert them into large 2-itemsets. The result is shown in Table 4.5.

• Joining large 2-itemsets L₂ itself to get the candidate 3-itemsets and doing prune step. For example, by joining {USA, Family cars} and {Family cars, poor}, we get {USA, Family cars, poor}. Then, we need to check whether all its subsets belong to L₂. Since {USA, poor} is not a large 2-itemset, prune it from candidate 3-itemsets. The result is shown in Table 4.6.

• Computing the support for each itemset in C₃ by checking the cube table and getting the count value saved in the corresponding row. Compared each support with MinSupport, those 3-itemsets whose support are no less than MinSupport are inserted into large 3-itemsets. The result is shown in Table 4.7.

<table>
<thead>
<tr>
<th>Set of Items</th>
<th>Location</th>
<th>Product</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>{China}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Canada}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{USA}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Japan}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Sports cars}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Family cars}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Wagons}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Trucks}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Vans}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{poor}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{mediocre}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{average}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{good}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{excellent}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Set of Items

<table>
<thead>
<tr>
<th>Large 1-itemsets</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Canada}</td>
<td>175</td>
</tr>
<tr>
<td>{USA}</td>
<td>185</td>
</tr>
<tr>
<td>{Family cars}</td>
<td>220</td>
</tr>
<tr>
<td>{Wagons}</td>
<td>125</td>
</tr>
<tr>
<td>{Poor}</td>
<td>125</td>
</tr>
<tr>
<td>{Good}</td>
<td>220</td>
</tr>
</tbody>
</table>

Table 4.3: Large 1-itemsets
<table>
<thead>
<tr>
<th>Candidate 2-itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Canada, Family cars}</td>
</tr>
<tr>
<td>{Canada, Wagons}</td>
</tr>
<tr>
<td>{USA, Family cars}</td>
</tr>
<tr>
<td>{USA, Wagons}</td>
</tr>
</tbody>
</table>

Table 4.4: Candidate 2-itemsets

<table>
<thead>
<tr>
<th>Large 2-itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>{USA, Family cars} support=100</td>
</tr>
<tr>
<td>{Canada, poor} support=105</td>
</tr>
<tr>
<td>{USA, good} support=140</td>
</tr>
<tr>
<td>{Family cars, poor} support=85</td>
</tr>
<tr>
<td>{Family cars, good} support=100</td>
</tr>
</tbody>
</table>

Table 4.5: Large 2-itemsets

<table>
<thead>
<tr>
<th>Candidate 3-itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>{USA, Family cars, good}</td>
</tr>
</tbody>
</table>

Table 4.6: Candidate 3-itemsets

<table>
<thead>
<tr>
<th>Large 3-itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>{USA, Family cars, good}</td>
</tr>
</tbody>
</table>

Table 4.7: Large 3-itemsets
4.2 Maintaining Incremental Multi-Dimensional Association Rules on Multi-Dimensional Database

4.2.1 The Procedure for Maintaining Incremental Multi-Dimensional Association Rules on Multi-Dimensional Database

In this section, the method for Maintaining Incremental Multi-dimensional Association Rules on Multi-Dimensional Database will be propose. The basic idea for this method is that it applies MAAP algorithm which has been presented in chapter 3, by using Apriori property, generates all large itemsets from the highest level old large itemsets.

Let us show an example to see the concrete procedure. Suppose we still use the example and its result which are shown in section 4.1.2. In this case, the input for MAAP algorithm will be large itemsets in the old multi-dimensional database.

Large 1-itemsets contains \{\{Canada\}, \{USA\}, \{Family cars\}, \{Wagons\}, \{poor\}, \{good\}\}.
Large 2-itemsets contains \{\{USA, Family cars\}, \{Canada, poor\}, \{USA, good\}, \{Family cars, poor\}, \{Family cars, good\}\}.
Large 3-itemsets contains \{\{USA, Family cars, good\}\}.
Small 1-itemsets contains \{\{China\}, \{Japan\}, \{Sport cars\}, \{Trucks\}, \{Vans\}, \{mediocre\}, \{average\}, \{excellent\}\}.
Small 2-itemsets contains \{\{Canada, Family cars\}, \{Canada, Wagons\}, \{USA, Wagons\}, \{Canada, good\}, \{USA, poor\}, \{Family cars, poor\}, \{Family cars, good\}\}.
The old highest level k=3.

Now, assume the data warehouse has been changed, thus, the corresponding data cube which is involved three dimensions (Location, Product and Profit) is also changed. The inserted cube table and deleted cube table are given in Table 4.8 and Table 4.9.
<table>
<thead>
<tr>
<th>Location</th>
<th>Product</th>
<th>Profit</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>China</td>
<td>Wagons</td>
<td>Good</td>
<td>10</td>
</tr>
<tr>
<td>China</td>
<td>Wagons</td>
<td>Excellent</td>
<td>5</td>
</tr>
<tr>
<td>Canada</td>
<td>Trucks</td>
<td>Mediocre</td>
<td>12</td>
</tr>
<tr>
<td>Canada</td>
<td>Trucks</td>
<td>Excellent</td>
<td>15</td>
</tr>
<tr>
<td>USA</td>
<td>Family cars</td>
<td>Good</td>
<td>35</td>
</tr>
<tr>
<td>USA</td>
<td>Wagons</td>
<td>Poor</td>
<td>20</td>
</tr>
<tr>
<td>Japan</td>
<td>Wagons</td>
<td>Average</td>
<td>6</td>
</tr>
<tr>
<td>Japan</td>
<td>Trucks</td>
<td>Mediocre</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 4.8: Inserted Cube Table

<table>
<thead>
<tr>
<th>Location</th>
<th>Product</th>
<th>Profit</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>China</td>
<td>Wagons</td>
<td>Excellent</td>
<td>8</td>
</tr>
<tr>
<td>China</td>
<td>Vans</td>
<td>Good</td>
<td>10</td>
</tr>
<tr>
<td>Canada</td>
<td>Sports cars</td>
<td>Mediocre</td>
<td>20</td>
</tr>
<tr>
<td>Canada</td>
<td>Family cars</td>
<td>Excellent</td>
<td>20</td>
</tr>
<tr>
<td>USA</td>
<td>Wagons</td>
<td>Poor</td>
<td>25</td>
</tr>
<tr>
<td>USA</td>
<td>Wagons</td>
<td>Good</td>
<td>15</td>
</tr>
<tr>
<td>USA</td>
<td>Vans</td>
<td>Average</td>
<td>10</td>
</tr>
<tr>
<td>Japan</td>
<td>Family cars</td>
<td>Good</td>
<td>10</td>
</tr>
<tr>
<td>Japan</td>
<td>Wagons</td>
<td>Average</td>
<td>20</td>
</tr>
<tr>
<td>Japan</td>
<td>Trucks</td>
<td>Mediocre</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 4.9: Deleted Cube Table

- First, test each itemset in old large itemsets $L_1$, e.g. itemset \{USA, Family cars, good\}, scan the inserted cube table and the deleted cube table to get the
corresponding count value 35, 0, then compute its new support in the new data warehouse, that is 100+35-0=135. Since its new support is no less than min_sup, include this itemset in new large 3-itemsets \( L_3 \)=\{USA, Family cars, good\}. In the same way, test next itemset in \( L_3 \).

- Applying Apriori property, simply include \{USA, Family cars\}, \{USA, good\}, \{Family cars, good\} in \( L_2 \); include \{USA\}, \{Family cars\}, \{good\} in \( L_1 \).

- From the first level 1-itemsets, start with searching the additional large itemsets, e.g. suppose \( i=1 \), for each itemset in \( L_1 \)-\( L_1 \), scan the inserted cube table and deleted cube table to get the count value from the corresponding row and then compute its new support in the updated data warehouse. Choosing all itemsets whose support are no less than MinSupport, insert them into \( L_1 \); for each itemset in \( C_1 \)-\( L_1 \), scan the inserted cube table and deleted cube table to get the count value from the corresponding row and then compute its new support in the updated data warehouse. Choosing all itemsets whose support are no less than MinSupport, insert them into \( L_1 \).

- Joining \( L_1 \times L_1 \) to get more additional candidate 2-itemset \( C_2 \).

- Increment \( i \), go to step 2 and begin next iteration until \( i=k \).

- Adjusting the highest level \( k \) by joining \( L_k \times L_k \) until \( L_k = \emptyset \).

The result for whole procedure is shown as Figure 4.4.

```
<table>
<thead>
<tr>
<th>Set of Items</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>{China}</td>
<td></td>
<td>{mean}</td>
</tr>
<tr>
<td>{Canada}</td>
<td></td>
<td>{poor}</td>
</tr>
<tr>
<td>{USA}</td>
<td></td>
<td>{mediocre}</td>
</tr>
<tr>
<td>{Japan}</td>
<td></td>
<td>{average}</td>
</tr>
<tr>
<td>{Sports cars}</td>
<td></td>
<td>{good}</td>
</tr>
<tr>
<td>{Family cars}</td>
<td></td>
<td>{excellent}</td>
</tr>
<tr>
<td>{Wagons}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Trucks}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Vans}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
### Large 1-itemsets

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canada</td>
<td>175</td>
</tr>
<tr>
<td>USA</td>
<td>185</td>
</tr>
<tr>
<td>Family cars</td>
<td>220</td>
</tr>
<tr>
<td>Wagons</td>
<td>125</td>
</tr>
<tr>
<td>Trucks</td>
<td>100</td>
</tr>
<tr>
<td>Poor</td>
<td>125</td>
</tr>
<tr>
<td>Good</td>
<td>220</td>
</tr>
<tr>
<td>Excellent</td>
<td>88</td>
</tr>
</tbody>
</table>

### Candidate 2-itemsets

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Canada, Family cars}</td>
<td></td>
</tr>
<tr>
<td>{Canada, Wagons}</td>
<td></td>
</tr>
<tr>
<td>{Canada, Trucks}</td>
<td></td>
</tr>
<tr>
<td>{USA, Family cars}</td>
<td></td>
</tr>
<tr>
<td>{USA, Wagons}</td>
<td></td>
</tr>
<tr>
<td>{USA, Trucks}</td>
<td></td>
</tr>
<tr>
<td>{Canada, Poor}</td>
<td></td>
</tr>
<tr>
<td>{Canada, Good}</td>
<td></td>
</tr>
<tr>
<td>{Canada, Excellent}</td>
<td></td>
</tr>
<tr>
<td>{USA, Poor}</td>
<td></td>
</tr>
<tr>
<td>{USA, Good}</td>
<td></td>
</tr>
<tr>
<td>{USA, Excellent}</td>
<td></td>
</tr>
<tr>
<td>{Family cars, Poor}</td>
<td></td>
</tr>
<tr>
<td>{Family cars, Good}</td>
<td></td>
</tr>
<tr>
<td>{Family cars, Excellent}</td>
<td></td>
</tr>
<tr>
<td>{Wagons, Poor}</td>
<td></td>
</tr>
<tr>
<td>{Wagons, Good}</td>
<td></td>
</tr>
<tr>
<td>{Wagons, Excellent}</td>
<td></td>
</tr>
<tr>
<td>{Trucks, Poor}</td>
<td></td>
</tr>
<tr>
<td>{Trucks, Good}</td>
<td></td>
</tr>
<tr>
<td>{Trucks, Excellent}</td>
<td></td>
</tr>
<tr>
<td>{USA, Family cars}</td>
<td>100</td>
</tr>
<tr>
<td>{USA, Wagons}</td>
<td>85</td>
</tr>
<tr>
<td>{Canada, poor}</td>
<td>105</td>
</tr>
<tr>
<td>{USA, good}</td>
<td>140</td>
</tr>
<tr>
<td>{Family cars, poor}</td>
<td>85</td>
</tr>
<tr>
<td>{Family cars, good}</td>
<td>100</td>
</tr>
</tbody>
</table>

### Candidate 3-itemsets

<table>
<thead>
<tr>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>{USA, Family cars, good}</td>
</tr>
</tbody>
</table>
Large 3-itemsets

\{USA, Family cars, good\}

Figure 4.4 The Result of An Example for Mining Association Rules on Multi-Dimensional Database

4.2.2 The Algorithm for Maintaining Incremental Multi-dimensional Association Rules on Multi-Dimensional Database

Algorithm MAAP() // Maintaining Multi-Dimensional Association Rules on Multi-Dimensional Database/
Input: set of old large itemsets (L_1), old highest level (k), a cube table involved mining task, inserted cube table and deleted cube table in the updated data warehouse
Output: New large itemsets (L_1') in the new database, new highest level k
Begin
// generate new large itemsets from old large itemsets at level k //
oldk = k; // save highest level //
L_k =\{\};
while k >= 1 begin //
for all itemsets x in L_k begin
if x is large in the updated database then begin
generate all non empty subsets of x;
add x to L_k;
add all subsets of x to appropriate L_{k-1}, \ldots L_1;
end //end of if\//
if L_k =\{\} then k = k-1 else exit;
end //end of while//
// generate new itemsets from old large items not yet in L_i' //
i=1;
while (i <= oldk) do begin
X_i = L_k - L_i; //compare L_i to L_k
for all itemsets x in X_i begin
if x is large itemset in the new database then
add x to L_i
else begin
delete supersets of x from high level large small itemsets;
end
for all itemsets \( x \) in \( Q_i (Q_i = C_i - L_i) \) begin
  if \( x \) is large itemset in the new database then
    add \( x \) to \( L_i \);
  end //end of while/
newk = i - 1;
//handle the case when new database has a higher level //\nwhile() do begin
  \( C_{newk+1} = \text{Apriori\_gen}(L_{newk}) \) /join \( L_i \times L_i \) generate more possible large
  itemsets//
  newk = newk + 1;
  if \( C_{newk} = \emptyset \) exit // while loop //;
  end //end of while/
L = \( \cup L_k \);
end // of MAAP //

Compute \( L_i \times L_i \);

{Insert into \( C_{i+1} \)
Select \( p \_item_1, p \_item_2, \ldots, p \_item_n, q \_item_i \);
From \( L \ p, L \ q \)
Where \( p \) and \( q \) has \( i-1 \) same items, and the other one from different dimensions;
Prune step: delete all itemsets \( \in S_{i+1} \) such that some \( i \)-subset of \( c \) is not in \( S_i \)}

Figure 4.5: An Algorithm for Maintaining Incremental Multi-dimensional
Association Rules on Multi-Dimensional Database
Chapter 5

Performance Analysis

In this chapter, some experiment results conducted to analyze the performance of the proposed MAAP algorithm with previous incremental maintenance association rules algorithms (FUP2 and Apriori) are presented. We first discuss a theoretical analysis in section 5.1. Then an introduction of two case experiments for evaluating the performance and benefit for MAAP algorithm is presented in section 5.2. Finally, two case experimental results are presented in section 5.3.

5.1 Theoretical Analysis

Comparing MAAP algorithm with FUP2 algorithm [CLK97], the one important benefit of MAAP algorithm is that it avoids computing some parts of large itemsets from the database, especially, generating $L_1$ and $L_2$ usually costs much time in mining association rules because $L_1$ and $L_2$ may contain the longest lists of candidate itemsets for which the database is scanned to compute their supports. If many low level large itemsets which are large itemsets in the new database can be generated from the first step of MAAP algorithm, there is some performance gain in terms of reduced computation time achieved by using the MAAP algorithm. A formula can be used to illustrate this point.

Assume highest large itemsets level in the old database is $k$, applying MAAP algorithm and getting a large itemset in the highest level $k$ is still large itemset in the new database, a total number of itemset computation can be saved, which is $C_k^1 + C_k^2 + ... + C_k^{k-1}$. 
From this formula, we see that if highest level is more higher and some large itemsets can be obtained from the highest level, many low level large itemset computation could be avoided.

Let us analyze the change of large itemsets in the new database and estimate when MAAP algorithm could have better performance than FUP2.

Further detail regarding the change situation is obtained, which means that change could have happened in the old association rules, change also could have happened in the new association rules or both of them. Furthermore, we found FUP2 algorithm cannot obtain good performance in some change situations. For example, when old large itemsets did not change a lot in new database and some new large itemsets appear in the new database, in this case the total large itemsets in the new database is changed a lot and FUP2 algorithm will be applied. In this case, the effort of computing all old large itemsets is almost wasted. An experiment is conducted to find a range, within this range, MAAP algorithm has better performance than FUP2. Some different situations among amount of change for large itemsets in the new database are shown in Figure 5.1.

![Figure 5.1: Different Situations among Large Itemsets Changes in the New Database](image)

From Figure 5.1, we can estimate for the last two situations, FUP2 have poor performance since many repetitive work will be involved.
5.2 Two Case Experiments

In order to evaluate the performance and benefit for MAAP algorithm, we conduct two different experiments.

The experiments were performed on a Pentium II 300 with 64MB of main memory and running UNIX solaris 7, the data resided in the UNIX solaris file system and was stored on an IDE 8.4GB hard drive.

Experiment 1: Given a fixed size of dataset (inserted part dataset and deleted part dataset are also fixed), we test CPU execution time in different thresholds of support among MAAP, FUP2 and Apriori algorithm. The aim of this experiment is that it shows the performance of MAAP algorithm is better than the one of FUP2 and Apriori algorithm in different support under the same dataset.

Experiment 2: Given a fixed size of dataset (include inserted dataset and deleted dataset) and a fixed support, we test CPU execution time when varying number of large itemsets in the old database are changed. Since the number of large itemsets changed may affect CPU time of MAAP algorithm, we conduct this experiment to observe the performance between MAAP and FUP2 algorithm.

In order to implement these experiments, we first generate a fixed size of dataset. This dataset is generated in terms of total number of items, total number of transactions, average size of items in each transactions, average size of the maximal potentially large itemsets (a potentially large itemset is a itemset which a customer usually buys them together), number of maximal potentially large itemsets. A summary of these parameters are shown in Table 5.1.
<table>
<thead>
<tr>
<th>D</th>
<th>NUMBER OF TRANSACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>AVERAGE SIZE OF THE TRANSACTIONS</td>
</tr>
<tr>
<td>L</td>
<td>AVERAGE SIZE OF THE MAXIMAL POTENTIALLY LARGE ITEMSETS</td>
</tr>
<tr>
<td>L</td>
<td>NUMBER OF MAXIMAL POTENTIALLY LARGE ITEMSETS</td>
</tr>
<tr>
<td>N</td>
<td>NUMBER OF ITEMS</td>
</tr>
</tbody>
</table>

Table 5.1: Parameters

In our experiments, the parameters chosen for dataset is the number of transactions (one hundred thousand records) $|D|=100000$, average size of the transactions (each transaction contains five items) $|T|=5$, average size of the maximal potentially large itemsets (average size of the maximal potentially large itemsets is two items) $|L|=2$, number of maximal potentially large itemsets (the total number of maximal potentially large itemsets is two thousand) $|L|_{total}=2000$, number of items (the total number of items is one thousand) $N=1000$.

5.3 Experiment Evaluation

Experiment 1 Given a dataset, the parameters chosen for this dataset is the number of transactions (one hundred thousand records) $|D|=100000$ records, average size of the transactions (each transaction contains five items) $|T|=5$, average size of the maximal potentially large itemsets (average size of the maximal potentially large itemsets is two items) $|L|=2$, number of maximal potentially large itemsets (the total number of maximal potentially large itemsets is two thousand) $|L|_{total}=2000$, number of items (the total number of items is one thousand) $N=1000$, assume the size of updated (inserted) dataset is 15000 records, the size of updated (deleted) dataset is 15000 records (these parameters are abbreviated as T5.I2.D100K-15K+15K), the support thresholds is varied between 1.4 and
1.9, the CPU time under above condition is tested. An experimental data is shown in Table 5.2.

**Dataset: T5.I2.D100K-15K+15K**

<table>
<thead>
<tr>
<th>Maintenance Algorithm</th>
<th>1.4</th>
<th>1.5</th>
<th>1.6</th>
<th>1.7</th>
<th>1.8</th>
<th>1.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apriori</td>
<td>13451.99</td>
<td>3716.02</td>
<td>2041.77</td>
<td>1268.18</td>
<td>856.88</td>
<td>625.77</td>
</tr>
<tr>
<td>FUP2</td>
<td>7807.21</td>
<td>2208.75</td>
<td>954.10</td>
<td>596.04</td>
<td>408.66</td>
<td>312.89</td>
</tr>
<tr>
<td>MAAP</td>
<td>4955.29</td>
<td>1608.67</td>
<td>700.98</td>
<td>436.68</td>
<td>302.82</td>
<td>235.61</td>
</tr>
</tbody>
</table>

Table 5.2: Maintenance Time for Different Maintenance Algorithm in Different Support Threshold

Table 5.2 shows CPU execution time for each maintenance algorithm in different support threshold within the same dataset.

These experimental data is plotted in Figure 5.2.

**T5.I2.D100K-15K+15K**

![Figure 5.2: Maintenance Time for Different Maintenance Algorithm in Different Support Threshold](image-url)
**Analysis** From Figure 5.2, we can see (i) as the size of support increases, the execution time of all the algorithms (Apriori, FUP2 and MAAP) decrease because of decreasing in the total number of candidate and large itemsets in each iteration. (ii) for the same support, the execution time of MAAP algorithm is less than the one of the FUP2 and Apriori algorithm. (iii) as the size of support increases, the difference of execution time between MAAP algorithm and FUP2 decreases, the reason is that if the support increases, the total number of large itemsets and candidate itemsets will decrease, in this situation there are no more itemsets needed to be scan for their support.

The benefit gain of CPU execution time (CPU\textsubscript{time}) among MAAP, FUP2 and Apriori algorithm is shown in Table 5.3 and Table 5.4.

<table>
<thead>
<tr>
<th>Support Threshold</th>
<th>FUP2 Algorithm CPU\textsubscript{time} (T_f) (seconds)</th>
<th>MAAP Algorithm CPU\textsubscript{time} (T_m) (seconds)</th>
<th>Benefit Gain (T_f - T_m) (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>7807.21</td>
<td>5738.3</td>
<td>2068.91</td>
</tr>
<tr>
<td>1.5</td>
<td>2208.75</td>
<td>1828.89</td>
<td>379.86</td>
</tr>
<tr>
<td>1.6</td>
<td>954.1</td>
<td>834.84</td>
<td>119.26</td>
</tr>
<tr>
<td>1.7</td>
<td>596.04</td>
<td>538.82</td>
<td>57.22</td>
</tr>
<tr>
<td>1.8</td>
<td>408.66</td>
<td>378.83</td>
<td>29.83</td>
</tr>
<tr>
<td>1.9</td>
<td>312.89</td>
<td>292.87</td>
<td>20.02</td>
</tr>
</tbody>
</table>

Table 5.3: The Benefit Gain of CPU Execution Time between MAAP and FUP2 Algorithm
<table>
<thead>
<tr>
<th>Support Threshold</th>
<th>Apriori Algorithm CPU(_{\text{time}}) T(_a)(seconds)</th>
<th>MAAP Algorithm CPU(_{\text{time}}) T(_m)(seconds)</th>
<th>Benefit Gain T(_a) - T(_m)(seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>13451.99</td>
<td>5738.3</td>
<td>7713.69</td>
</tr>
<tr>
<td>1.5</td>
<td>3716.02</td>
<td>1828.89</td>
<td>1887.13</td>
</tr>
<tr>
<td>1.6</td>
<td>2041.77</td>
<td>834.84</td>
<td>1206.93</td>
</tr>
<tr>
<td>1.7</td>
<td>1268.18</td>
<td>538.82</td>
<td>729.36</td>
</tr>
<tr>
<td>1.8</td>
<td>856.88</td>
<td>378.83</td>
<td>478.05</td>
</tr>
<tr>
<td>1.9</td>
<td>625.77</td>
<td>292.87</td>
<td>332.9</td>
</tr>
</tbody>
</table>

Table 5.4: The Benefit Gain of CPU Execution Time between MAAP and Apriori Algorithm

The improvement ratio compared MAAP with FUP2 and Apriori algorithm is shown in Table 5.5 and Table 5.6.

<table>
<thead>
<tr>
<th>Support Threshold</th>
<th>FUP2 Algorithm CPU(_{\text{time}}) T(_a)(seconds)</th>
<th>MAAP Algorithm CPU(_{\text{time}}) T(_m)(seconds)</th>
<th>Improvement Ratio (T(_a) - T(_m))/T(_a)×100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>7807.21</td>
<td>5738.3</td>
<td>26.5</td>
</tr>
<tr>
<td>1.5</td>
<td>2208.75</td>
<td>1828.89</td>
<td>17.2</td>
</tr>
<tr>
<td>1.6</td>
<td>954.1</td>
<td>834.84</td>
<td>12.5</td>
</tr>
<tr>
<td>1.7</td>
<td>596.04</td>
<td>538.82</td>
<td>9.6</td>
</tr>
<tr>
<td>1.8</td>
<td>408.66</td>
<td>378.83</td>
<td>7.3</td>
</tr>
<tr>
<td>1.9</td>
<td>312.89</td>
<td>292.87</td>
<td>6.4</td>
</tr>
</tbody>
</table>

Table 5.5: The Improvement Ratio of CPU Execution Time between MAAP and FUP2 Algorithm
<table>
<thead>
<tr>
<th>Support Threshold</th>
<th>Apriori Algorithm CPU time $T_a$(seconds)</th>
<th>MAAP Algorithm CPU time $T_m$(seconds)</th>
<th>Improvement Ratio $(T_a - T_m)/T_a \times 100%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>13451.99</td>
<td>5738.3</td>
<td>57.3</td>
</tr>
<tr>
<td>1.5</td>
<td>3716.02</td>
<td>1828.89</td>
<td>50.8</td>
</tr>
<tr>
<td>1.6</td>
<td>2041.77</td>
<td>834.84</td>
<td>59.1</td>
</tr>
<tr>
<td>1.7</td>
<td>1268.18</td>
<td>538.82</td>
<td>56.5</td>
</tr>
<tr>
<td>1.8</td>
<td>856.88</td>
<td>378.83</td>
<td>55.8</td>
</tr>
<tr>
<td>1.9</td>
<td>625.77</td>
<td>292.87</td>
<td>53.1</td>
</tr>
</tbody>
</table>

Table 5.6: The Improvement Ratio of CPU Execution Time between MAAP and Apriori Algorithm

**Conclusion** From Table 5.5 and Table 5.6, we can see MAAP algorithm is faster than FUP2 and Apriori algorithm in maintaining association rules when given a fixed size of dataset.

**Experiment 2** Given a dataset, the parameters chosen for this dataset is the number of transactions (one hundred thousand records) $|D|=10000$ records, average size of the transactions (each transaction contains five items) $|T|=5$, average size of the maximal potentially large itemsets itemsets (average size of the maximal potentially large itemsets is two items) $|I|=2$, number of maximal potentially large itemsets (the total number of maximal potentially large itemsets is two thousand) $|L|=2000$, number of items (the total number of items is one thousand) $N=1000$, assume the support is 1.4, the size of updated (inserted) database is 1.5K records, the size of updated (deleted) database is 1.5K records (these parameters are abbreviated as T5.I2.D100K-1.5K+1.5K), $x$ represents the percentage changed of old large itemsets in the new database and $x$ is varied between 10% and 50%. The CPU execution time under above condition is tested. An experimental data is shown in Table 5.7.
Dataset: T5.I2.D10K-1.5K+1.5K

<table>
<thead>
<tr>
<th>CPU Execution Time (seconds)</th>
<th>The Percentage Change of Old Large Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintenance Algorithm</td>
<td></td>
</tr>
<tr>
<td>FUP2</td>
<td>1426.26</td>
</tr>
<tr>
<td>MAAP</td>
<td>1059.71</td>
</tr>
<tr>
<td></td>
<td>1567.33</td>
</tr>
<tr>
<td></td>
<td>1623.52</td>
</tr>
<tr>
<td></td>
<td>1765.59</td>
</tr>
<tr>
<td></td>
<td>1348.54</td>
</tr>
<tr>
<td></td>
<td>1546.93</td>
</tr>
<tr>
<td></td>
<td>1837.32</td>
</tr>
</tbody>
</table>

Table 5.7: CPU Execution Time Between MAAP and FUP2 in Different Percentage Change of Old Large Itemsets

These experimental data is plotted in Figure 5.3.

Figure 5.3: Maintenance Time for FUP2 and MAAP Algorithm in Different Percent Change in Old Large Itemsets
Analysis from Figure 5.3, we can see (i) as the change of old large itemsets increases, the CPU execution time of MAAP and FUP2 algorithms increase because of increasing in the total number of large itemsets needed to be computed. (ii) as the change of old large itemsets increases, the different of CPU execution time between MAAP and FUP2 algorithms decreases, when the change of old large itemsets is about 45%, CPU execution times for MAAP algorithm is higher than the one of FUP2 algorithm. The reason is that if the percentage change of old large itemsets is about 45%, MAAP algorithm tested many large itemsets, these large itemsets are no longer large in the new database, thus, cost much CPU computation time to get their support.

The benefit gain of CPU execution time (CPU\textsubscript{time}) between MAAP and FUP2 algorithm is shown in Table 5.8.

<table>
<thead>
<tr>
<th>Change Percent</th>
<th>FUP2 Algorithm CPU\textsubscript{time} (T_f) (seconds)</th>
<th>MAAP Algorithm CPU\textsubscript{time} (T_m) (seconds)</th>
<th>Benefit Gain (T_f - T_m) (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.3</td>
<td>1426.26</td>
<td>1059.71</td>
<td>366.55</td>
</tr>
<tr>
<td>25.1</td>
<td>1567.33</td>
<td>1348.54</td>
<td>218.79</td>
</tr>
<tr>
<td>37.8</td>
<td>1623.52</td>
<td>1541.93</td>
<td>81.59</td>
</tr>
<tr>
<td>49.6</td>
<td>1765.59</td>
<td>1837.32</td>
<td>-71.32</td>
</tr>
</tbody>
</table>

Table 5.8: The Benefit Gain of CPU Execution Time between MAAP and FUP2 Algorithm

Conclusion From the observation of Table 5.8, if the percent of change in the old large itemsets is no great than 45%, MAAP algorithm has better than FUP2 algorithm. When the percent of change in the old large itemsets is more than 45%, the performance of MAAP algorithm is down.
Chapter 6

Future Research and Conclusions

6.1 Conclusions

This thesis presents a new algorithm (MAAP) for maintaining discovered association rules in the updated database. This algorithm applies an Apriori property to the set of large itemsets in the old database, generates some parts of the lower level large itemsets in the new database using all previous old large itemsets that are confirmed to be still large in the new database. Thus, it eliminates the need to compute parts of lower level large itemsets and saves rule maintenance time by reducing the number of itemsets scanned. It achieves more benefit when high level large itemsets can be used to generate a lot of low level large itemsets in the first step of applying the Apriori property.

Two experimental case studies are conducted to analysis the performance and the benefit of the proposed MAAP algorithm. The experimental results demonstrate MAAP algorithm has better performance than FUP2 algorithm in some situations. It improves and reduces maintenance time in maintaining incremental association rules. So, in some situations (when the percent change of old large itemsets is less than around 45%), MAAP algorithm is a better algorithm for updating association rules.
6.2 Future work

The following aspects may be considered for future work.

- At what kth highest level it is most beneficial to start applying this technique. The highest level that yields most low level large itemsets when the Apriori property is applied is the needed level. For example, if the total number of large itemsets in \( L_3 \) is 5, and the total number of large itemsets in \( L_4 \) is 1, comparing numbers 3,5 with 4, 1, may tell us to begin from \( L_3 \), since \( L_3 \) may cover more lower level large itemsets than \( L_4 \).

- An experiment could be conducted to apply MAAP algorithm for maintaining incremental multi-dimensional rules in multi-dimensional databases.

- For mining multi-dimensional association rules, materializing which aggregate views for a data cube involved mining task should be considered in order to save response time and system resources.
Bibliography


Appendix A: Condensed Apriori Algorithm

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define SUPPORT 1.5
#define NUMTRANS 100000
#define FILENAME "data1.5"
#define ITEMSETLEN 5
#define ITEMSETSLEN 1000
#define CITEMSETSLEN 10000
#define COUTNUM 10000

char *change_to_str(int);
int prune(struct *,int,int);

struct large1
{
  char item[ITEMSETLEN];
  int sup;
}L1[ITEMSETSLEN];

struct candidate2
{
  char item1[ITEMSETLEN];
  char item2[ITEMSETLEN];
}C2[CITEMSETSLEN];

struct large2
{
  char item1[ITEMSETLEN];
  char item2[ITEMSETLEN];
  int sup;
}L2[CITEMSETSLEN];

struct temp
{
  char item1[ITEMSETLEN];
  char item2[ITEMSETLEN];
  char item3[ITEMSETLEN];
  char item4[ITEMSETLEN];
  char item5[ITEMSETLEN];
  char item6[ITEMSETLEN];
  char item7[ITEMSETLEN];
  char item8[ITEMSETLEN];
  char item9[ITEMSETLEN];
  char item10[ITEMSETLEN];
};

main()
{
  FILE *fpt;
  char str[100],*str1;
  int count[COUTNUM];
  char *s1,*s2,*s3;
  int f1,f2,f3;
  int i,j,x,n,m,nu1,k1,k2,k3,k4,k5,m,n,n1,n2,n3,count_num=0;
  clock_t t;
}
t=clock();

/* initial count array */
for(i=0;i<COUNTNUM;i++)
    count[i]=0;

/* scan 1-item */

fpt=fopen(FILENAME,"r");
while(fgets(str100,fpt)!=NULL)
{
    s1=strtok(str1,"");
    s2=strtok(NULL," ");
    s3=strtok(NULL," ");
    x=atoi(s3);
    count[x]=count[x]+1;
}
fclose(fpt);

/* save 1-itemset into large 1-itemset */

for(i=0;i<1000;i++)
    if(count[i]/(float)NUMTRANS>=SUPPORT/100)
    {
        str1=change_co_str[i];
        strcpy(LL[k].item,str1);
        LL[k].sup=count[i];
        printf("%s %d\n",LL[k].item,LL[k].sup);
        k++;
    }

/* generating candidate 2-itemsets and save some into large 2-itemsets */

for(i=0;i<COUNTNUM;i++)
    count[i]=0;

for(i=0;i<k;i++)
    for(j=i+1;j<k;j++)
    {
        fpt=fopen(FILENAME,"r");
        while(fgets(str100,fpt)!=NULL)
        {
            s1=strtok(str1," ");
            s2=strtok(NULL," ");
            s3=strtok(NULL," ");
            num=atoi(s3);
            if(num==f)
            {
                f=num;
                fl=0;
            }
            else
            {
                if(strcmp(s3,LL[i].item,strlen(s3)-1)==0)
                    fl=1;
                if(strcmp(s3,LL[j].item,strlen(s3)-1)==0&&(!fl))
                    count[m]=count[m]+1;
            }/*end of while */
        fclose(fpt);
    }
    if(count[m]/(float)NUMTRANS>=SUPPORT/100)
strcpy(L2[k1].item1,L1[i].item);
strcpy(L2[k1].item2,L1[j].item);
L2[k1].supi=COUNT[i];
printf("%s %s %d \n",L2[k1].item1,L2[k1].item2,L2[k1].sup);
k1++;
}
N++;
} /* end of j loop*/

/* generating candidate 3-itemsets and save some into large 3-itemsets */

for(i=0;i<COUNTNUM;i++)
count[i]=0;

n=0;
for(i=0;i<k1;i++)
for(j=i+1;j<k1;j++)
if((strcmp(L2[i].item1,L2[j].item1)==0)&&
   (strcmp(L2[i].item2,L2[j].item2)<0))
{
    strcpy(temp1.item1,L2[i].item1);
    strcpy(temp1.item2,L2[i].item2);
    strcpy(temp1.item3,L2[i].item3);
    if(prune(&temp1,k1,1)==1)
    {
      strcpy(C3[n].item1,L2[i].item1);
      strcpy(C3[n].item2,L2[i].item2);
      strcpy(C3[n].item3,L2[i].item3);
      n++;
    }
  }
  f=1;
  f1=0;
  m=0;
for(i=0;i<n;i++)
{
  fpt=fopen(FILENAME,"r");
  while(fgets(str,100,fpt) != NULL)
  {
     s1=strtok(str," ");
     s2=strtok(NULL," ");
     s3=strtok(NULL," ");
     num1=atoi(s2);
     
     if(num1==f)
     {
      f=num1;
      f1=0;
      f2=0;
      }
      if(strcmp(s3,C3[i].item1.strlen(s3)-1)==0)
      f1=1;
      if((strcmp(s3,C3[i].item2.strlen(s3)-1)==0)&&(f1==1))
      f2=1;
      if((strcmp(s3,C3[i].item3.strlen(s3)-1)==0)&&(f1==1)&&(f2==1))
      count[m]=count[m]+1;
    } /* end of while */
  fclose(fpt);
  if(count[m]/(float)NUMTRANS>=SUPPORT/100)
  {
    strcpy(L3[k2].item1,C3[i].item1);
  }
strcpy(L3[k2].item1,C3[i].item2);
strcpy(L3[k2].item2,C3[i].item3);
L3[k2].sup=count2[m];
printf("%s %s %d\n",L3[k2].item1,L3[k2].item2,
L3[k2].item3,L3[k2].sup);
k2++;
} 
m++;
}

/* generating candidate 4-itemsets */
n1=0;
for(i=0;i<k2;i++)
for(j=i+1;j<k2;j++)
if((strcmp(L3[i].item1,L3[j].item1)==0)&&
   (strcmp(L3[i].item2,L3[j].item2)==0)&&
   (strcmp(L3[i].item3,L3[j].item3)<0))
{
    strcpy(temp.item1,L3[i].item1);
    strcpy(temp.item2,L3[i].item2);
    strcpy(temp.item3,L3[i].item3);
    strcpy(temp.item4,L3[j].item3);
    if(prune(&temp,k2,4)==1)
    {
      strcpy(C4[n1].item1,L3[i].item1);
      strcpy(C4[n1].item2,L3[i].item2);
      strcpy(C4[n1].item3,L3[i].item3);
      strcpy(C4[n1].item4,L3[j].item3);
      n1++;
    }
  }

f=1;
f1=0;
f2=0;
m=0;
for(i=0;i<n1;i++)
{
  fpt=fopen(FILENAME,"r");
  while(fgets(str,100,fpt)!=NULL)
  {
    s1=strtok(str," ");
    s2=strtok(NULL," ");
    s3=strtok(NULL," ");
    num1=atoi(s2);
    if(num1>f)
    {
      f=num1;
      f1=0;
      f2=0;
      f3=0;
    }
    if(strcmp(s3,C4[i].item1,strlen(s3)-1)==0)
    f1=1;
    if(strcmp(s3,C4[i].item2,strlen(s3)-1)==0)&&(f1==1)
    f2=1;
    if(strcmp(s3,C4[i].item3,strlen(s3)-1)==0)&&(f2==1)
    f3=1;
    if(strcmp(s3,C4[i].item4,strlen(s3)-1)==0)&&(f3==1)
    count[m]=count[m]+1;
fclose(fpc);

if(count[a]/(float)NUMTRANS>=SUPPORT/100)
{
  strcpy(L4[k3].item1.C4[i].item1);
  strcpy(L4[k3].item2.C4[i].item2);
  strcpy(L4[k3].item3.C4[i].item3);
  strcpy(L4[k3].item4.C4[i].item4);
  L4[k3].sup=count[a];
  printf("%s %s %s %d\n",L4[k3].item1,L4[k3].item2,
    L4[k3].item3,L4[k3].item4,L4[k3].sup);
  k3++;
}
  m++;

...

printf("%.2fs\n",(clock()-t)/(double)CLOCKS_PER_SEC);
}

char *change_to_str(int x)
{
  int flag=0,y,z,j=0,i=100;
  char *str;
  char c;
  char s[5]=" ";
  while(i=1)
{
  if((x/i!=0)||((flag!=0))
  {
    y=x/i;
    z=x-y*i;
    x=z;
    c=y+48;
    s[j]=c;
    j++;
    flag=1;
  }
  i=i/10;
}
  s[j]=`0';
  str=s;
  return str;
}

int prune(p.x,y)
struct temp *p;
int x;
int y,flag=0;
{
  if(y=x)
  {
    for(i=0;i<x;i++)
    {
      if((strcmp(L2[i].item1,p->item1)==0)&(strcmp(L2[i].item1,p->item2)==0))
      {
        flag=1;
        break;
      }
    }
    if(flag==0)
    return 0;
  else
  {
    flag=0;
  }
for(i=0;i<x;i++)
if(strcmp(L2[i].item1.p->item1)==0&&(strcmp(L2[i].item1.p->item3)==0))
{
    flag=1;
    break;
}
if(flag==0)
    return 0;
else
{
    flag=0;
    for(i=0;i<x;i++)
    if(strcmp(L2[i].item1.p->item1)==0&&(strcmp(L2[i].item1.p->item3)==0))
    {
        flag=1;
        break;
    }
    if(flag==0)
        return 0;
    else
        return 1;
}
Appendix B: Condensed FUP2 Algorithm

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define SUPPORT 1.5
#define NUMTRANS 100000
#define INSERTTRANS 15000
#define DELETETRANS 15000
#define FILENAME "data1.5"
#define INSERTFILE "insert1.5"
#define DELETETFILE "delete1.5"
#define UNCHANGEFILE "unchange1.5"
#define ITEMSETLEN 5
#define CITEMSETSLEN 10000
#define COUNTNUM 10000

char *change_to_str(int);
int prune(struct *,int,int);

struct nlarge1
{
    char item[ITEMSETLEN];
    int sup;
}NL1[ITEMSETSLEN];

struct nlarge2
{
    char item1[ITEMSETLEN];
    char item2[ITEMSETLEN];
    int sup;
}NL2[ITEMSETSLEN];

struct large1
{
    char item[ITEMSETLEN];
    int sup;
}L1[ITEMSETSLEN];

struct large2
{
    char item1[ITEMSETLEN];
    char item2[ITEMSETLEN];
    int sup;
}L2[ITEMSETSLEN];

struct candidate1
{
    char item[ITEMSETLEN];
}C1[CITEMSETSLEN];

struct candidate2
{
    char item1[ITEMSETLEN];
    char item2[ITEMSETLEN];
}C2[CITEMSETSLEN];
struct s_p1
{
  char item[ITEMSETLEN];
  int sup;
} P1[ITEMSETLEN];

struct s_p2
{
  char item1[ITEMSETLEN];
  char item2[ITEMSETLEN];
  int sup;
} P2[ITEMSETLEN];

struct s_q1
{
  char item[ITEMSETLEN];
  int sup;
} Q1[ITEMSETLEN];

struct s_q2
{
  char item1[ITEMSETLEN];
  char item2[ITEMSETLEN];
  int sup;
} Q2[ITEMSETLEN];

main()
{
  FILE *fpt;
  char str[100], str1;
  char *s1, *s2, *s3;
  int count[COUNTNUM];
  int f, f1=0, f2=0, f3=0, k=0, k1=0, k2=0, k3=0;
  int old_large1_len, old_large2_len, old_large3_len, old_large4_len;
  int i, m, n, numl, p, q, flag, j=0, x, num=0;
  clock_t t;
  t=clock();

  /* read old large itemsets into structure */
  fpt=fopen("large1_1.5","r");
  if(fgets(str, 100, fpt) != NULL)
  {
    old_large1_len=strtol(str);
    while(fgets(str, 100, fpt) != NULL)
    {
      s1=strtok(str, " ");
      s2=strtok(NULL, " ");
      strcpy(L1[k].item, s1, strlen(s1)-1);
      L1[k].sup=atoi(s2);
      k++;
    } /* end of while */
  } /* end of if */
  fclose(fpt);

  /* generate P1 and Q1 */
  m=0;
  n=0;
for(i=0;i<1000;i++)
{
    flag=0;
    for(j=0;j<k;j++)
    
    if(strcmp(Cl[i].item,L1[j].item)==0)
    {
        strcpy(P1[m].item,L1[j].item);
        P1[m].sup=L1[j].sup;
        flag=1;
        m++;
        break;
    }
    if(flag==0)
    {
        strcpy(Q1[n].item,Cl[i].item);
        Q1[n].sup=0;
        n++;
    }
}

/* scan delete and insert file to generate NLI */

for(i=0;i<m;i++)
{

    fpt=fopen(INSERTFILE,"r");
    while(fgets(str,100,fpt)!=NULL)
    {
        s1=strtok(str," ");
        s2=strtok(NULL," ");
        s3=strtok(NULL," ");
        if(strcmp(P1[i].item,s3,strlen(s3)-1)==0)
        P1[i].sup=P1[i].sup+1;
    }
    fclose(fpt);
}

for(i=0;i<m;i++)
{

    fpt=fopen(DELETEFILE,"r");
    while(fgets(str,100,fpt)!=NULL)
    {
        s1=strtok(str," ");
        s2=strtok(NULL," ");
        s3=strtok(NULL," ");
        if(strcmp(P1[i].item,s3,strlen(s3)-1)==0)
        P1[i].sup=P1[i].sup-1;
    }
    fclose(fpt);
}

j=0;
for(i=0;i<m;i++)
    if(P1[i].sup/(float)NUMTRANS==SUPPORT/10)
{
    strcpy(NLI[j].item,P1[i].item);
    NLI[j].sup=P1[i].sup;
    j++;
}

for(i=0;i<n;i++)
{
fpt=fopen(INCLUDEFILE,"r");
while(fgets(str,100,fpt)!=NULL)
{
    s1=strtok(str," ");
    s2=strtok(NULL," ");
    s3=strtok(NULL," ");
    if(strcmp(Q1[i].item,s3,strlen(s3)-1)==0)
        Q1[i].sup=Q1[i].sup+1;
}
fclose(fpt);
}
for(i=0;i<n;i++)
{
    fpt=fopen(DELETEFILE,"r");
    while(fgets(str,100,fpt)!=NULL)
    {
        s1=strtok(str," ");
        s2=strtok(NULL," ");
        s3=strtok(NULL," ");
        if(strcmp(Q1[i].item,s3,strlen(s3)-1)==0)
            Q1[i].sup=Q1[i].sup-1;
    }
    fclose(fpt);
}
for(i=0;i<n;i++)
if(Q1[i].sup/(float)(DELETETRANS+INSERTTRANS)>=SUPPORT/100)
{
    fpt=fopen(UNCHANGEDFILE,"r");
    while(fgets(str,100,fpt)!=NULL)
    {
        s1=strtok(str," ");
        s2=strtok(NULL," ");
        s3=strtok(NULL," ");
        if(strcmp(Q1[i].item,s3,strlen(s3)-1)==0)
            Q1[i].sup=Q1[i].sup+1;
    }
    fclose(fpt);
    if(Q1[i].sup/(float)NUMTRANS>=SUPPORT/100)
    {
        strcpy(NLL[j].item,Q1[i].item);
        NLL[j].sup=Q1[i].sup;
        j++;
    }
}
/* generating candidate 2-itemsets */
q=0;
for(i=0;i<j;i++)
for(p=i+1;p<j;p++)
if(strcmp(NLL[i].item,NLL[p].item)==0)
    {strcpy(C2[q].item1,NLL[i].item);
    strcpy(C2[q].item2,NLL[p].item);
    q++;
    }
/* generating P2 and Q2 */
m=0;
n=0;

for(i=0;i<q;i++)
{
  flag=0;
  for(j=0;j<k1;j++)
    if((strcmp(C2[i].item1,L2[j].item1)==0) взгляд C2[i].item1,L2[j].item1==0))
      {
        strcpy(P2[m].item1,L2[j].item1);
        strcpy(P2[m].item2,L2[j].item2);
        P2[m].sup=L2[j].sup;
        flag=1;
        m++;
        break;
      }
    if(flag==0)
      {
        strcpy(Q2[n].item1,C2[i].item1);
        strcpy(Q2[n].item2,C2[i].item2);
        Q2[n].sup=0;
        n++;
      }
}

/* scan delete and insert file to generate NL */

fsl;
for(i=0;i<m;i++)
{
  fpt=fopen(INSETFILE,"r");
  while(fgets(str100,fpt))\n  {
    sl=strtok(str," ");
    s2=strtok(NULL," ");
    s3=strtok(NULL," ");
    num1=atoi(s3);
    if(num1!=f)
      {
        f=num1;
        fl=0;
      }
    if(strncmp(P2[i].item1,s3,strlen(s3)-1)==0)
      fl=1;
    if((strncmp(P2[i].item2,s3,strlen(s3)-1)==0) взгляд fl=1)\n      P2[i].sup=P2[i].sup+1;
  }
  fclose(fpt);

  fsl;
  fl=0;
  for(i=0;i<m;i++)
  {
    fpt=fopen(DELETFILE,"r");
    while(fgets(str100,fpt))\n    {
      sl=strtok(str," ");
      s2=strtok(NULL," ");
      s3=strtok(NULL," ");
      num1=atoi(s2);
      }
if(num1!=f)
{
    f=num1;
    fl=0;
}

if(strcmp(P2[i].item1,s3,strlen(s3)-1)==0)
    fl=1;

if(strcmp(P2[i].item2,s3,strlen(s3)-1)==0&&((fl==1))
    P2[i].sup=P2[i].sup+1;
}
fclose(fpt);

j=0;
for(i=0;i<n;i++)
    if(P2[i].sup/(float)NUMTRANS>(*SUPPORT/100)
        {
            strcpy(NL2[j].item1,P2[i].item1);
            strcpy(NL2[i].item2,P2[i].item2);
            NL2[j].sup=P2[i].sup;
            j++;
        }

f=1;
for(i=0;i<n;i++)
    {
        fpt=fopen(INSSERTFILE,"r");
        while(fgets(str,100,fpt)!=NULL)
            {
                s1=strtok(str,*);
                s2=strtok(NULL,*);
                s3=strtok(NULL,*);
                num1=atoi(s2);

                if(num1!=f)
                    {
                        f=num1;
                        fl=0;
                    }

                if(strcmp(Q2[i].item1,s3,strlen(s3)-1)==0)
                    fl=1;

                if(strcmp(Q2[i].item2,s3,strlen(s3)-1)==0&&((fl==1))
                    Q2[i].sup=Q2[i].sup+1;
                fclose(fpt);
            }
    }

f=1;
fl=0;
for(i=0;i<n;i++)
    {
        fpt=fopen(DELETEFILE,"r");
        while(fgets(str,100,fpt)!=NULL)
            {
                s1=strtok(str,*);
                s2=strtok(NULL,*);
                s3=strtok(NULL,*);
                num1=atoi(s2);
            }
if(num1==f)
{
    f=num1;
    f1=0;
}
if(strcmp(Q2[i].item1,s3,strlen(s3)-1)==0)
    f1=1;
if((strcmp(Q2[i].item2,s3,strlen(s3)-1)==0)&&(f1==1))
    Q2[i].sup=Q2[i].sup+1;
fclose(fpt);
}

f=1;
f1=0;
for(i=0;i<n;i++)
    if(Q2[i].sup/((float)(INSERTTRANS+DELETETRANS))>=SUPPORT/100)
        fpt=fopen(UNCHANGEFILE, "r");
        while(fgets(str100,fpt)!=NULL)
        {
            s1=strtok(str," ");
            s2=strtok(NULL," ");
            s3=strtok(NULL," ");
            num1=atoi(s2);
            if(num1!=f)
            {
                f=num1;
                f1=0;
            }
        }
if(strcmp(Q2[i].item1,s3,strlen(s3)-1)==0)
    f1=1;
if((strcmp(Q2[i].item2,s3,strlen(s3)-1)==0)&&(f1==1))
    Q2[i].sup=Q2[i].sup+1;
fclose(fpt);
if(Q2[i].sup/((float)(NUMTRANS>=SUPPORT/100))
    strcpy(NL2[j].item1,Q2[i].item1);
    strcpy(NL2[j].item2,Q2[i].item2);
    NL2[j].sup=Q2[i].sup;
    j++;
}

/* generate candidate 3-itemsets */
q=0;
for(i=0;i<j;i++)
    for(p=i+1;p<j;p++)
        if((strcmp(NL2[i].item1,NL2[p].item1)==0)&&(strcmp(NL2[i].item2,NL2[p].item2)<0))
            { 
                strcpy(temp.item1,L3[i].item1);
                strcpy(temp.item2,L3[i].item2);
                strcpy(temp.item3,L3[j].item3);
                if(prune(&temp,j,3)==1)
                { 
                    strcpy(C3[q].item1,NL2[i].item1);
                    strcpy(C3[q].item2,NL2[i].item2);
                    strcpy(C3[q].item3,NL2[p].item2);
                    q++;
            
            }


```
    printf("%.2fs\n", (clock() - t) / (double)CLOCKS_PER_SEC);

/* end of main */

char *change_co_str(int x)
{
    int flag = 0, y, z, j = 0, i = 100;
    char *str;
    char c;
    char s[5] = " ";
    while (i = 1)
    {
        if ((y = i) != 0) || (flag != 0))
        {
            y = x / i;
            z = x - y * i;
            x = z;
            c = y;
            s[i] = c;
            j = 0;
            flag = 1;
        }
        i = i / 10;
    }
    s[i] = '\0';
    str = s;
    return str;
}

int prune(p, x, y)
struct temp *p;
int x;
int y, flag = 0;
{
    if (y = 3)
    {
        for (i = 0; i < x; i++)
        {
        if ((strcmp(L2[i].item1, p->item1) == 0) && (strcmp(L2[i].item1, p->item2) == 0))
            {
                flag = 1;
                break;
            }
        if (flag == 0)
            return 0;
        else
            {
                flag = 0;
                for (i = 0; i < x; i++)
                {
                if ((strcmp(L2[i].item1, p->item1) == 0) && (strcmp(L2[i].item1, p->item3) == 0))
                    {
                        flag = 1;
                        break;
                    }
                if (flag == 0)
                    return 0;
                else
                    {
                        flag = 0;
                        for (i = 0; i < x; i++)
                        {
                        if ((strcmp(L2[i].item1, p->item1) == 0) && (strcmp(L2[i].item1, p->item3) == 0))
                            {
                            flag = 1;
                        ```
break;
} else
  return 1;
}
}
Appendix C: Condensed MAAP Algorithm

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define SUPPORT 1.5
#define NUMTRANS 100000
#define INSERTTRANS 15000
#define DELTRANS 15000
#define FILENAME "data1.5"
#define INSERTFILE "insert1.5"
#define DELETEFILE "delete1.5"
#define UNCHANGECFILE "unchangel.5"
#define ITEMSETLEN 6
#define ITEMSETSLEN 1000
#define CITEMSETSLEN 10000
#define COUNTNUM 10000

char *change_to_str(int);
int prune(struct *, int, int);
void insert_low_large(struct *, int, int);
int check1(struct *, int);
int check2(struct *, int);
int find_sup(char *, char *, char *, int);
int find_sup1(char *, char *, int);
int find_sup2(char *, int);

struct nlarge1
{
    char item[ITEMSETLEN];
    int sup;
}NL1[ITEMSETSLEN];

struct nlarge2
{
    char item1[ITEMSETLEN];
    char item2[ITEMSETLEN];
    int sup;
}NL2[ITEMSETSLEN];

struct large1
{
    char item[ITEMSETLEN];
    int sup;
}L1[ITEMSETSLEN];

struct large2
{
    char item1[ITEMSETLEN];
    char item2[ITEMSETLEN];
    int sup;
}L2[ITEMSETSLEN];

struct candidate1
{
    char item[ITEMSETLEN];
}C1[CITEMSETSLEN];
struct candidate2
{
    char item1[ITEMSETLEN];
    char item2[ITEMSETLEN];
    }C2[ITEMSETLEN];

struct s_p1
{
    char item1[ITEMSETLEN];
    int sup;
    }P1[ITEMSETLEN];

struct s_p2
{
    char item1[ITEMSETLEN];
    char item2[ITEMSETLEN];
    int sup;
    }P2[ITEMSETLEN];

struct s_q1
{
    char item1[ITEMSETLEN];
    int sup;
    }Q1[ITEMSETLEN];

struct s_q2
{
    char item1[ITEMSETLEN];
    char item2[ITEMSETLEN];
    int sup;
    }Q2[ITEMSETLEN];

int j1=0,j2=0,j3=0;

main()
{
    FILE *fpt;
    char str[100],*str1,str2[100];
    char *s1,*s2,*s3;
    int count[COUNTNUM];
    int i,m,n,l,n2,n3,n4,num1,x,y=0,p,q,flag,flagl,highlevel=9;
    int old_large1_len,old_large2_len,old_large3_len,old_large4_len;
    int f,ff=0,f1=0,f2=0,f3=0,j=0,k=0,kl=0,k2=0,k3=0,num=0;
    clock_t t;
    t=clock();
    /* read old large itemsets from file to structure array */
    fpt=fopen("large1_2.0","r");
    if(fgets(str,100,fpt)!=NULL)
        {
            old_large1_len=atoi(str);
            while(fgets(str,100,fpt)!=NULL)
            { 
                sl=strtok(str,"");
                s2=strtok(NULL," ");
                strcpy(Ll[k].item.sl,strlen(sl)-1);
                Ll[k].sup=atoi(s2);
                k++;
            }/* end of while */
        }/* end of if */
    fclose(fpc1);
/* from old high level large itemsets to obtain low level large itemsets */

for(r=highlevel;r>=1;r--)
{
    if(r==9)
    {
    }
    else if(r==3)
    {
        for(i=0;i<k3;i++)
        {
            fpt=fopen(INSERTNAME, "r");
            while(fgets(str,100,fpt) != NULL)
            {
                s1=strtok(str, " ");
                s2=strtok(NULL, " ");
                s3=strtok(NULL, " ");
                num=atoi(s2);

                if(num!=f)
                {
                    f=num;
                    f1=0;
                    f2=0;
                }

                if(strcmp(s3,L3[i].item1,strlen(s3)-1)==0)
                    f1=1;

                if(strcmp(s3,L3[i].item2,strlen(s3)-1)==0)
                    f2=1;

                if((strcmp(s3,L3[i].item3,strlen(s3)-1)==0) && (f2==1))
                    L3[i].sup=L3[i].sup+1;
        }
    } /*end of while */
    fclose(fpt);

    fpt=fopen(DELETENAME, "r");
    while(fgets(str,100,fpt) != NULL)
    {
        s1=strtok(str, " ");
        s2=strtok(NULL, " ");
        s3=strtok(NULL, " ");
        num=atoi(s2);

        if(num!=f)
        {
            f=num;
            f1=0;
            f2=0;
        }

        if(strcmp(s3,L3[i].item1,strlen(s3)-1)==0)
            f1=1;

        if(strcmp(s3,L3[i].item2,strlen(s3)-1)==0)
            f2=1;

        if((strcmp(s3,L3[i].item3,strlen(s3)-1)==0) && (f2==1))
            L3[i].sup=L3[i].sup-1;
    } /*end of while */
fclose(fpt);

if(L3[i].sup/(float)NUMTRANS>SUPPORT/100)
{
  strcpy(temp_item1,L3[i].item1);
  strcpy(temp_item2,L3[i].item2);
  strcpy(temp_item3,L3[i].item3);
  temp.sup=L3[i].sup;
  insert_low_large(p,1,y);
  ff=1;
  y++;
}
if(ff==1)
  break;

/* generate P1 and Q1 */

m=0;
for(i=0;i<1000;i++)
{
  flag=0;
  for(j=0;j<k;j++)
  {
    if(strcmp(Cl[i].item.Ll[j].item)==0)
    {
      flag1=0;
      for(p=0;p<j+1;p++)
      {
        if(strcmp(Cl[i].item.NL1[p].item)==0)
        {
          flag1=1;
          break;
        }
      }
      if(flag1==1)
      {
        strcpy(P1[m].item.Ll[j].item);
        P1[m].sup=Ll[j].sup;
        flag=1;
        m++;
        break;
      }
    }
  }

  if(flag==0)
  {
    strcpy(Q1[n].item.Cl[i].item);
    Q1[n].sup=0;
    n++;
  }
}

/* scan delete and insert file to generate NL1 */

for(i=0;i<m;i++)
{
  fpt=fopen(INSERTFILE,"r");
  while(fgets(str,100,fpt)!=NULL)
  {
    s1=strtok(str," ");
    s2=strtok(NULL," ");
    s3=strtok(NULL," ");
    if(strcmp(P1[i].item,s3,strlen(s3)-1)==0)
Pl[i].sup=Pl[i].sup+1;
}
fclose(fpt);
}

for(i=0;i<m;i++)
{
  fpt=fopen(DELETEFILE, "r");
  while(fgets(str,100,fpt)!=NULL)
  {
    s1=strtok(str, ";");
    s2=strtok(NULL, ";");
    s3=strtok(NULL, ";");
    if(strcmp(PI[i].item,s1, strlen(s1)-1) == 0)
      PI[i].sup=PI[i].sup+1;
  }
  fclose(fpt);
}

for(i=0;i<n;i++)
{
  if(PI[i].sup/(float)NUMTRANS>=SUPPORT/100)
  {
    strcpy(NL[i].item,PI[i].item);
    NL[i].sup=PI[i].sup;
    ji++;
  }
}

for(i=0;i<n;i++)
{
  fpt=fopen(INCREASEFILE, "r");
  while(fgets(str,100,fpt)!=NULL)
  {
    s1=strtok(str, ";");
    s2=strtok(NULL, ";");
    s3=strtok(NULL, ";");
    if(strcmp(QI[i].item,s3, strlen(s3)-1) == 0)
      QI[i].sup=QI[i].sup+1;
  }
  fclose(fpt);
}

for(i=0;i<n;i++)
{
  fpt=fopen(DELETEFILE, "r");
  while(fgets(str,100,fpt)!=NULL)
  {
    s1=strtok(str, ";");
    s2=strtok(NULL, ";");
    s3=strtok(NULL, ";");
    if(strcmp(QI[i].item,s3, strlen(s3)-1) == 0)
      QI[i].sup=QI[i].sup-1;
  }
  fclose(fpt);
}

for(i=0;i<n;i++)
{
  if(QI[i].sup/(float)(INSERTTRANS+DELETETRANS)>=SUPPORT/100)
  {
    fpt=fopen(UNCHANGEFILE, "r");
    while(fgets(str,100,fpt)!=NULL)
    {
      
    }
  }
}
s1=strtok(str," ");
    s2=strtok(NULL," ");
    s3=strtok(NULL," ");
    if(strcmp(Q1[i].item,s3,strlen(s3)-1)==0)
        Q1[i].sup=Q1[i].sup+1;
}
fclose(fpt);
if(Q1[i].sup/(float)NUMTRANS>=SUPPORT/100)
{
    strcpy(NL1[jl].item,Q1[i].item);
    NL1[jl].sup=Q1[i].sup;
    jl++;
}

/* generating candidate 2-itemsets */
q=0;
for(i=0;i<j;i++)
    for(p=1;p<j;p++)
        if(strcmp(NL1[i].item,NL1[p].item)<=0)
            {strcpy(C2[q].item1,NL1[i].item);
             strcpy(C2[q].item2,NL1[p].item);
             q++;
            }
/* generating P2 and Q2 */
m=0;
n=0;
for(i=0;i<q;i++)
{
    flag=0;
    for(j=0;j<k;j++)
        if((strcmp(C2[i].item1,L2[j].item1)==0)&&(strcmp(C2[i].item2,L2[j].item2)==0))
            {flag1=0;
             for(p=0;p<j;p++)
                 if(strcmp(C2[i].item1,NL2[p].item1)==0)&&(strcmp(C2[i].item2,NL2[p].item2)==0))
                     {flag1=1;
                      break;
                     }
                if(flag1==1)
                {
                    strcpy(P2[m].item1,L2[j].item1);
                    strcpy(P2[m].item2,L2[j].item2);
                    P2[m].sup=L2[j].sup;
                    flag1=1;
                    m++;
                    break;
                }
        }
    if(flag==0)
    {
        strcpy(Q2[n].item1,C2[i].item1);
        strcpy(Q2[n].item2,C2[i].item2);
        Q2[n].sup=0;
        n++;
    }
}
/* scan delete and insert file to generate NL2 */

f=1;
for(i=0;i<m;i++)
{

fpt=fopen(INSERTFILE,"r");
while(fgets(str.100,fpt)!=NULL)
{
    sl=strtok(str," ");
    s2=strtok(NULL," ");
    s3=strtok(NULL," ");
    num1=atoi(s2);
    if(num1==f)
    {
        f=num1;
        f1=0;
    }
    if((strcmp(P2[i].item1,s3,strlen(s3)-1)==0)
    f1=1;
    if((strcmp(P2[i].item2,s3,strlen(s3)-1)==0)&&(f1==1))
    P2[i].sup=P2[i].sup+1;
}
fclose(fpt);
}

f=1;
fi=0;
for(i=0;i<m;i++)
{

fpt=fopen(DELETEFILE,"r");
while(fgets(str.100,fpt)!=NULL)
{
    sl=strtok(str," ");
    s2=strtok(NULL," ");
    s3=strtok(NULL," ");
    num1=atoi(s2);
    if(num1==f)
    {
        f=num1;
        f1=0;
    }
    if((strcmp(P2[i].item1,s3,strlen(s3)-1)==0)
    f1=1;
    if((strcmp(P2[i].item2,s3,strlen(s3)-1)==0)&&(f1==1))
    P2[i].sup=P2[i].sup+1;
}
fclose(fpt);
}

for(i=0;i<m;i++)
    if((P2[i].sup/((float)NUMTRANS)*SUPPORT/100)
    {
        strcpy(NL2[j2].item1,P2[i].item1);
        strcpy(NL2[j2].item2,P2[i].item2);
        NL2[j2].sup=P2[i].sup;
        j2++;
    }

    printf("%.2fs
",(clock()-t)/(double)CLOCKS_PER_SEC);

} /* end of main */
char *change_to_str(int x)
{
    int flag=0,y,z,j=0,i=100;
    char *str;
    char c;
    char s[51]= " ";
    while(i>=1)
    {    
        if((x%i==0) || (flag==0))
            {      
            y=x/i;
            z=x-y*i;
            x=z;
            c=y+48;
            s[j]=c;
            j++;
            flag=1;
        }
        i=i/10;
    }
s[j]='\0';
    str=s;
    return str;
}

int check1(q,count_num)
struct temp *q;
int count_num;
{
    int i,f=0;
    for(i=0;i<count_num;i++)
        if((strcmp(NL1[i].item,q->item)==0))
            {            
            fg=1;
            break;
            }
        if(fg=1)
            return 1;
        else
            return 0;
}

int check2(q,count_num)
struct temp *q;
int count_num;
{
    int i,f=0;
    for(i=0;i<count_num;i++)
        if((strcmp(NL2[i].item1,q->item1)==0) && (strcmp(NL2[i].item2,q->item2)==0))
            {            
            f=1;
            break;
            }
        if(f=1)
            return 1;
        else
            return 0;
}

void insert_low_large(p,k,count)
struct temp *p;
int k,count;
{
    if(k==9)
if(k==3)
{
    strcpy(NL3[count].item1.p->item1);
    strcpy(NL3[count].item2.p->item2);
    strcpy(NL3[count].item3.p->item3);
    NL3[count].sup+=p->sup;
    count++;
}

if(check2(p,j1)!=1)
{
    strcpy(NL2[j2].item1.p->item1);
    strcpy(NL2[j2].item2.p->item2);
    NL2[j2].sup=find_sup1(p->item1,p->item2);
    j2++;
}

if(check2(p,j2)!=1)
{
    strcpy(NL2[j2].item1.p->item1);
    strcpy(NL2[j2].item2.p->item2);
    NL2[j2].sup=find_sup1(p->item1,p->item3);
    j2++;
}

if(check2(p,j2)!=1)
{
    strcpy(NL2[j2].item1.p->item2);
    strcpy(NL2[j2].item2.p->item3);
    NL2[j2].sup=find_sup1(p->item2,p->item3);
    j2++;
}

if(check1(p,j1)!=1)
{
    strcpy(NL1[j1].item.p->item1);
    NL1[j1].sup=find_sup2(p->item1);
    j1++;
}

if(check1(p,j1)!=1)
{
    strcpy(NL1[j1].item.p->item2);
    NL1[j1].sup=find_sup2(p->item2);
    j1++;
}

if(check1(p,j1)!=1)
{
    strcpy(NL1[j1].item.p->item3);
    NL1[j1].sup=find_sup2(p->item3);
    j1++;
}

int find_sup1(r1,r2)
char r1[1],r2[1];
{
    int i=0;
}
while(strcmp(L2[i].item1,**)!=0)
    if((strcmp(r1,L2[i].item1)==0)&&(strcmp(r2,L2[i].item2)==0))
        return L2[i].sup;
    else
        i++;
}
}

int find_sup2(r1)
char r1[];
{
    int i=0;
    while(strcmp(L1[i].item,**)!=0)
        if(strcmp(r1,L1[i].item)==0)
            return L1[i].sup;
    else
        i++;
}
}
VITA AUTHORS

Zequn Zhou was born in 1966 in Tianjin, China. She graduated from Tianjin 2th High School in 1984. Then she went to Tianjin Institute of Technology where she obtained a Bachelor of Computer Engineering degree in 1988. She is currently a candidate for the Master's degree in Computer Science at the University of Windsor and will graduate in the Fall term, 2000.