IMPROVED ALGORITHM TO MINE ASSOCIATION RULES

A DISSERTATION

Submitted in partial fulfilment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

SAURABH GUPTA

ER & DCI
NOIDA

IIT Roorkee-ER&DCI, Noida
C-56/1, "Anusandhan Bhawan"
Sector 62, Noida-201 307
FEBRUARY, 2003

Enrolment No. 019047

621.380285

GUP

CANDIDATE'S DECLARATION

This is to certify that the work, which is being presented in this dissertation, entitled "**IMPROVED ALGORITHM TO MINE ASSOCIATION RULES**", in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Information Technology** submitted in **IIT, Roorkee – ER&DCI Campus, Noida,** is an authentic record of my own work carried out from August 2002 to February 2003, under the supervision of **Dr. P.R. GUPTA**, Reader, Electronics Research and Development Centre of India, Noida.

I have not submitted the matter embodied in this dissertation for the award of any other degree.

Date:

Place: Noida

[illegible]

(Saurabh Gupta)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 24·2·2003

Place: Noida

[illegible]

[illegible]

(Dr. P.R. Gupta)
Reader,
ER&DCI, Noida

ACKNOWLEDGEMENT

I hereby take the privilege express my deepest sense of gratitude for **Dr. Prem Vrath**, Director, Indian Institute of Technology, Roorkee, and **Mr. R.K. Verma,** Executive Director, ER&DCI, Noida for providing me with this valuable opportunity to carry out this work. I am also very grateful to **Dr. A.K. Awasthi**, our programme coordinator and **Dr. R.P. Agrawal** and **Mr. V.N. Shukla,** our course coordinators for providing the best of the facilities for the completion of this work and constant encouragement towards the goal.

I have no words to thank my guide, **Dr. P.R. Gupta**, Reader, ER&DCI, Noida for his guidance and invaluable suggestions during the entire course of this work. She provided me continuous inspiration and support throughout the course of this dissertation.

I am highly indebted to **Dr. R.C. Joshi**, Professor, IIT-Roorkee, to provide me insight of data mining and warehousing. I am also thankful to **Miss Durga Toshniwal**, Student P.H.D., and IIT-Roorkee, to provide me constant support throughout the entire course of this work. I am also grateful to **Mr. Munish Kumar** for the cooperation extended by him in the successful completion of this work.

It is impossible to mention the names of all those persons who have been involved, directly or indirectly, with this work and I extend my gratitude to all of them. However, I feel, I owe special thanks to all my friends who have helped me formulate my ideas and have been a constant support. I find myself short of words to thank my father, mother and brother who have always been by my side throughout my life.

[illegible]

(Saurabh Gupta)

019047

Contents

ABSTRACT

This work is an attempt to develop an efficient algorithm to mine association rules. The problem of association rule generation has recently gained considerable prominence in the data mining community because of its use as a tool for knowledge discovery. Consequently, there has been a spurt of research activity in the recent years surrounding this problem.

Data mining is motivated by the decision support problem faced by most large retail organizations. Progress in bar-code technology has made it possible for retail organizations to collect and store massive amount of data, referred to as the basket data. A record in such data typically consists of the transaction data and the items bought in the transaction. Successful organizations view such databases as important component of the marketing strategy. They are interested in instituting information-driven marketing process, managed by database technology, which enables marketers to develop and implement customized marketing programs and strategies.

An association rule identifies a combination of attribute or items that occur together with greater frequency than might be expected if the values or items were independent of one-another. Association rules find the relationship between the different attributes in a transaction database. Such rules track the patterns in transactions such as finding how the presence of one attribute in the transaction affects the presence of another and so forth.

An association rule is the expression of the form **A=>B** where A and B are Boolean attributes and the symbol => is

called quantifier. The idea of an association rule is to develop a systematic method by which a user can figure out how to infer the presence of some sets of attributes, given the presence of other attributes in a transaction. Such information is useful in making decision such as customer targeting, shelving, and sales promotion.

Here the main focus is on reducing number of candidate item sets generated and number of database scans in the process of association rules mining.

Chapter 1

Introduction

1.1Overview

Data mining is nowadays one of the most active research topics in computer science. It is now proven that many areas could benefit from it e.g. To increase the number of items sold, for instance, by appropriately arranging the products in the shelves of a supermarket (they may, for example, be placed adjacent to each other in order to invite even more customers to buy them together).

Association analysis is a major functionality of data mining. Many papers investigated on various methods for association rule mining, concept and theories. Most of the current research on association analysis has two general goals: reduction of candidate Itemsets and reduction of database scans.

Typical algorithm for mining association rules is Apriori algorithm. This algorithm performs reasonably well when all maximal frequent item sets are short. However performance drastically decreases when some of the maximal frequent item sets are relatively long. So the attempt is to mine association rules (frequent item sets) by a method that will efficiently work for small as well as long item sets.

1.2 Problem Description

Given a set of transactions D, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user specified minimum support (called min_supp) and minimum confidence (called min_conf) respectively.

Here is the formal statement of problem [5]: Let $I=\{i_1, i_2 \ldots i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that T is the subset of I. Associated with each transaction is a unique identifier, called its TID. A transaction T contains X, a set of some items in I, if X is the subset of T. An association rule is an implication of the form X => Y, where X and Y are the proper subset of I and no item is common in X and Y. The rule X => Y holds in the transaction set D with confidence c, if c% of transactions in D that contain X also contain Y. The rule X => Y has support s in the transaction set D, if s% of transactions in D contain X U Y.

1.3 Problem Decomposition

The problem of discovering all association rules can be decomposed into two sub problems:

1. Find all sets of items (Itemsets) that have transaction support above minimum support. The support for an itemset is the number of transactions that contain the itemset. Itemsets with minimum support are called large Itemsets, and all others small Itemsets.
2. Use the large Itemsets to generate the desired rules. The general idea is that if, say, ABCD and AB are large Itemsets, then If conf >= minconf, then the rule AB=>CD holds.

   Actual aim of this work is to reduce the number of candidate itemsets generated and to reduce the number of database scans in the process of association rules mining.

1.4 Organization of Report

The first chapter gave an overview of Association analysis and discussed the problem to be solved. The second chapter presents the essence of the literature surveyed and discusses relevant theoretical issues. The third chapter carries out a detailed analysis of the problem, the solution for which is to be developed. Chapter four presents the detailed design of the proposed solution follows this. Chapter five gives the implementation of the solution. In chapter six, results obtained from the software developed are presented and discussed. Finally, chapter eight concludes the work.

Chapter 2

Literature Survey

2.1 Data mining

Simply stated, data mining refers to extraction of interesting (non-trivial, implicit, previously unknown and potentially useful) information or patterns from data in large databases [2].

Automated data collection tools and matures database technology lead to tremendous amounts of data stored in databases, data warehouses and other information repositories. So we are drowning in data, but starving for knowledge!

Solution: Data warehousing and data mining

2.2 Data mining as knowledge discovery process

Figure 2.1 shows Data mining as knowledge discovery process [2]. The steps involved are:

- Learning the application domain:
  Relevant prior knowledge and goals of application
- Creating a target data set: data selection
- Data cleaning and preprocessing: (may take 60% of effort!)
  To remove noise and inconsistent data
- Data integration
  Where data relevant to the analysis task are retrieved from the database.
- Data reduction and transformation
  Where data are transformed or consolidated into forms appropriate for mining by performing summary or aggregation operations, for instance.
- Choosing functions of data mining
  Summarization, classification, regression, association, clustering.
- Choosing the mining algorithm(s)
- Data mining:

Search for information/patterns of interest.

- Pattern evaluation

  To identify the truly interesting patterns representing knowledge based on some interestingness measures.

- Knowledge presentation

  Visualization, transformation, removing redundant patterns, etc

  - Use of discovered knowledge

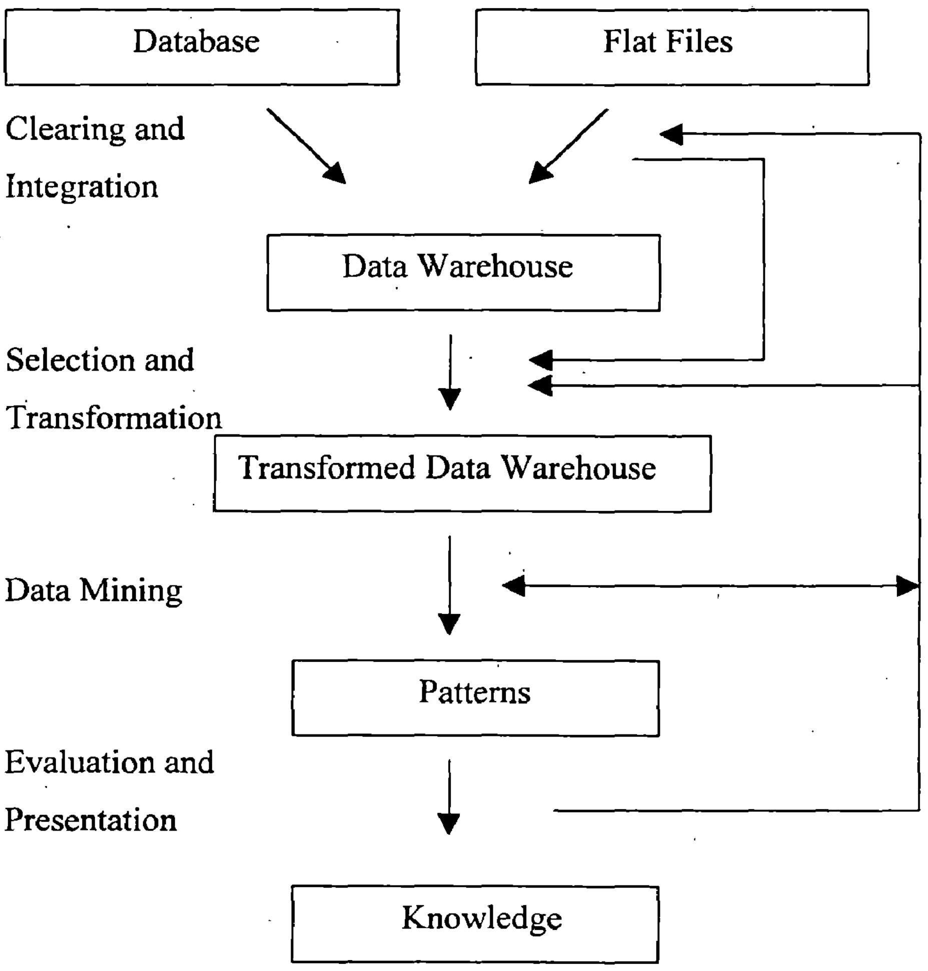


Fig. 2.1: Data Mining as Knowledge Discovery Process

2.3 Data warehousing

Data warehousing is the process of constructing and using data warehouses [2]. Data warehouse can be defined in many different ways [2].

- A repository of multiple heterogeneous data sources organized under a unified schema at a single site in order to facilitate management decision-making.
- A decision support database that is maintained separately from the organization's operational database.
- Support information processing by providing a solid platform of consolidated, historical data for analysis.
- "A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision-making process."—W. H. Inmon

2.4 Data Mining Functionalities

Data mining functionalities [2] are used to specify the kind of pattern to be found in data mining task.

Data mining functionalities and the kinds of patterns they can discover are described below.

2.4.1 Concept/Class Description: Characterization and discrimination

Data can be associated with class or concepts. For example, in a store "XYZ", classes of items for sale include computers and printers and concepts of customers include bid spenders and budget spenders. It can be useful to describe individual classes and concepts in summarized, concise and yet precise terms. Such description of a class and concepts are called concept/class description. This description can be derived via data characterization or data discrimination.

Data characterization is the process of summarizing the data of the class under study.

Data discrimination is the process of comparing target class with one or more comparative classes.

2.4.2 Association Analysis

Association analysis is the discovery of association rules showing attribute-value conditions that occur frequently together in a given set of data

Example:

Age (X, "20..29") ^ income (X, "20..29K") => buys (X, "PC")

[Support = 2%, confidence = 60%]

2.4.3 Classification and Prediction

Classification is the process of finding models (functions) that describe and distinguish classes or concepts for future prediction

E.g., classify countries based on climate, or classify cars based on gas mileage

Prediction is the process of Predicting some unknown or missing numerical values

2.4.4 Cluster analysis

If class label is unknown, Group data to form new classes, e.g., cluster houses to find distribution patterns.

Clustering is based on the principle of maximizing the intra-class similarity and minimizing the interclass similarity.

2.4.5 Outlier Analysis

Outlier is a data object that does not comply with the general behavior of the data It can be considered as noise or exception but is quite useful in fraud detection and rare event analysis.

2.4.6 Evolution Analysis

Data evolution analysis describes and models regularities or trends for objects whose behavior changes over time.

2.5 Frequent sets mining

Frequent item sets are the sets of items that have minimum support specified by the user [2].

- A subset of a frequent item set must also be a frequent item set, i.e., if {AB} is a frequent item set, both {A} and {B} should be a frequent item sets.
- Frequent item sets are used to generate association rules.

2.6 Association Analysis

Finding association rules that represent correlation between items is called association analysis or association rule mining [2].

2.6.1 Association Rules

Association rules find the relation between the items in a database of sales transactions.

An association rule identifies a combination of attribute or items that occur together with greater frequency than might be expected if the values or items were independent of one-another.

The association rule is the expression of the form **A => B** where A and B are Boolean attributes and the symbol => is called quantifier. Meaning of the association rule **A =>B** is that Boolean attributes A and B are associated in the way given by the quantifier =>.

Boolean attributes A and B are conjunctions of literals. Figure 2.2 shows examples of literals are Sex (F), District (U.P) and Quality (bad). They are derived from attributes Sex, District and Quality corresponding to columns in data matrix concerning loans of the fictitious bank.

The Boolean attribute Sex (F) is true in a row of data matrix if there is the value F in this row and in the column Sex. The Boolean attribute District (U.P) is true in the row of data matrix if there is the value U.P in this row and in the column District. The Boolean attribute Quality (bad) is true in the row of data matrix if there is the value bad in this row in the column Quality:

Id	**Sex**	**District**	**Quality**	***Sex (F)***	***District (U.P)***	***Quality (bad)***
1	M	U.P	Good	*False*	*True*	*False*
2	M	Gujarat	Bad	*False*	*False*	*True*
3	F	Delhi	Bad	*True*	*False*	*True*
...	...	...	...	...	...	...
6180	M	U.P	Bad	*False*	*True*	*True*
6181	F	M.P	Good	*True*	*False*	*False*

Fig 2.2: Transaction Data Base

An example of association rule is,

Sex (F) & District (U.P) =>Quality (bad). 30%

It means that at least 30 per cent of clients – women living in U.P have the loan of bad quality.

2.6.2 Rule Measures: Support and Confidence

2.6.2.1 Support

Support 's', is the percentage of transactions in database D that contain AUB (i.e. both A and B). This is taken to be the probability, P (AUB) [2]. That is,

For rule A => B,

Support (A => B) = P (AUB)

$$\text{Support } (A \Rightarrow B) = \frac{\text{Support\_count } (AUB)}{\text{Support\_count total}}$$

2.6.2.2 Confidence

The rule A => B has confidence 'c' in the transaction set D if c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, P (B/A) [2]. That is,

For rule A => B,

Confidence (A => B) = P (B/A)

$$\text{Confidence (A => B)} = \frac{\text{Support\_count (AUB)}}{\text{Support\_count A}}$$

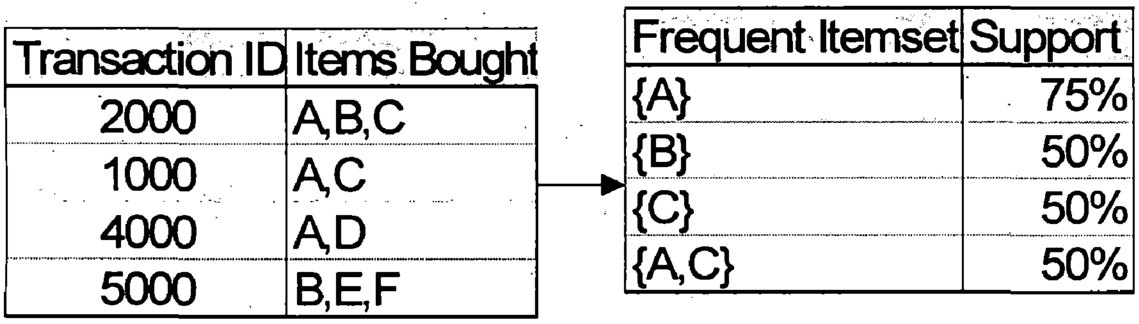
Where support_count (AUB) is the number of transactions containing the itemsets AUB, and support_count (A) is the number of transactions containing the itemset A.

Rules that satisfy both minimum support and minimum confidence are called strong association rules.

Example:

Min. support 50%

Min. confidence 50%



Transaction ID	Items Bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Frequent Itemset	Support
{A}	75%
{B}	50%
{C}	50%
{A,C}	50%

Fig.2.3: Example: Support and Confidence

For rule $A \Rightarrow C$,

Support = support ({AUC}) = 50%

Confidence = support ({AUC})/support ({A}) = 66.6%

2.6.3 Types of Association Rules

There are various types of association rules [2] are present:

Boolean vs. quantitative associations

Based on the types of values handled

- Boolean:

  Buys (x, "SQL Server") ^ Buys (x, "DM Book") => Buys (x, "DB Miner") [0.2%, 60%]

- Quantitative:

  Age (x, "30…39") ^ Income (x, "42…48K") => Buys (x, "PC") [1%, 75%]

Single dimension vs. multiple dimensional associations

Based on the dimension

- Single Dimensional:

  Buys (x, "diapers") => Buys (x, "beers") [0.5%, 60%]

- Multidimensional:

  Major (x, "CS") ^ Takes (x, "DB") => Grade (x, "A") [1%, 75%]

Multiple-level analysis

Based on the level of abstraction

- Multilevel:

  Age (X, 30...39) => Buys (X," laptop computer")

- Single level:

  Age (X, 30…39) => Buys (X," computer")

2.6.4 Basic Method of Association analysis

An important approach regarding association roles was proposed by Agrawal [5]. It is a two-phase approach as follows:

Generate all combinations of items that have fractional transaction support above a certain user-defined threshold called min_supp. All such combinations are called large itemsets.

Given an itemset S={I_1, I_2…I_k}, It can be used to generate at most k rules of the type [S- {I_r}] =>{I_r}, for each r is the element of {1…k}. Once these rules have been generated, only those rules above a certain user-defined threshold called min_conf may be retained.

In order to generate the large 1-itemsets, an iterative approach is used to first generate the set of large 1-itemset L_1, then the set of large itemsets L_2, and so on until for some value of r the set L_r is empty. At this stage the algorithm can be terminated. During the kth iteration of this procedure, a set of candidates C_k is generated, by performing a (k-2) join on the large itemsets L_{k-1}. The itemsets in this set C_k are candidates for large itemsets, and the final set of large itemsets L_k must be a subset of C_k. Each element of C_k needs to be validated against the transaction database to see if it indeed belongs to L_k. The validation of the candidate itemset C_k against the transaction database seems to be bottleneck operation for the algorithm. This method requires multiple passes over a transaction database, which may potentially be quite large. For evaluating itemsets with a specific number of items, one pass is required over the transaction database. Thus, if the large itemset with the maximum number of items has 9 items in it, then the method requires 9 passes over the transaction database. This may result in substantial I/O times for the algorithm.

2.6.4.1 Apriori Algorithm

2.6.4.1.1 The Apriori Principle

Any subset of a frequent itemset must be frequent [2]

i.e., if {AB} is a frequent itemset, both {A} and {B} should be a frequent itemset.

2.6.4.1.2. Key Steps

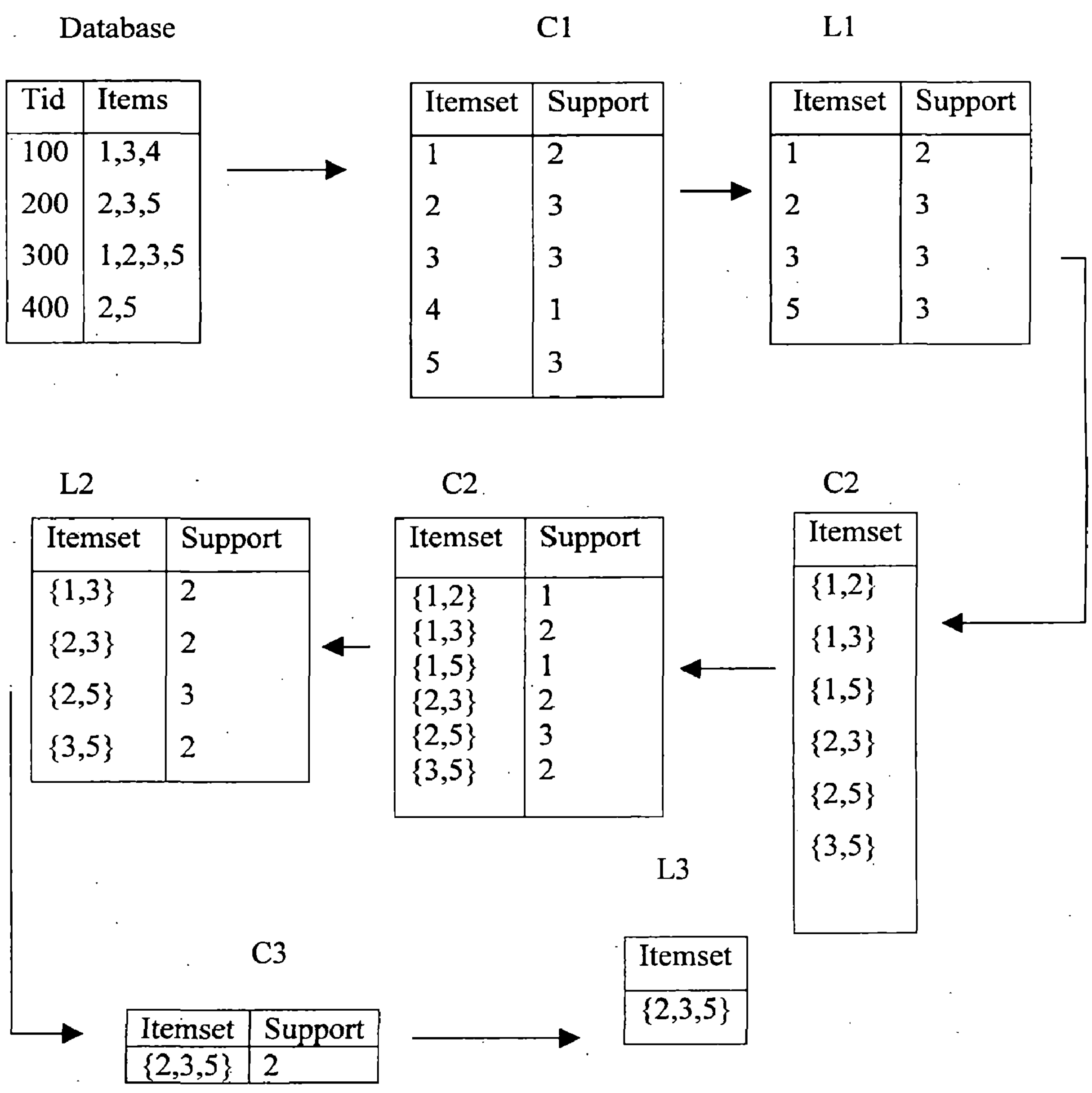
- Iteratively find frequent itemsets (the sets of items that have minimum support) with cardinality from 1 to *k* (k-itemset).
- Use the frequent itemsets to generate association rules.

2.6.4.1.3. Key Operations

There are two major operations in finding the frequent item sets:

- **Join:** joining L_{k-1} with itself generates C_k.
- **Prune:** Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset.

Apriori Example



Database

Tid	Items
100	1,3,4
200	2,3,5
300	1,2,3,5
400	2,5

C1

Itemset	Support
1	2
2	3
3	3
4	1
5	3

L1

Itemset	Support
1	2
2	3
3	3
5	3

C2

Itemset
{1,2}
{1,3}
{1,5}
{2,3}
{2,5}
{3,5}

C2

Itemset	Support
{1,2}	1
{1,3}	2
{1,5}	1
{2,3}	2
{2,5}	3
{3,5}	2

L2

Itemset	Support
{1,3}	2
{2,3}	2
{2,5}	3
{3,5}	2

C3

Itemset	Support
{2,3,5}	2

L3

Itemset
{2,3,5}

Fig.2.4: Apriori Example

2.6.4.1.4 Performance Evaluation

Here are the bottlenecks of Apriori algorithm:

- Huge candidate sets

  10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets

  To discover a frequent pattern of size 100, e.g., $\{a_1, a_2 \dots a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.

- Multiple scans of database

  Needs (n +1) scans, n is the length of the longest pattern.

The performance of Apriori algorithm drastically decreases when any of the maximal frequent itemsets becomes longer, because a maximal frequent itemset of size l implies the presence of (2l – 2) additional frequent itemsets (its nontrivial subsets) as well, such algorithms explicitly examine each of which. In data mining applications where items are correlated, maximum frequent itemsets could be long.

2.6.4.2 Enhancements Over Apriori Algorithm

After the initial algorithms proposed by Agrawal [10], other researchers have extensively studied the problem and a number of fast variants have been proposed. Agrawal has discussed how the algorithm for finding large itemsets may be speed up substantially by introducing a pruning approach, which reduces the size of the candidate C_k. This algorithm uses the pruning trick that all subsets of a large itemset must also be large. Thus, if some (k-1)-subset of an itemset I, i.e. the subset of C_k does not belong to L_{k-1}, then that itemset can be pruned from further consideration. This process of pruning eliminates the need for finding the support of the candidate itemset I.

Subsequent work on the large itemset method has concentrated on the following aspects [4]:

1. Improving the I/O costs by reducing the number of passes over the transaction database.
2. Improving the computational efficiency of the large itemset generation procedure.
3. Find the efficient parallel algorithm to mine association rules.

Here is a brief survey of the work done in each of the above categories.

2.6.4.2.1 AprioriTid Algorithm

The AprioriTid algorithm proposed by Agrawal and Srikant [5] has the additional property that the database is not used at all for counting the support of candidate itemsets after the first pass. Rather, an encoding of the candidate itemsets used in the previous pass is employed for this purpose. In later passes, the size of this encoding can become much smaller than the database, thus saving much reading effort.

The AprioriTid algorithm also uses the Apriori generation function (given in previous section) to determine the candidate itemsets before the pass begins. The interesting feature of this algorithm is that the database D is not used for counting support after the first pass. Rather the set C_k" is used for this purpose. Each member of the set C_k" is of the form <TID, {X_k}>, where each X_k is a potentially large k-itemset present in the transaction with identifier TID. For k=1, C_k" corresponds to the database D, although conceptually each item I is replaced by the itemset {I}. For k>1, the member of C_k" corresponding to transaction t is <t.TID, {c is the element of C_k | c contained in t}>. If a transaction does not contain any candidate k-itemset, then C_k" will not have an entry for this transaction. Thus, the number of entries in C_k" may be smaller than the number of transactions in the database, especially for large values of k. In addition, for large values of k, each entry may be smaller than the corresponding transaction because very few candidates may be contained in the transaction. However, for small values of k, each entry may be larger than the corresponding transaction because an entry in C_k includes all candidate k-itemsets contained in the transaction.

Consider the database given in Fig 2.5 and assume that minimum support is 2 transactions. Self-joining L_1 gives the candidate itemsets C_2. Then the support of candidates in C_2 is counted by iterating over the entries in C_1" and generate C_2". The first entry in C_1" is {{1}, {3}, {4}}, corresponding to transaction 100. The Ct corresponding to this entry t is {{1, 3}}, because {1, 3} is a member of C_2 and both ({1, 3}−{1}) and ({1, 3}−{3}) are members of t.set of itemsets. Self-joining L_2 gives C_3. Making a pass over the data with C_2" and C_3 generates C_3". Note that there is no entry in C_3"for the transactions with TIDs 100 and 400, since they do not contain any of the itemsets in C_3.

The candidate {2, 3, 5} in C_3 turns out to be large and is the only member of L_3. When C_4 is generated using L_3, it turns out to be empty, and so terminated.

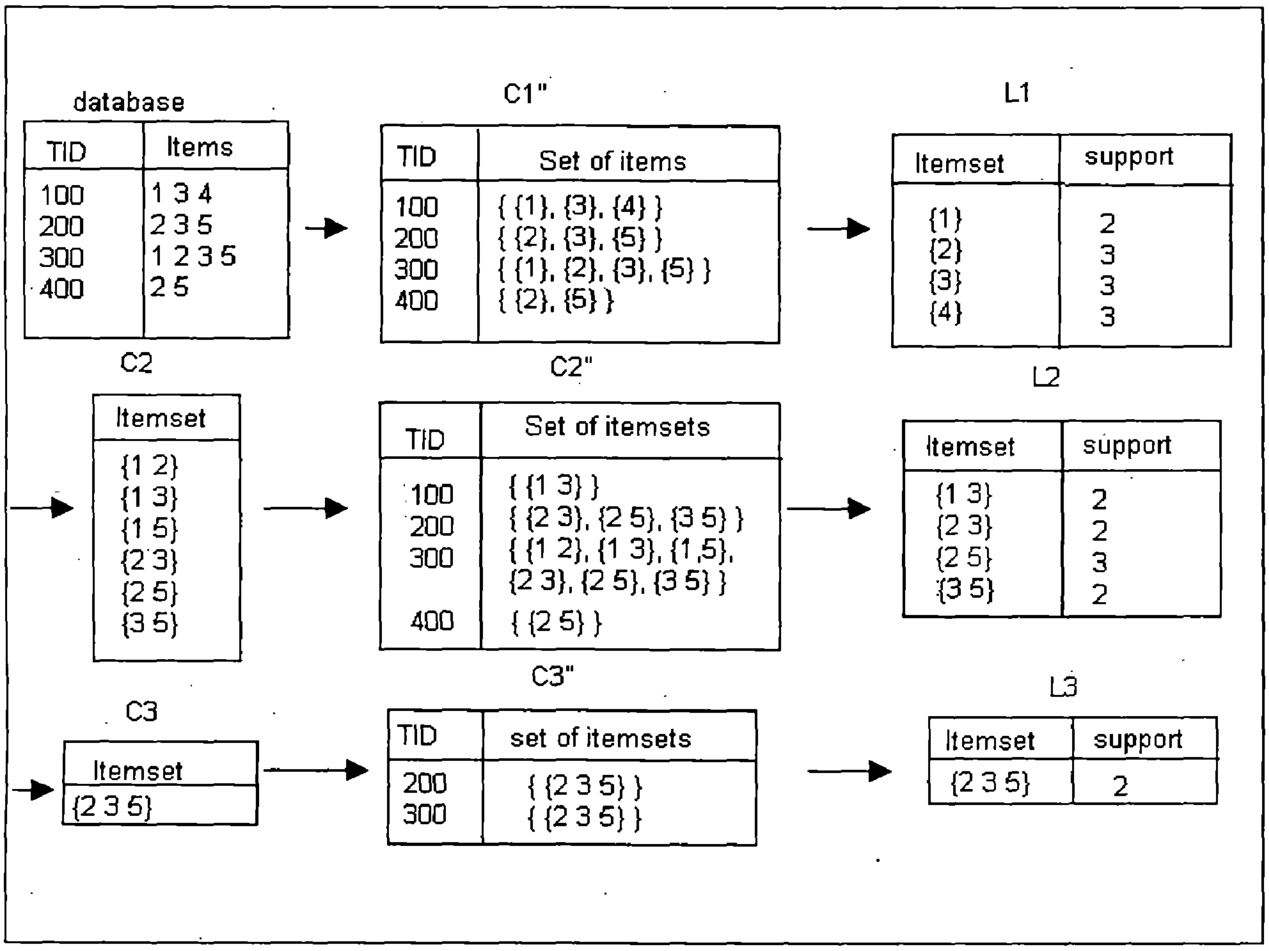


Fig2.5: AprioriTid Example [5]

In early passes Apriori performs better than AprioriTid algorithm but after that performance of AprioriTid drastically increases. As shown in Fig 2.6, For AprioriTid algorithm, in some initial passes the size of C" may be too large to fit in memory. Hence use Apriori algorithm for these passes and switch to AprioriTid algorithm when it expects that set C" will fit in memory at the end of pass.

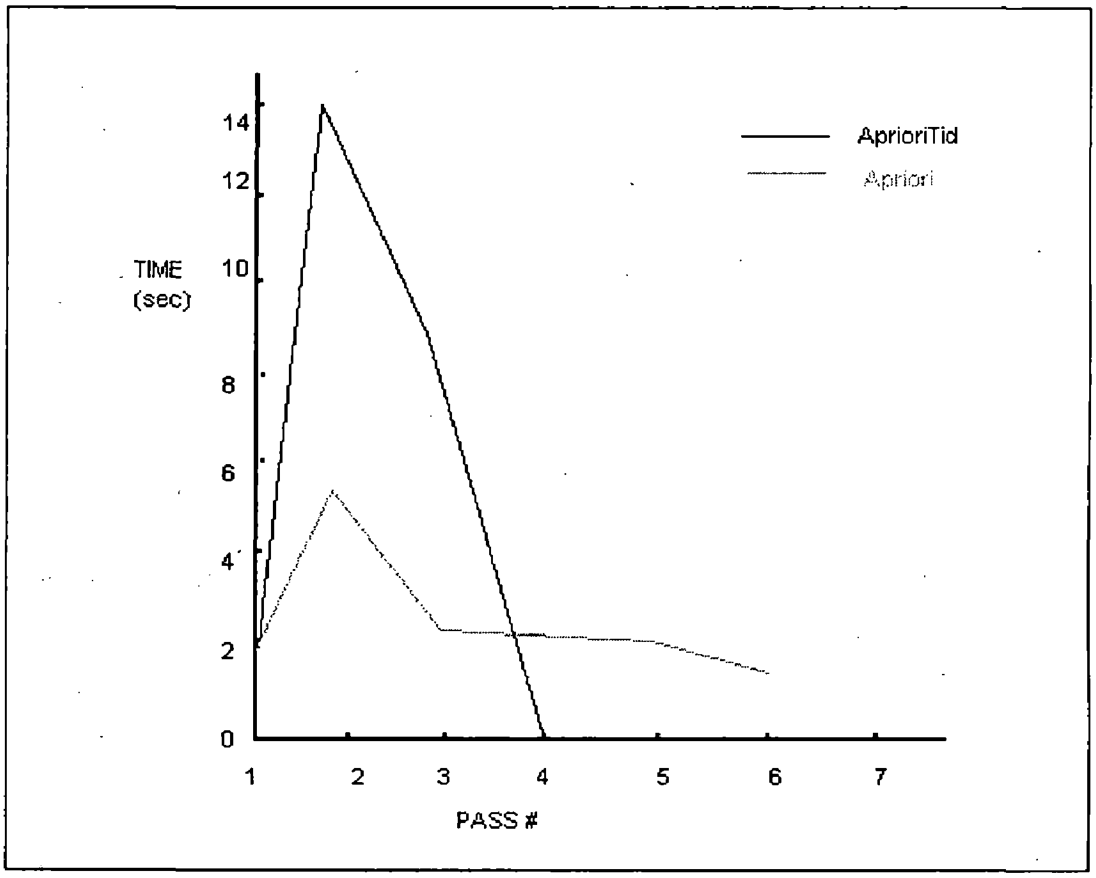


Fig2.6: Performance Comparison [5]

2.6.4.2.2 Hash-Based Algorithm

For efficiently finding large itemsets it was proposed by "Park" [4] [6]. It was observed that most of the time is spent in evaluating and finding large 2-itemsets. The algorithm of "Park" attempts to improve this approach by providing a hash based algorithm for quickly finding large 2-temsets. A hash based technique can be used to reduce the size of the candidate k-itemset, C_k, for k>1. Consider database given in Figure 2.7, when scanning each transaction in the database to generate the frequent 1-itemset, L_1, from the candidate 1-itemset in C_1, All the 2-itemsets can be generated for each transaction. Hash them into the different buckets of a hash table structure, and increase the corresponding bucket count. A 2-itemset, whose corresponding bucket count in the hash table is below the support threshold, cannot be frequent and thus should be removed from the candidate set. Such a hash table- based technique may substantially reduce the number of the candidate k-itemsets examined (especially when k=2).

TID	List of item IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Hash function
H (x, y)=((order of x)*10+(order of y mod 7))

Hash table H2 for candidate 2-itemset

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4

bucket contents	{I1.I4}
	{I3,I5}
		{I1,I5}
		{I1,I5}
			{I2,I3}
			{I2,I3}
			{I2,I3}
			{I2,I3}
				{I2,I4}
				{I2,I4}
					{I2,I5}
					{I2,I5}
						{I1,I2}
						{I1,I2}
						{I1,I2}]
						{I1,I2}
							{I1,I3}
							{I1,I3}
							{I1,I3}
							{I1,I3}

Fig.2.7: Hash table for Candidate 2-Itemsets [2]

Brin [4] proposed a method for large itemset generation, which reduces the number of passes over the transaction database by counting some (k+1)-itemsets in parallel with counting k-itemsets. In most previously proposed algorithms for finding large itemsets, the support for a (k+1)-itemset was counted after k-itemsets have already been generated. In this work, it was proposed that one could start counting a (k+1)-itemset as soon as it was suspected that this itemset might be large. Thus, the algorithm could start counting for (k+1)-itemsets much earlier than completing the counting of k-itemsets. The total number of passes required by this algorithm is usually much smaller than the maximum size of a large itemset.

2.6.4.2.3 Partitioning Algorithm

Savasere [7] proposed an algorithm for finding large itemsets by dividing the database into n partitions. The size of each partition is such that the set of transactions can be maintained in main memory. Then, large itemsets are generated separately for each partition. Let LP_i be the set of large itemsets associated with the ith partition. Then, if an itemset is large, then it must be the case that it must belong to at least one of LP_i for i is the element of {1...k}. Now, the support of the candidates $U_i^k{}_{=1}LP_i$ can be counted in order to find the large itemsets. This method requires just two passes over the transaction database in order to find the large itemsets. The approach described above is highly parallelizable, and has been used to generate large itemsets by assigning each partition to a processor. At the end of the each iteration of the large itemset method the processors need to communicate with one another in order to find the global counts of the candidate k-itemsets. Often, this communication process may impose a substantial bottleneck on the running time of the algorithm. In other cases, the time taken by the individual processors in order to generate the processor-specific large itemsets may be the bottleneck.

A common feature of most of the algorithms reviewed above and proposed in the literature is that most such researches are variations on the "bottom-up theme" proposed by the Apriori algorithm. For databases in which the itemsets may be long, these algorithms may require substantial computational effort. Consider for example a database in which the length of the longest itemset is 40. In this case, there are 2^{40} subsets of this single itemset, each of which would need to be validated against the transaction database. Thus, the success of the above algorithms critically relies on the fact that the lengths of the frequent patterns in the database are typically short.

2.6.4.2.4 Look-Ahead Algorithm

Bayardo [8] has proposed an interesting algorithm for itemset generation very recently. This algorithm uses clever "look-ahead" techniques in order to identify longer patterns earlier on. The subsets of these patterns can then be pruned from further consideration. Initial computational results indicate that the algorithm can lead to substantial performance improvements over the Apriori method.

2.6.4.2.5 Transaction Reduction Method

Transaction reduction method [2] is used to reduce the number of transactions. A transaction that does not contain any frequent k-itemset can't have (k+1)-itemset, hence can be pruned.

2.6.4.2.6 AIS and SETM Algorithms

The problem of association rule mining was first introduced in [5]. An algorithm called AIS was given for discovering the frequent set. SETM algorithm [9] was later designed to use only standard SQL commands to find the frequent set. The Apriori algorithm [10], described above, performs much better than AIS and SETM.

2.6.4.2.7 The OCD Algorithm

It is worth adding, that concurrently with the Apriori algorithm, OCD algorithm uses the same closure property to eliminate candidates [11].

2.6.4.2.8 DHP and Partition Algorithm

DHP algorithm [12] extended the Apriori algorithm by introducing a hash filter for counting the upper bound of the support of candidates in the next pass. Some candidates can be pruned before reading the database in the next pass.

Partition algorithm [13] proposed to divide the database into equal sized partitions. Each partition is processed independently to produce a *local frequent set* for that partition. After all local frequent sets are discovered, their union, the *global candidate se*t, forms a superset of the actual frequent set. The database is then read again to produce the actual support for the global candidate set. The entire process takes only two (read) passes.

2.6.4.2.9 Sampling Algorithm

Sampling Algorithm [14] proposed to consider first (small) samples of the database and discover an *approximate frequent set* by using a standard bottom-up approach algorithm. The approximate frequent set is then verified against the entire database. False frequent itemsets need to be removed and missing frequent itemsets need to be recovered.

2.6.4.2.10 A-Random-MFS, DIC, and MaxClique Algorithms

A-Random-MFS algorithm [17] is a randomized algorithm for discovering the maximum frequent set. A single run of the algorithm cannot guarantee correct results. A complete algorithm requires repeatedly calling the randomized algorithm until no new maximal frequent itemset can be found.

Dynamic itemset counting (DIC) algorithm [16] combines candidates of different lengths into one pass. The database is divided into partitions of equal size. In each pass, after the first I partitions are read, some itemsets containing up to I +1 items may become candidates based on the database partitions read so far.

MaxClique [17] used a hybrid traversal, which contains a look-ahead phase followed by a pure bottom-up phase. The look-ahead phase consists of extending the frequent 2-itemsets until the extended itemset becomes infrequent. After the look-ahead phase, an Apriori-like traversal is executed.

One of the most important differences between MaxClique and Hybrid approach is that MaxClique only looks ahead at some long candidate itemsets during the initialization stage (in the second pass). In contrast, the Hybrid algorithm repeatedly maintains the upper bound of the frequent itemsets (TOPC) throughout the entire process. The look-ahead candidate itemsets are dynamically adjusted based on all available information discovered so far. In fact, the TOPC is the most accurate approximation one can get while no additional knowledge of the data is available.

Another important difference is that MaxClique used a bottom-up approach to calculate the look-ahead candidate itemsets. Conceptually, it keeps applying Apriori-gen until no more candidates can be generated. In contrast, Hybrid approach uses a top-down approach. It updates the TOPC only when a new infrequent itemset is discovered.

Ignoring implementation details, MaxClique can be viewed as a special case of Hybrid Search.

2.6.4.2.11 Max-Miner

This work is inspired by Max-Miner algorithm. *Max-Miner* algorithm [8] was recently proposed to discover the maximum frequent set. This algorithm partitions the candidate set into groups with the same prefix. Like Hybrid Search, it looks ahead at some long candidate itemsets throughout the search. The main difference is the long candidate itemsets that it examines. Max-Miner looks ahead at longest itemsets that can be constructed from every group. A frequency heuristic is used to reorder the items such that the most frequent items appear in the most candidate groups.

After preliminary comparison with the Max-Miner from the algorithmic point of view it is felt that Max-Miner and Hybrid Search could be complementary. One of the possibilities is to run Max-Miner in the first few passes and switch to Hybrid Search for the later passes.

Chapter 3

Hybrid Algorithm – A collective strength

Typical algorithms for mining frequent itemsets operate in a bottom-up, breadth-first search direction. The computation starts from frequent 1-itemsets (the minimum length frequent itemsets) and continues until all maximal (length) frequent itemsets are found. During the execution, every frequent itemset is explicitly considered. Such algorithms perform well when all maximal frequent itemsets are short. However, performance drastically decreases when some of the maximal frequent itemsets are relatively long. This work is an attempt to develop a new algorithm, which combines both bottom-up and the top-down approach.

The primary search direction is still bottom-up, but a restricted search is also conducted in the top-down direction. This search is used only for maintaining and updating a new data structure, that is called TOPC. It is used to prune early candidates that would be normally encountered in the bottom-up search. A very important characteristic of the algorithm is that it does not require explicit examination of every frequent itemset. Therefore the algorithm performs well even when some maximal frequent itemsets are long. As its output, the algorithm produces the BOTC, i.e., the set containing all maximal frequent itemsets, thus specifying immediately all frequent itemsets.

The improvement in performance can be up to several orders of magnitude, compared to the Apriori algorithm.

The problem is formulated as follows: Given a large database of sets of items (Representing market basket data, alarm signals, etc.), discover all frequent itemsets (sets of items), where a frequent itemset is one that occurs in at least a user-defined percentage (minimum support) of the database. Depending on the semantics attached to the input database, the frequent itemsets, and the term "occurs," we get the key components of different data mining problems such as the discovery of association rules.

The performance of Apriori algorithm drastically decreases when *any* of the maximal frequent itemsets becomes longer, because a maximal frequent itemset of size l implies the presence of (2l-2) additional frequent itemsets (its nontrivial subsets) as well, such algorithms explicitly examine each of which. In data mining applications where items are correlated, maximum frequent itemsets could be long.

Therefore, instead of examining all the frequent itemsets, an alternative approach might be to "shortcut" the process and attempt to search for maximal frequent itemsets "more directly," as they immediately specify all frequent itemsets.

The search for the maximum frequent set can proceed from the 1-itemsets to *n*-itemsets (bottom-up) or from the *n*-itemsets to 1-itemsets (top-down). But Hybrid approach searches for the TOPL from *both bottom-up* and *top-down directions*. It performs well even when the maximal frequent itemsets are long.

The bottom-up search is similar to Apriori algorithm. However, the top-down search is different. It is implemented efficiently by introducing an auxiliary data structure, the TOPC, as explained later. By incorporating the computation of the TOPC in algorithm, it is possible to efficiently approach the TOPL from both top-down and bottom-up directions. Unlike the bottom-up search that goes up one level in each pass, the TOPC can help the computation "move down" many levels in the top-down direction in one pass.

This algorithm not only reduces the number of passes of reading the database but also reduces the number of candidates (for whom support is counted). In such cases, eliminating the candidates that are subsets of maximal frequent itemsets found in the TOPC reduces both I/O time and CPU time.

3.1 Frequent Itemset and its Properties

3.1.1 The Maximum Frequent Set

Among all the frequent itemsets, some will be *maximal frequent itemsets*: they have no proper supersets that are themselves frequent. The TOPL is the set of all the maximal frequent itemsets. The problem of discovering the frequent set can be reduced to the problem of discovering the TOPL. The TOPL immediately specifies of frequent itemsets; these are precisely the non-empty subsets of its elements. The TOPL forms a border between frequent and infrequent sets.

3.1.2 Properties

Two properties can be used to classify some of the unclassified itemsets:

- **Property 1:** If an itemset is infrequent, all it supersets must be infrequent, and they need not be examined further
- **Property 2:** If an itemset is frequent, all its subsets must be frequent, and they need not be examined further

3.2 Discovering Frequent Itemsets

In general, it is possible to search for the maximal frequent itemsets either bottom-up or top-down. If all maximal frequent itemsets are expected to be short (close to 1 in size), it seems efficient to search for them bottom-up. If all maximal frequent itemsets are expected to be long (close to n in size) it seems efficient to search for them top-down.

Here a realization is sketched of the most commonly used approach of discovering the frequent itemsets: a bottom-up approach. It consists of repeatedly applying a *pass*, itself consisting of two steps. At the end of pass k all frequent itemsets of size k or less have been discovered. As the *first step* of pass (k +1), itemsets of size (k +1) each having two frequent k-subsets with the same first (k –1) items are generated. Itemsets that are supersets of infrequent itemsets are pruned (and discarded), as of course they are infrequent (by property 1). The remaining itemsets form the set of candidates for this pass. As the second step, the support of the candidates is computed (by reading the database), and they are classified as either frequent or infrequent.

Example 1

Consider a database containing five distinct items, 1, 2, 3, 4, and 5. There are four transactions in this database: {1,2,3,4,5}, {1,3}, {1,2}, and {1,2,3,4}. The minimum support is set to 0.5. Figure 3.1 shows an example of this bottom-up approach. All five 1-itemsets ({1}, {2}, {3}, {4}, {5}) are candidates in the first pass. After the support counting phase, the 1-itemset {5} is determined to be infrequent. Property 1 need not consider all the supersets of {5}. So the candidates for the second pass are {1,2}, {1,3}, {1,4}, {2,3}, {2,4}, {3,4}. The same procedure repeats until all the maximal frequent itemsets are obtained in this example, only one: {1,2,3,4}.

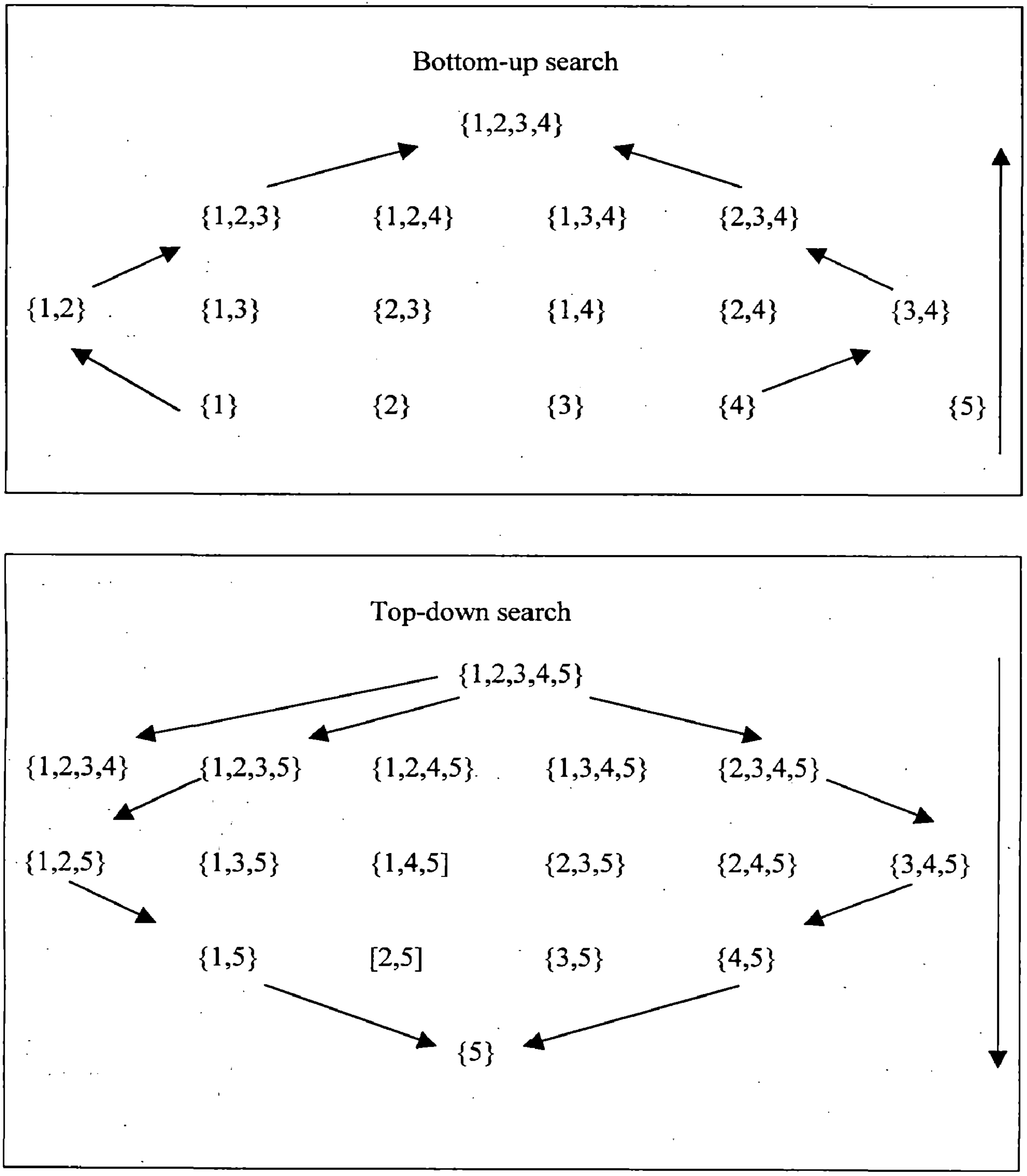


Fig.3.1: One Way Search

In this bottom-up approach, *every* frequent itemset must have been a candidate at some pass and is therefore also explicitly considered. When some maximal frequent itemsets happen to be long, this method will be inefficient. In such a case, it might be more efficient to search for the long maximal frequent itemsets using a top-down approach.

A top-down approach starts with the single *n*-itemset and decreases the size of the candidates by one in every pass. When a *k*-itemset is determined to be infrequent, all of its (*k*-1)-subsets will be examined in the next pass. However, if a *k*-itemset is frequent, then all of its subsets must be frequent and need not be examined (by Property 2).

Example 2

Figure 3.1 shows example of two-way search. Consider the same database as the previous example. The 5 -itemset {1,2,3,4,5} is the only candidate in the first pass. After the support counting phase, it is infrequent. The candidates for the second pass are all the 4-subsets of itemset {1,2,3,4,5}. In this example, itemset {1,2,3,4} is frequent and all the others are infrequent. By Property 2, all subsets of {1,2,3,4} are frequent (but not maximal) and need not be examined. The same procedure repeats until all maximal frequent itemsets are obtained (i.e., after all infrequent itemsets are visited)..

In this top-down approach, *every* infrequent itemset is explicitly examined. As shown in Figure given above, every infrequent itemset (itemset {5} and its supersets) needs to be visited before the maximal frequent itemsets are obtained. Note that, in a "pure" bottom-up approach, only Property 1 above is used to prune candidates. This is the technique that Apriori algorithm uses to decrease the number of candidates. In a "pure" top-down approach, only Property 2 is used to prune candidates.

3.3 Hybrid Approach- A Collective Strength

The aim of this work is to reduce the number of candidates and the number of passes in the process of association rules mining

As discussed in the last section, the bottom-up approach is good for the case when *all* maximal frequent itemsets are short and the top-down approach is good when *all* maximal frequent itemsets are long. If some maximal frequent itemsets are long and some are short, then both one-way search approaches will not be efficient. To design an algorithm that can efficiently discover both long and short maximal frequent itemsets, one might think of simply running both bottom-up and top-down programs at the same time. It is possible to do much better than that. Recall that the bottom-up approach described above uses only Property 1 to reduce the number of candidates and the top-down approach uses only Property 2 to reduce the number of candidates. In Hybrid approach both top-down and the bottom-up searches are combined. That synergistically relies on *both* properties to prune candidates. A key component of the approach is the use of information gathered in the search in one direction to prune more candidates during the search in the other direction. If some maximal frequent itemset is found in the top-down direction, then this itemset can be used to eliminate (possibly many) candidates in the bottom-up direction. The subsets of this frequent itemset can be pruned because they are frequent (Property 2). Of course, if an infrequent itemset is found in the bottom-up direction, then it can be used to eliminate some candidates in the top-down direction (Property 1). This "two-way search approach" can fully make use of both properties and thus speed up the search for the maximum frequent set.

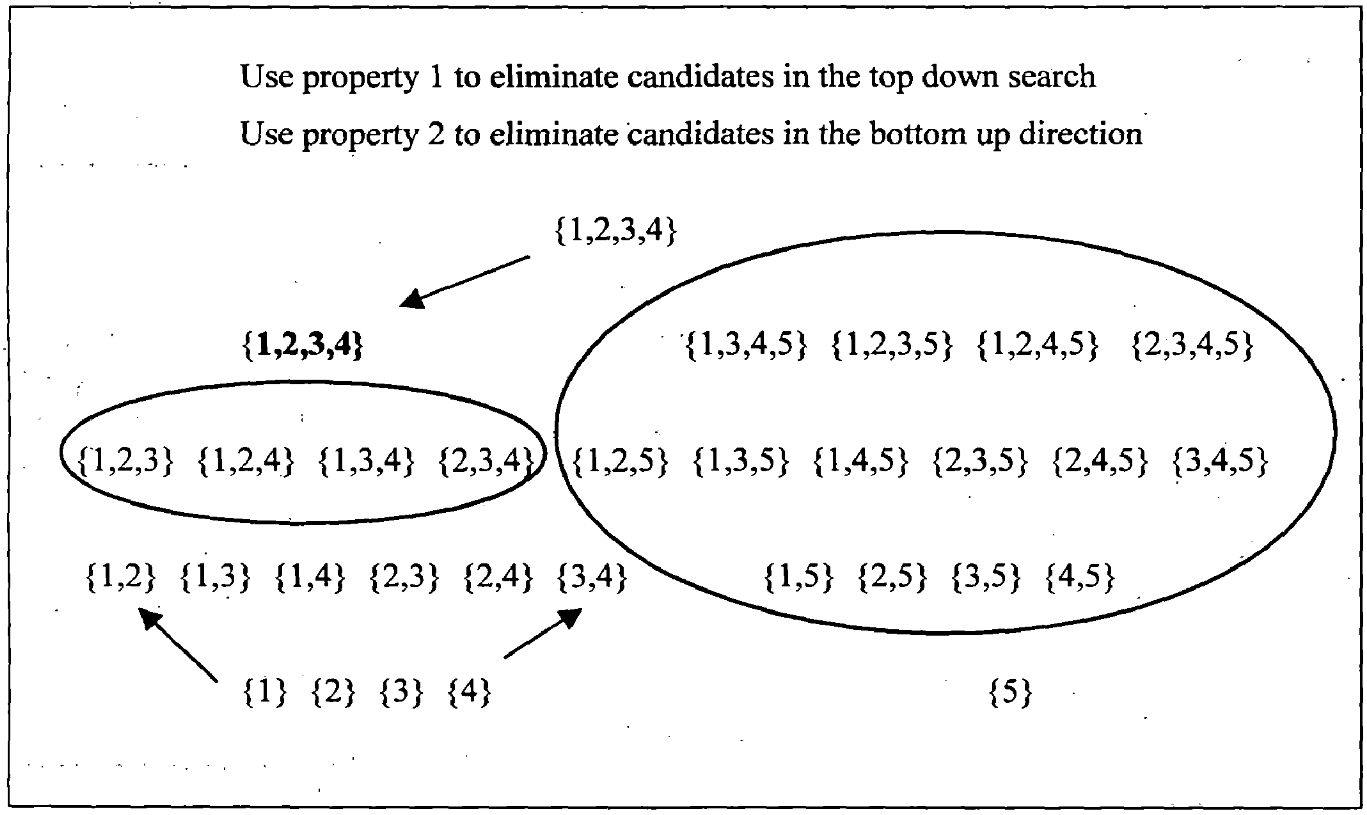


Fig.3.2: Two way Search

Example 3

Consider two way search example given above in Fig.3.2, In the first pass, all five 1-itemsets are the candidates for the bottom-up search and the 5-itemset {1,2,3,4,5} is the candidate for the top-down search. After the support counting phase, infrequent itemset {5} is discovered by the bottom-up search and this information is shared with the top-down search. This infrequent itemset {5} not only allows the bottom-up search to eliminate its supersets as candidates but also allows the top-down search to eliminate its supersets as candidates in the second pass. In the second pass, the candidates for the bottom-up search are {1,2}, {1,3}, {1,4}, {2,3}, {2,4}, and {3,4}. Itemsets {1,5}, {2,5}, {3,5}, and {4,5} are not candidates, since they are supersets of {5}. The only candidate for the top-down search in the second pass is {1,2,3,4}, since all the other 4-subsets of {1,2,3,4,5} are supersets of {5}. After the second support counting phase, {1,2,3,4} is discovered to be frequent by the top-down search. This information is shared with the bottom-up search. All of its subsets are frequent and need not be examined. In this example, itemsets {1,2,3}, {1,2,4}, {1,3,4}, and {2,3,4} will not be candidates for bottom-up or top-down searches. After that, the program can terminate, since there are no candidates for either bottom-up or top-down searches.

In this example, the number of candidates considered, was smaller than required by either bottom-up or top-down search. In addition to this fewer passes are needed to read the database than either bottom-up or top-down searches. The "pure" bottom-up approach would have taken four passes and the "pure" top-down approach would have taken five passes for this database while Hybrid approach takes *only* two. In fact, this hybrid approach will always use at most as many passes as the minimum 10 of the passes used by bottom-up approach and top-down approach. Reducing the number of candidates is of critical importance for the efficiency of the frequent set discovery process, since the cost of the entire process comes from reading the database (I/O time) to generate the supports of candidates (CPU time) and the generation of new candidates (CPU time). The support counting of the candidates is the most expensive part. Therefore, the number of candidates dominates the entire processing time. Reducing the number of candidates not only can reduce the I/O time but also can reduce the CPU time, since fewer candidates need to be counted and generated. Therefore, it is important that Hybrid Search reduces both the number of candidates and the number of passes. A realization of this two-way search algorithm is discussed next.

The computation of Hybrid algorithm follows the bottom-up breadth-first search approach. It is based on the Apriori and Max Miner algorithms, and for greatest ease of exposition it is presented as a modification to the Apriori algorithm.

Briefly speaking, in each pass, in addition to counting supports of the candidates in the bottom-up direction, the algorithm also counts supports of the itemsets in the top-down search. This will help in pruning candidates, but will also require changes in candidate generation, as explained later.

Consider a pass *k*, during which, in the bottom-up direction, itemsets of size *k* are to be classified. If, during the top-down direction some itemset that is an element of the TOPC of cardinality greater than *k* is found to be frequent, then all its subsets of cardinality *k* can be pruned from the set of candidates considered in the bottom-up direction in this pass. They, and their supersets will never be candidates throughout the rest of the execution, potentially improving performance. But of course, as the maximum frequent set is ultimately computed, they "will not be forgotten."

Similarly, when a new infrequent itemset is found in the bottom-up direction, the algorithm will use it to update the TOPC. The subsets of the TOPC must not contain this infrequent itemset.

Fig 3.3 given below conceptually shows the combined two-way search. The TOPC is initialized to contain a single element, the itemset of cardinality *n* containing all the elements of the database. As an example of its utility, consider the first pass of the bottom-up search. If some *m* 1-itemsets are infrequent after the first pass (after reading the database once), the TOPC will have one element of cardinality *n-m*. Removing the m infrequent items from the initial element of the TOPC generates this itemset. In this case, the top-down search goes down *m* levels in one pass. In general, unlike the search in the bottom-up direction, which goes up one level in one pass, *the top-down search can go down many levels in one pass.*

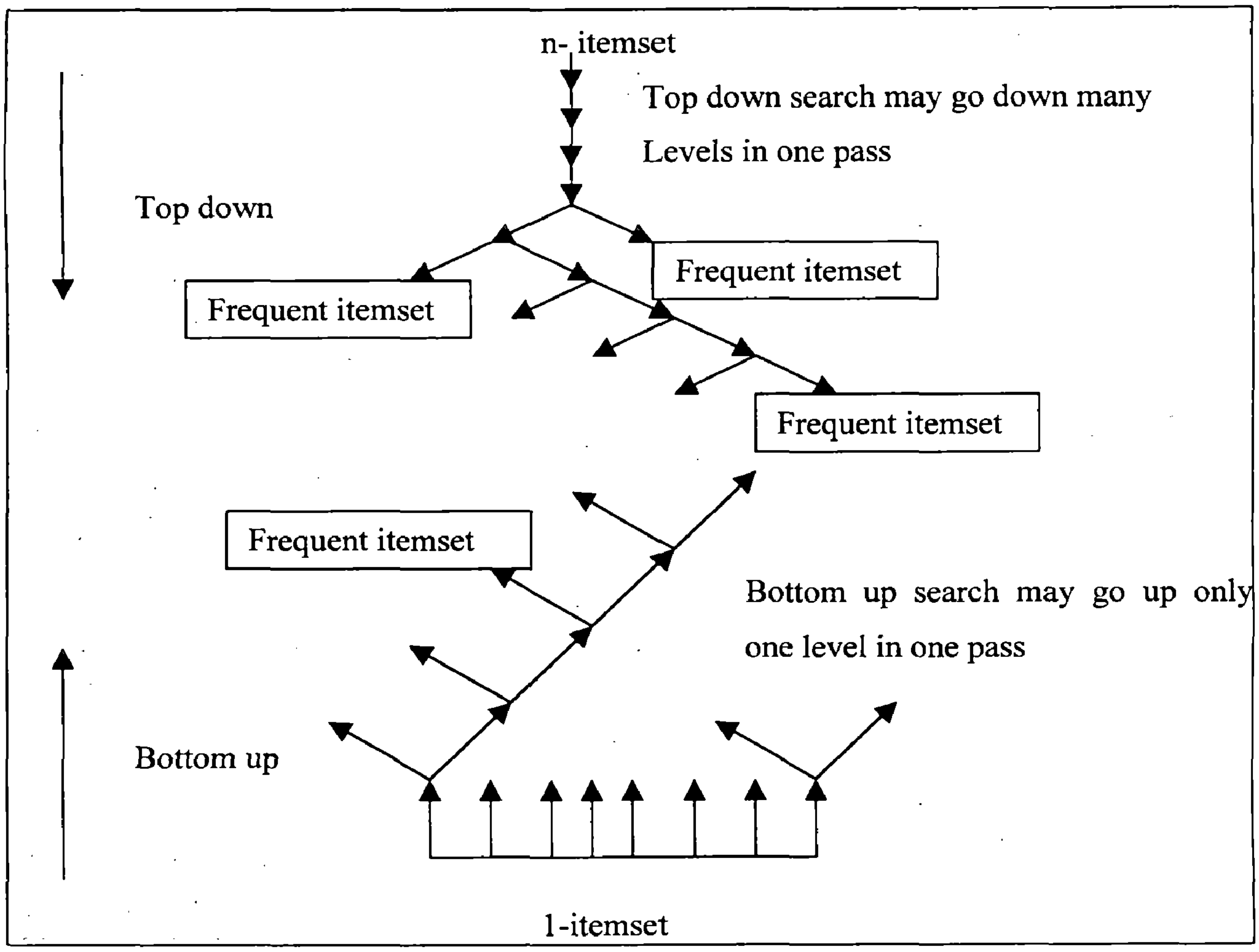

n- itemset
Top down search may go down many Levels in one pass
Top down
Frequent itemset
Frequent itemset
Frequent itemset
Frequent itemset
Bottom up search may go up only one level in one pass
Bottom up
1-itemset

Fig.3.3: Working of Two Way Search

Notice that the bottom up and the top down searches do not proceed in a symmetrical fashion. The reason is that by a general assumption there are no extremely long frequent itemsets. If this assumption is not likely to hold, one can easily reverse the roles of the searches in the two directions. By using the TOPC, it will be possible to discover some maximal frequent itemsets in early passes. This early discovery of the maximal frequent itemsets can reduce the number of candidates and the passes of reading the database, which in turn can reduce the CPU time and I/O time. This is especially significant when the maximal frequent itemsets discovered in the early passes are long.

To reduce the number of database scans, transaction having k items can be removed or marked, at the end of pass k. So the number of transactions are reduced for further iterations.

Chapter 4

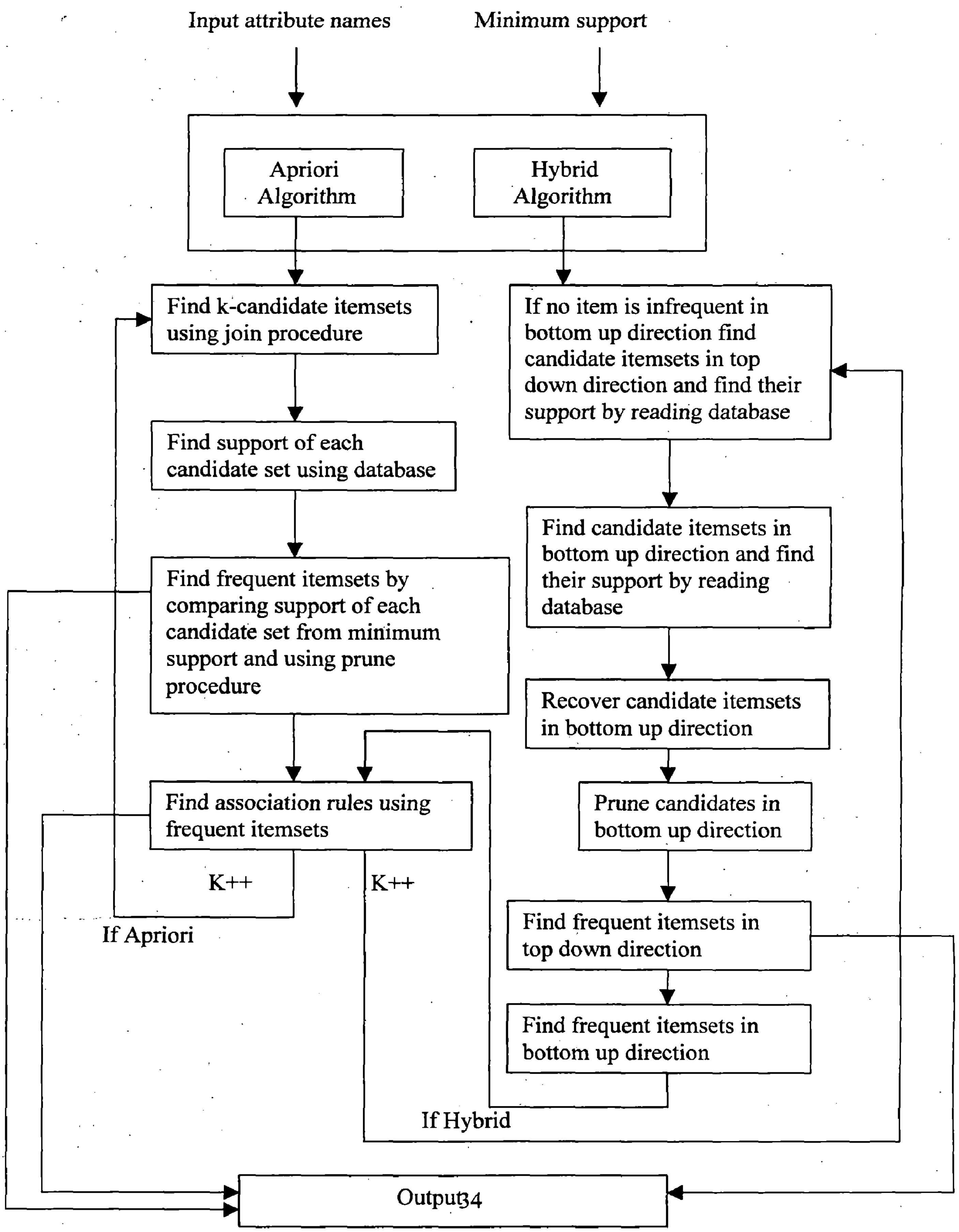
System Design

Having analyzed the problem and identified the pre-processing operations that are required of the software to be developed, the following solution is proposed:

The software takes as input the name of attributes to be correlated and minimum support. Then candidate itemsets are generated using Apriori algorithm or Hybrid algorithm. Itemsets having support greater than minimum support are called frequent itemsets. These frequent itemsets are used to generate association rules, which are output of the software. Database used in this software is designed in oracle, having transaction IDs and items.

Figure 4.1 presents the diagrammatic representation of the design discussed above:

Figure 4.1: Diagrammatic representation of the proposed design.


Input attribute names
Minimum support
Apriori Algorithm
Hybrid Algorithm
Find k-candidate itemsets using join procedure
Find support of each candidate set using database
Find frequent itemsets by comparing support of each candidate set from minimum support and using prune procedure
Find association rules using frequent itemsets
K++
K++
If Apriori
If no item is infrequent in bottom up direction find candidate itemsets in top down direction and find their support by reading database
Find candidate itemsets in bottom up direction and find their support by reading database
Recover candidate itemsets in bottom up direction
Prune candidates in bottom up direction
Find frequent itemsets in top down direction
Find frequent itemsets in bottom up direction
If Hybrid
Output

To make Hybrid approach effective two issues are addressed: First, how to update the TOPC efficiently? Second, once the subsets of the maximal frequent itemsets found in the TOPC are removed, how to generate the correct candidate set for the subsequent passes in the bottom-up direction?

4.1. Updating the TOPC Efficiently

Consider some itemset *Y* that has been "just" classified as infrequent. It will be a subset of one or more itemsets in the TOPC, and it is required to update the TOPC such that its subsets no longer contain *Y*. To update the TOPC, the following process will be done for every superset of *Y* that is in the TOPC. Every such itemset (say *X*) is replaced by $|Y|$ itemsets, each obtained by removing from *X* a single item (element) of *Y*. Such newly generated itemset is added to the TOPC only when it is not already a subset of any itemset in the TOPC.

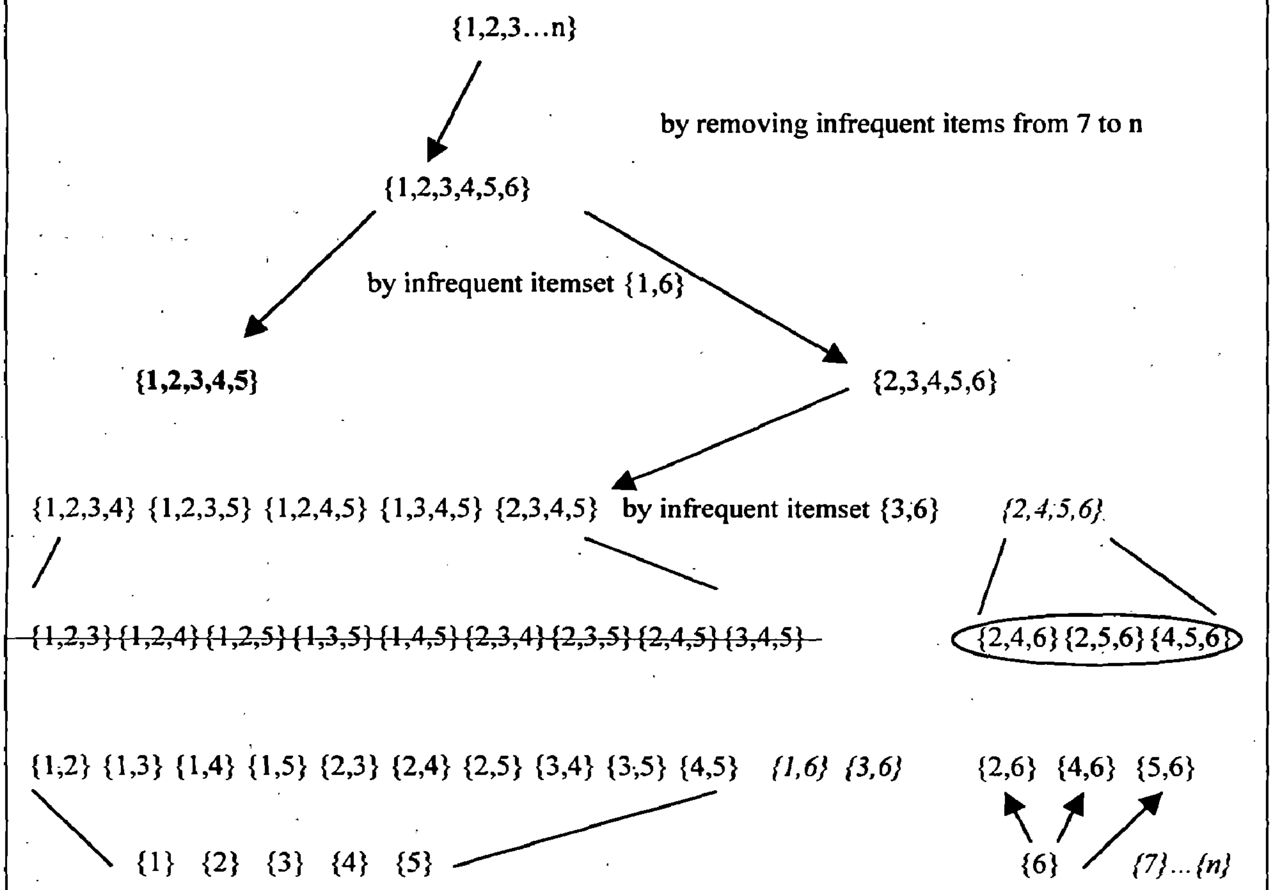


Fig. 4.2: Hybrid Search

Example 1

Consider Hybrid search given in fig 4.2, suppose {{1,2,3,4,5,6}} is the current ("old") value of the TOPC and two new infrequent itemsets {1,6} and {3,6} are discovered. Consider first the infrequent itemset {1,6}. Since the itemset {1,2,3,4,5,6} (element of the TOPC) contains items 1 and 6, one of its subsets will be {1,6}, by removing item 1 from itemset 13.

From {1,2,3,4,5,6}, 2,3,4,5,6} is found, and by removing item 6 from itemset {1,2,3,4,5,6}, {1,2,3,4,5} is found. After considering itemset {1,6}, the TOPC becomes {{1,2,3,4,5}, {2,3,4,5,6}}. Itemset {3,6} is then used to update this TOPC. Since {3,6} is a subset of {2,3,4,5,6}, two itemsets {2,3,4,5} and {2,4,5,6} are generated to replace {2,3,4,5,6}. Itemset {2,3,4,5} is a subset of itemset {1,2,3,4,5} in the new TOPC, and it will not be added to the TOPC. Therefore, the TOPC becomes {{1,2,3,4,5}, {2,4,5,6}}.

4.2 New Candidate Generation Algorithms

As discussed previously a preliminary candidate set will be generated after the *join* procedure is called. In Hybrid algorithm, after a maximal frequent itemset is added to the TOPL, all of its subsets in the frequent set (computed so far) will be removed. The example shows that if the original *join* procedure of the Apriori-gen algorithm is applied, some of the needed itemsets could be missing from the preliminary candidate set. Consider Fig 4.2 given above suppose that the original frequent itemset L_3 is {{1,2,3}, {1,2,4}, {1,2,5}, {1,3,4}, {1,3,5}, {1,4,5}, {2,3,4}, {2,3,5}, {2,4,5}, {2,4,6}, {2,5,6}, {3,4,5}, {4,5,6}}. Assume itemset {1,2,3,4,5} in the TOPC is determined to be frequent. Then all 3-itemsets of the original frequent set L_3 will be removed from it by Hybrid algorithm, except for {2,4,6}, {2,5,6}, and {4,5,6}. Since the Apriori-gen algorithm uses a (k - 1)-prefix test on the frequent set to generate new candidates, and no two itemsets in the current frequent set {{2,4,6}, {2,5,6}, {4,5,6}} share a 2-prefix, no candidate will be generated by applying the *join* procedure on this frequent set. However, the correct preliminary candidate set should be {{2,4,5,6}}. Based on the above observation, some missing candidates need to be recovered.

4.3 New Preliminary Candidate Set Generation Procedure

In new preliminary candidate set generation procedure, the *join* procedure of the Apriori-gen algorithm is first called to generate a temporary candidate set, which might be incomplete. In such a case, a *recovery* procedure will be called to recover the missing candidates. All missing candidates can be obtained by restoring some itemsets to the current frequent set. The restored itemsets are extracted from the TOPL of the current pass, which implicitly maintains all frequent itemsets discovered so far. The first group of itemsets that needs to be restored contains those k-itemsets that have the same (k-1)-prefix as some itemset in the current frequent set. Consider then in pass k, an itemset X in the TOPL and an itemset Y in the current frequent set such that $|X| > k$. Suppose that the first (k-1) items of Y are in X and the $(k\text{-}1)^{st}$ item of Y is equal to the j^{th} item of X. The k-subsets of X is obtained that have the same (k-1)-prefix as Y by taking one item of X that has an index greater than j and combining it with the first (k-1) items of Y, thus getting one of these k-subsets. After these k-itemsets are found, candidates are recovered by combining them with itemset Y.

Example 2

Consider Hybrid search given in Fig 4.2, the TOPL is {{1,2,3,4,5}} and the current frequent set is {{2,4,6}, {2,5,6}, {4,5,6}}. The only 3-subset of {{1,2,3,4,5}} that needs to be restored for itemset {2,4,6} to generate a new candidate is {2,4,5}. This is because it is the only subset of {{1,2,3,4,5}} that has the same length and the same 2-prefix as itemset {2,4,6}. By combining {2,4,5} and {2,4,6}, missing candidate {2,4,5,6} is recovered. No itemsets need to be restored for itemsets {2,5,6} and {4,5,6}.

The second group of itemsets that need to be restored consists of those k-subsets of the TOPL having the same (k-1)-prefix but having no common superset in the TOPL. A similar *recovery* procedure can be applied after they are restored.

Chapter 5

Implementation

The project will use Windows 9x/NT as the platform, C as programming language and Oracle Pro C as precompiler to access database. Some important procedures used in implementation are given below:

5.1 Updating the TOPC Efficiently

```
Algorithm: TOPC-gen procedure
Input: Old TOPC and the infrequent set $GARBAGE_k$ found in pass k
Output: New TOPC
1. For all itemsets s, the element of $GARBAGE_k$
2. For all itemsets m, the element of TOPC
3. If s is a subset of m
4. TOPC = TOPC \ {m}
5. For all items e, element of itemset s
6. If {m \{e}} is not a subset of any itemset in the TOPC
7. TOPC = TOPC U {m \ {e}}
8. Return TOPC
```

5.2 Recovery Procedure

In new preliminary candidate set generation procedure, the *join* procedure of the Apriori-gen algorithm is first called to generate a temporary candidate set, which might be incomplete. In such a case, a *recovery* procedure will be called to recover the missing candidates.

All missing candidates can be obtained by restoring some itemsets to the current frequent set. The restored itemsets are extracted from the TOPL of the current pass, which implicitly maintains all frequent itemsets discovered so far.

```
Algorithm: The recovery procedure
Input: $C_{k+1}$ from join procedure, $L_k$, and current TOPL
Output: a complete candidate set $C_{k+1}$
1. For all itemsets l in $L_k$
2. For all itemsets m in TOPL
3. If the first (k-1) items in l are also in m
4. /* Suppose m.item j = l.item ( k –1) */
5. For i from (j +1) to m
6. $C_{k+1} = C_{k+1}$ U {{l.item 1, l.item 2,..., l.item k, m.item i }}
```

5.3 New Candidate Generation Algorithm

In summary, candidate generation process contains three steps as described below.

```
Algorithm: New candidate generation procedure
Input: $L_k$, current TOPC, and current TOPL
Output: new candidate set $C_{k+1}$
1. Call the join procedure as in the Apriori algorithm
2. Call the recovery procedure if necessary
3. Call the prune procedure
```

5.4 The Basic Hybrid-Search Algorithm

Here is the complete algorithm, *The Hybrid-Search Algorith*m, which relies on the combined approach for determining the maximum frequent set.

```
Algorithm: The Hybrid-Search algorithm
Input: a database and a user-defined minimum support
Output: TOPL, which contains all maximal frequent itemsets,
corresponding association rules
1. $L_0$ =Ø; k=1; $C_1$ ={{i}|i is the element of I}
2. TOPC = {{1 2...n}}; TOPL = Ø
3. While $C_k$=Ø
```

```
4. Read database and count supports for $C_k$ and TOPC.
5. Remove frequent itemsets from TOPC and add them to TOPL
6. $L_k$= {frequent itemsets in $C_k$} \{subsets of TOPL}
7. GARBAGE k = {infrequent itemsets in $C_k$}
8. Call the TOPC-gen algorithm if $GARBAGE_k = \emptyset$
9. Call the join procedure to generate $C_{k+1}$
10. If any frequent itemset in $C_k$ is removed in line 6
11. Call recovery procedure to recover candidates to $C_{k+1}$
12. Call new prune procedure to prune candidates in $C_{k+1}$
13. k:=k+1
14. End-while
15. Return TOPL
```

The TOPC is initialized to contain one itemset, which consists of all the database items. The TOPC is updated whenever new infrequent itemsets are found (line 8). If an itemset in the TOPC is found to be frequent, then its subsets will not participate in the subsequent support counting and candidate set generation steps. Line 6 will exclude those itemsets that are subsets of any itemset in the current TOPL, which contains the frequent itemsets found in the TOPC. If some itemsets in L_k are removed, the algorithm will call the *recovery* procedure to recover missing candidates (line 11).

5.5 Join Procedure

```
Algorithm: self-joining
        Input: Set of frequent itemsets $L_{k-1}$
        Output: Set of candidate itemsets $C_k$
        1. Select $p.item_1, p.item_2, ..., p.item_{k-1}, q.item_{k-1}$
           From $L_{k-1}$ p, $L_{k-1}$ q
               Where $p.item_1=q.item_1$,. , $p.item_{k-2}=q.item_{k-2}$, $p.item_{k-1} < q.item_{k-1}$
        2. Return Ck
```

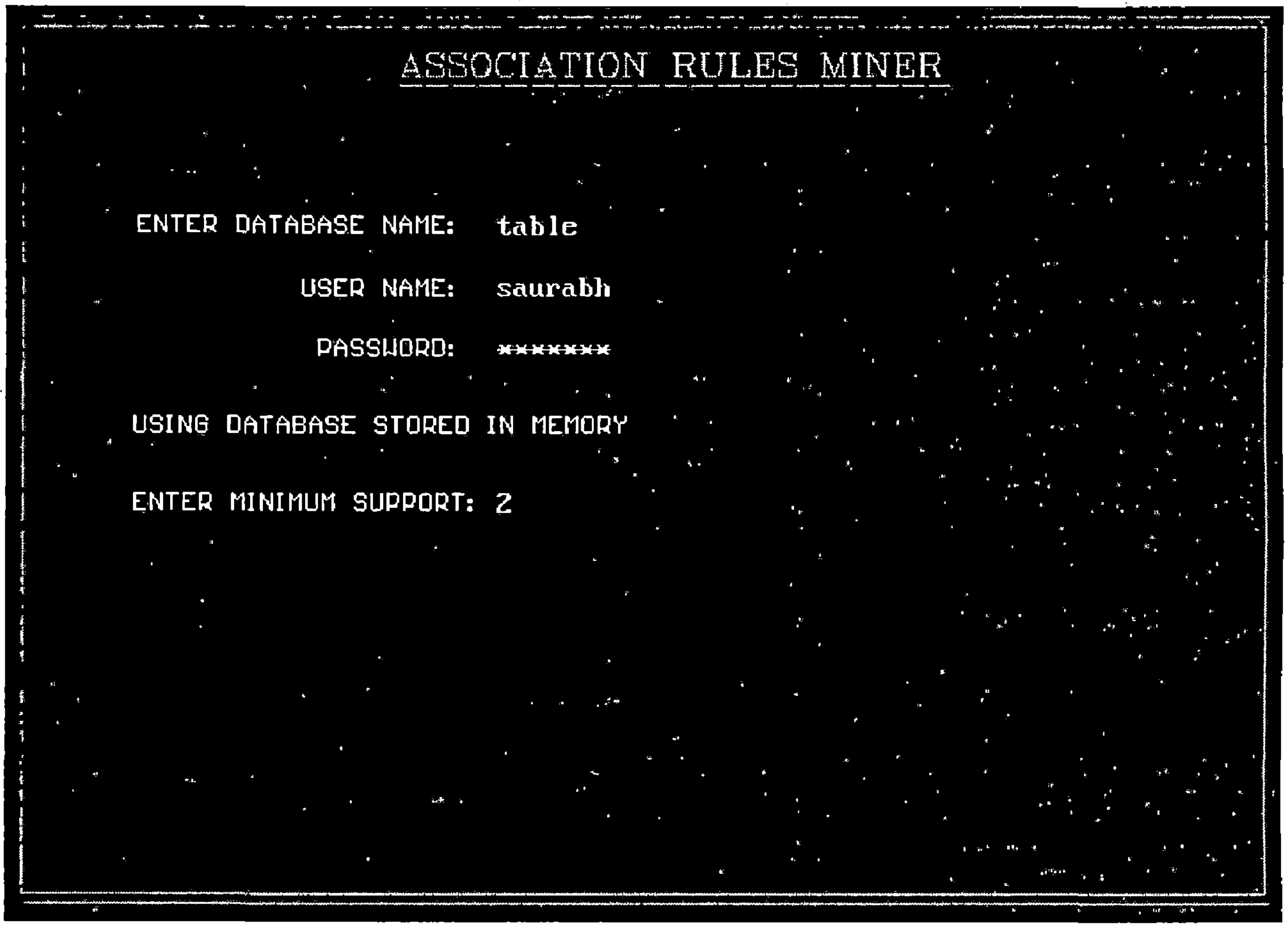
5.6 Prune Procedure

```
Algorithm: pruning
Input: $C_k$
Output: $C_k$
        1. For all itemsets c in $C_k$ do
2.      For all (k-1)-subsets s of c do
        If (s is not in $L_{k-1}$) then delete c from $C_k$
3. Return $C_k$
```

5.7 Apriori Procedure

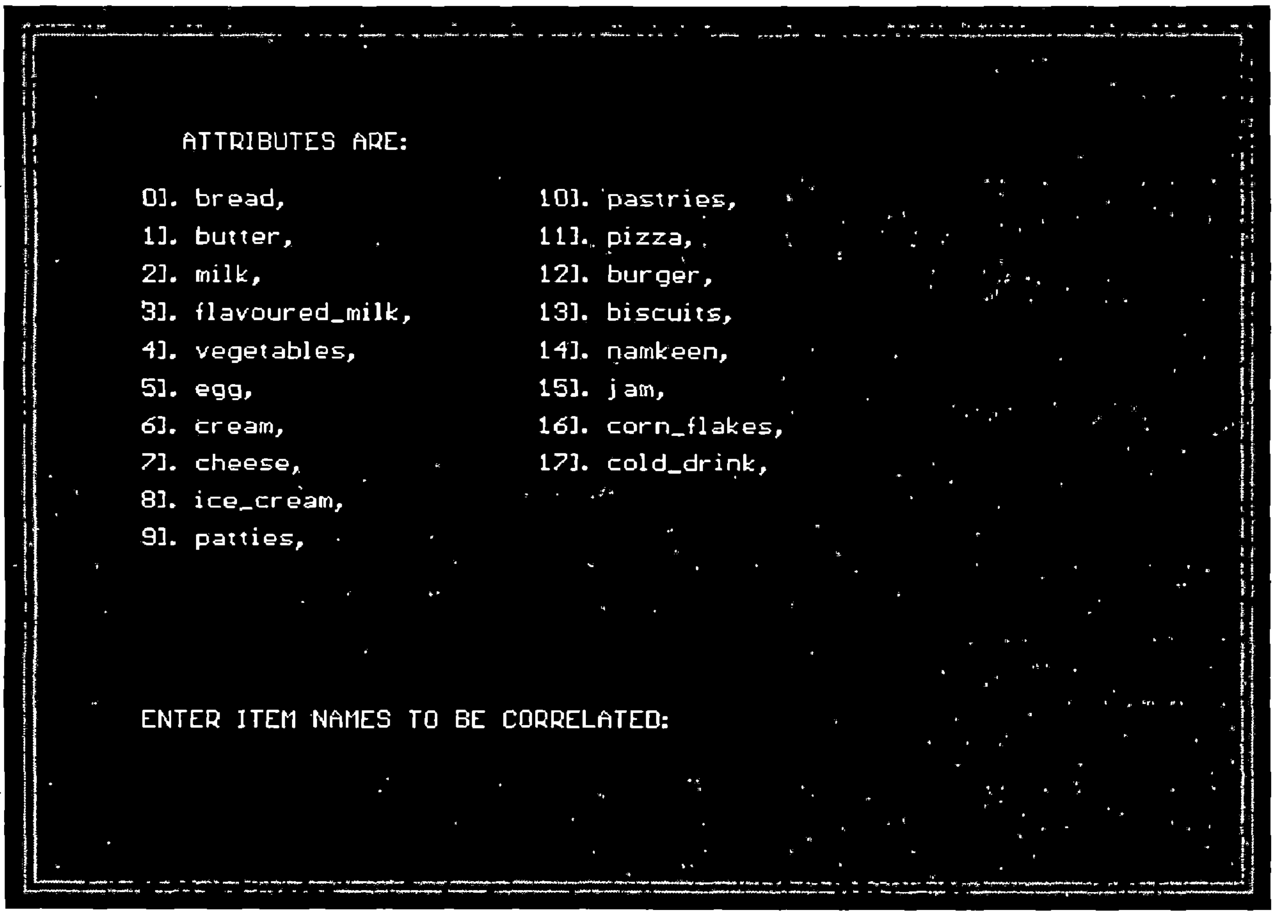
```
Algorithm: Apriori
    Input: Minimum support min_support, Attributes to be correlated
    Output: Set of frequent itemsets $L_k$
        $C_k$: Candidate itemset of size k
        $L_i$: {frequent items}
        1. For ($k = 1$; $L_k$ !=$\varnothing$; $k$++) do begin
        2. $C_{k+1}$ = candidates generated from $L_k$;
        3. For each transaction $t$ in database do
                4. Increment the count of all candidates in $C_{k+1}$
        that are contained in $t$
        5. $L_{k+1}$ = candidates in $C_{k+1}$ with min_support
        End
        Return $\cup_k L_k$;
```

Chapter 6
Results and Their Interpretation



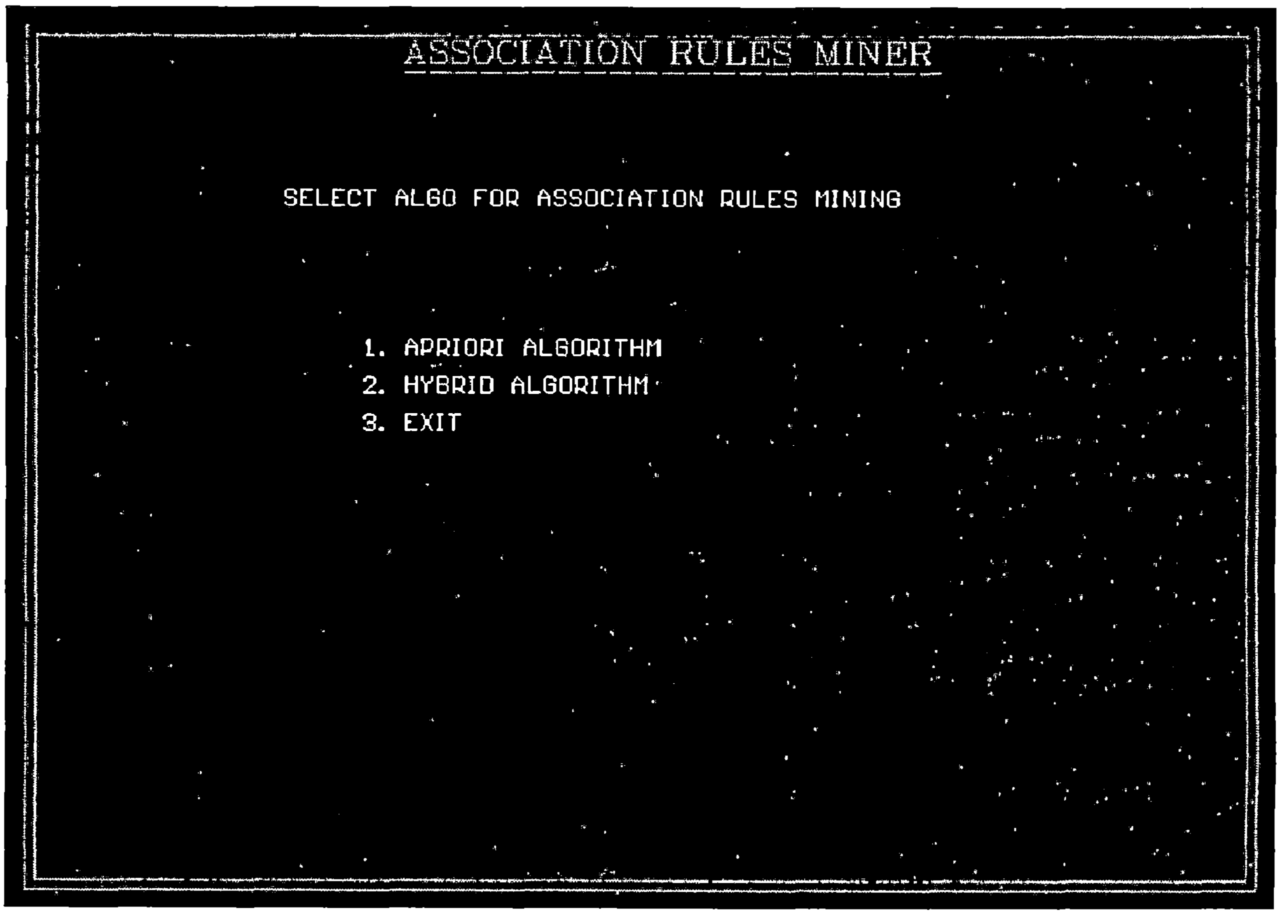
```
ASSOCIATION RULES MINER

ENTER DATABASE NAME:  table

          USER NAME:  saurabh

           PASSWORD:  *******

USING DATABASE STORED IN MEMORY

ENTER MINIMUM SUPPORT: 2
```

Fig 6.1: Screen to take user input database name and minimum support



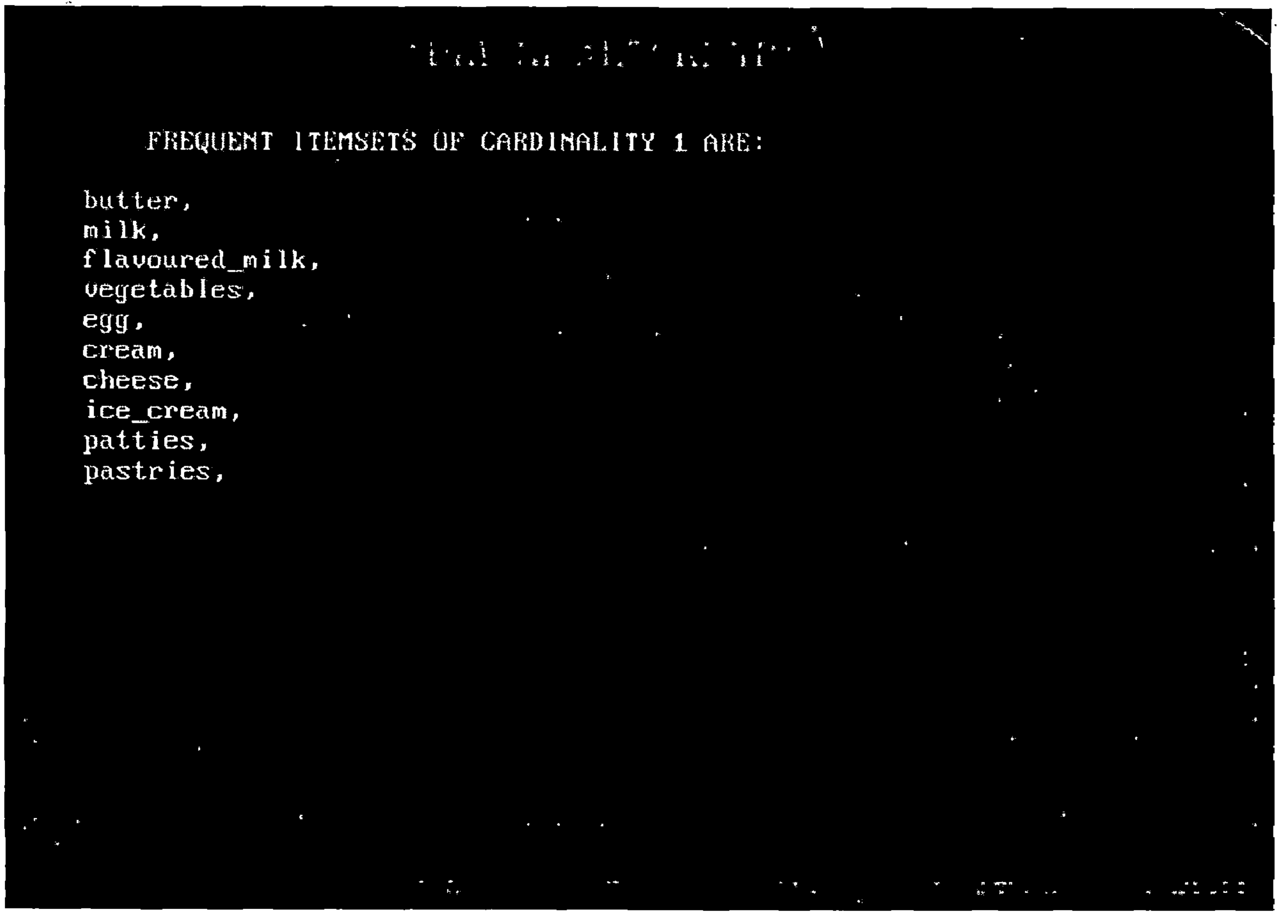
```
ATTRIBUTES ARE:

0]. bread,                  10]. pastries,
1]. butter,                 11]. pizza,
2]. milk,                   12]. burger,
3]. flavoured_milk,         13]. biscuits,
4]. vegetables,             14]. namkeen,
5]. egg,                    15]. jam,
6]. cream,                  16]. corn_flakes,
7]. cheese,                 17]. cold_drink,
8]. ice_cream,
9]. patties,


ENTER ITEM NAMES TO BE CORRELATED:
```

Fig 6.2: Screen to list attributes and take user input attributes



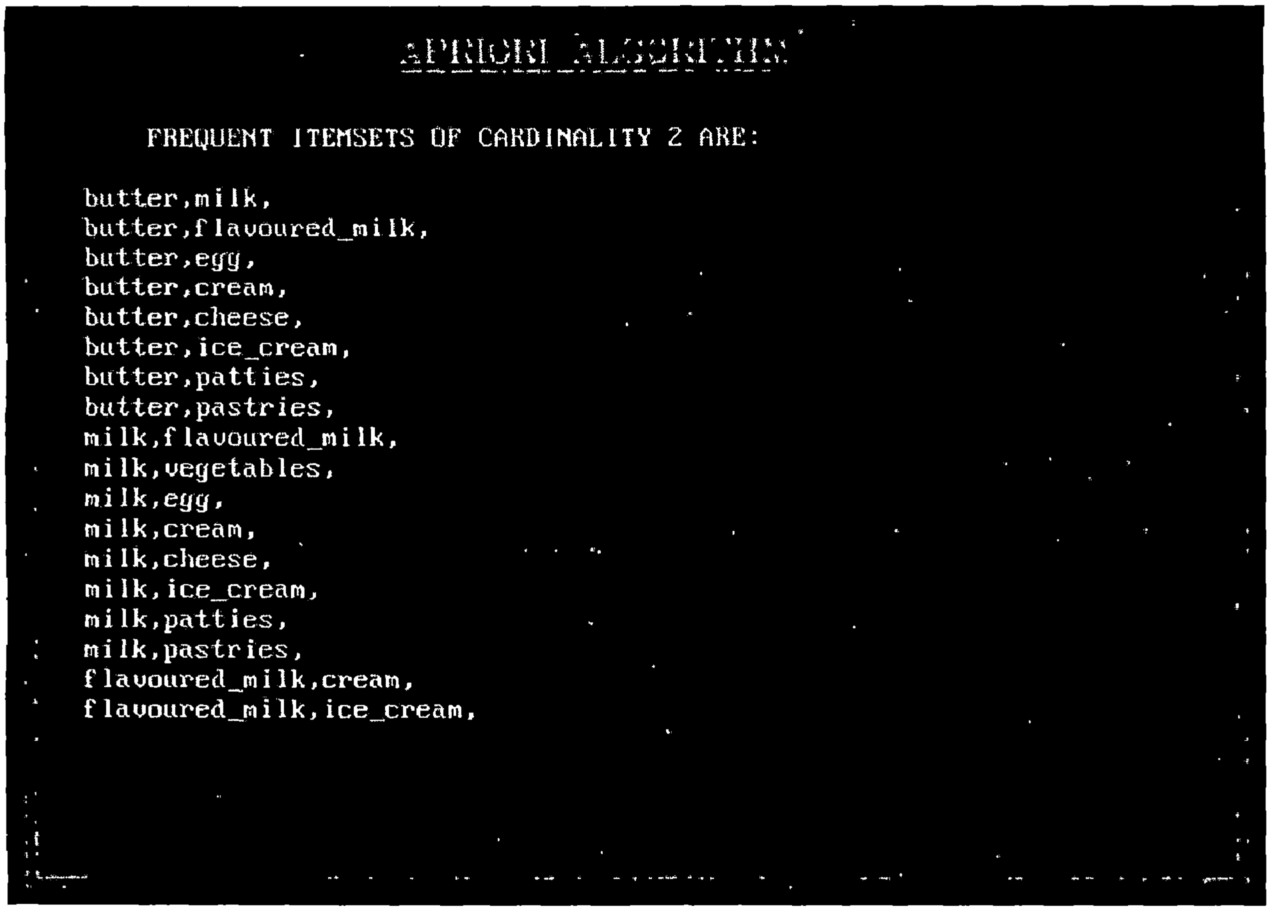
```
ASSOCIATION RULES MINER

SELECT ALGO FOR ASSOCIATION RULES MINING

        1. APRIORI ALGORITHM
        2. HYBRID ALGORITHM
        3. EXIT
```

Fig 6.3: Screen to ask for algorithm to be used



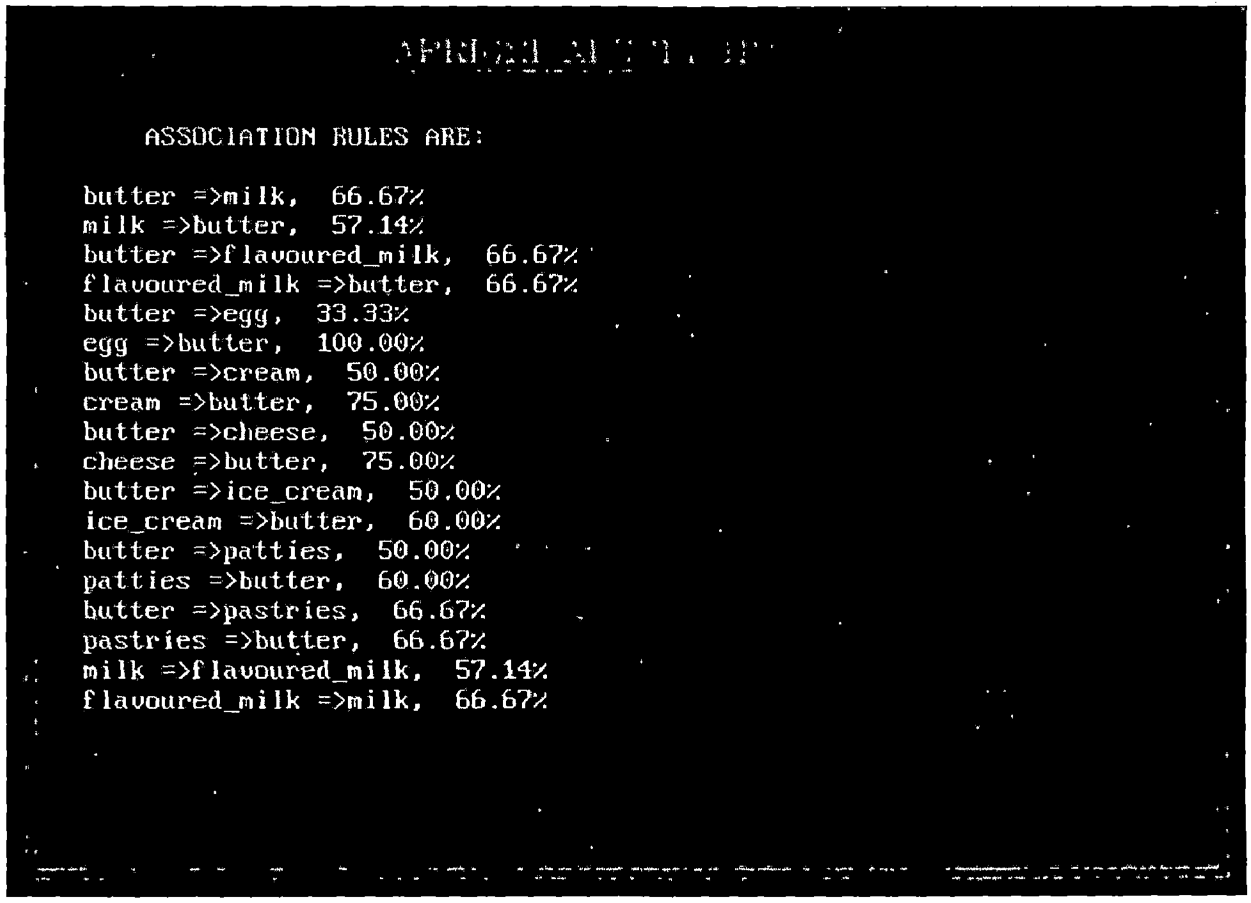
```
    FREQUENT ITEMSETS OF CARDINALITY 1 ARE:

butter,
milk,
flavoured_milk,
vegetables,
egg,
cream,
cheese,
ice_cream,
patties,
pastries,
```

Fig 6.4: Frequent item sets of cardinality 1 generated by Apriori algorithm



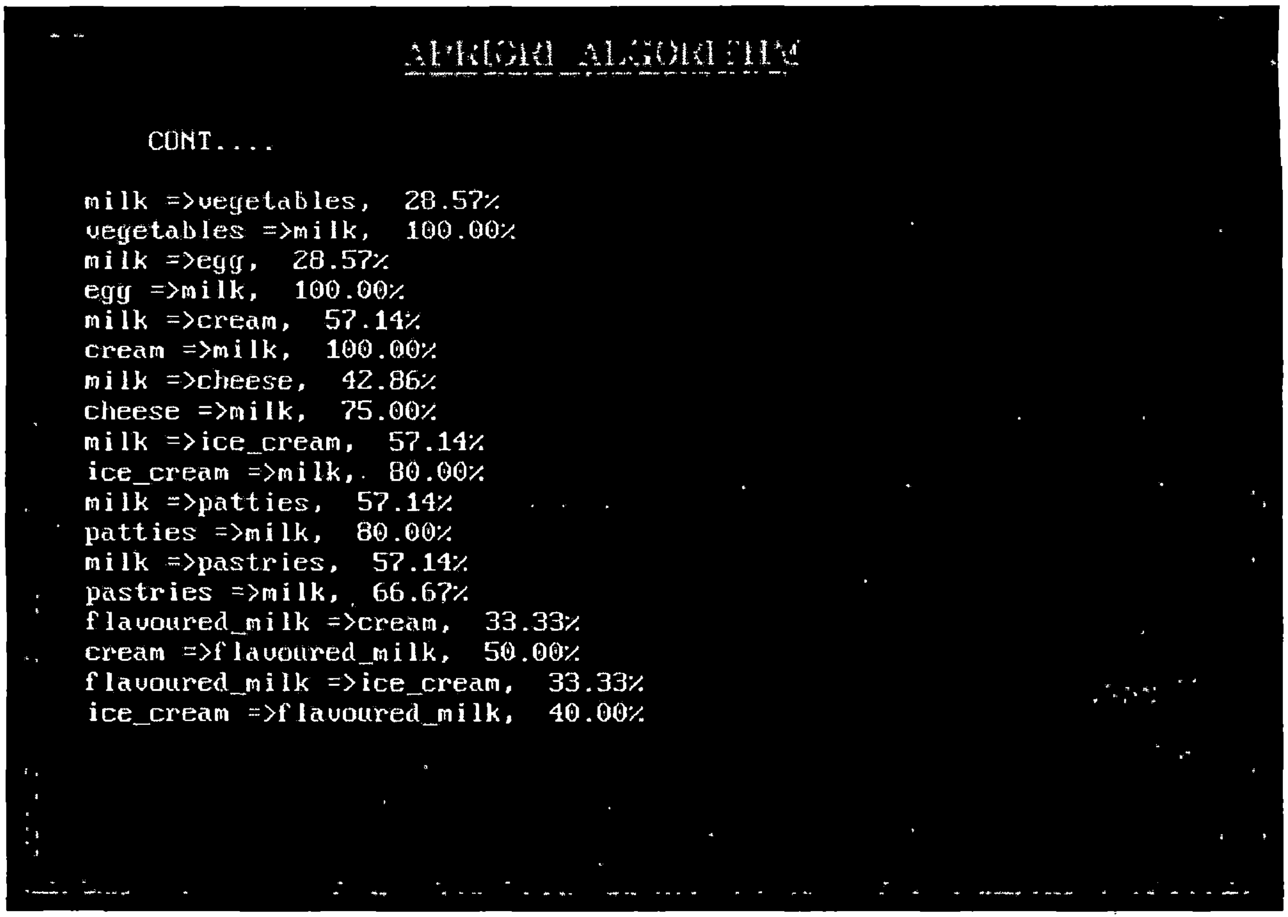
```
APRIORI ALGORITHM

    FREQUENT ITEMSETS OF CARDINALITY 2 ARE:

butter,milk,
butter,flavoured_milk,
butter,egg,
butter,cream,
butter,cheese,
butter,ice_cream,
butter,patties,
butter,pastries,
milk,flavoured_milk,
milk,vegetables,
milk,egg,
milk,cream,
milk,cheese,
milk,ice_cream,
milk,patties,
milk,pastries,
flavoured_milk,cream,
flavoured_milk,ice_cream,
```

Fig 6.5: Screen to show frequent item sets of cardinality 2 generated by Apriori algorithm



```
[illegible]

   ASSOCIATION RULES ARE:

butter =>milk,  66.67%
milk =>butter,  57.14%
butter =>flavoured_milk,  66.67%
flavoured_milk =>butter,  66.67%
butter =>egg,  33.33%
egg =>butter,  100.00%
butter =>cream,  50.00%
cream =>butter,  75.00%
butter =>cheese,  50.00%
cheese =>butter,  75.00%
butter =>ice_cream,  50.00%
ice_cream =>butter,  60.00%
butter =>patties,  50.00%
patties =>butter,  60.00%
butter =>pastries,  66.67%
pastries =>butter,  66.67%
milk =>flavoured_milk,  57.14%
flavoured_milk =>milk,  66.67%
```

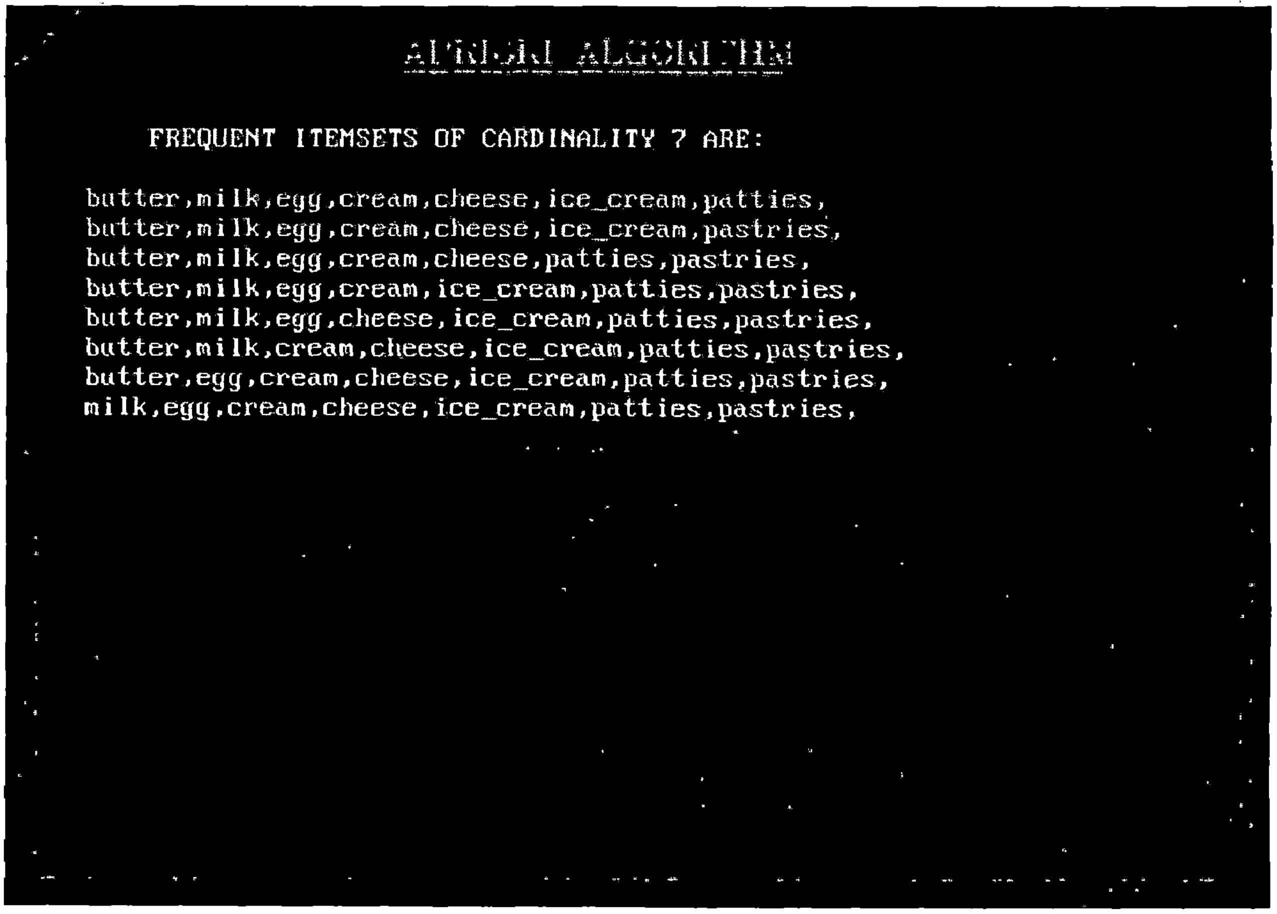
Fig 6.6: Screen to show association rules generated by Apriori algorithm



APRIORI ALGORITHM

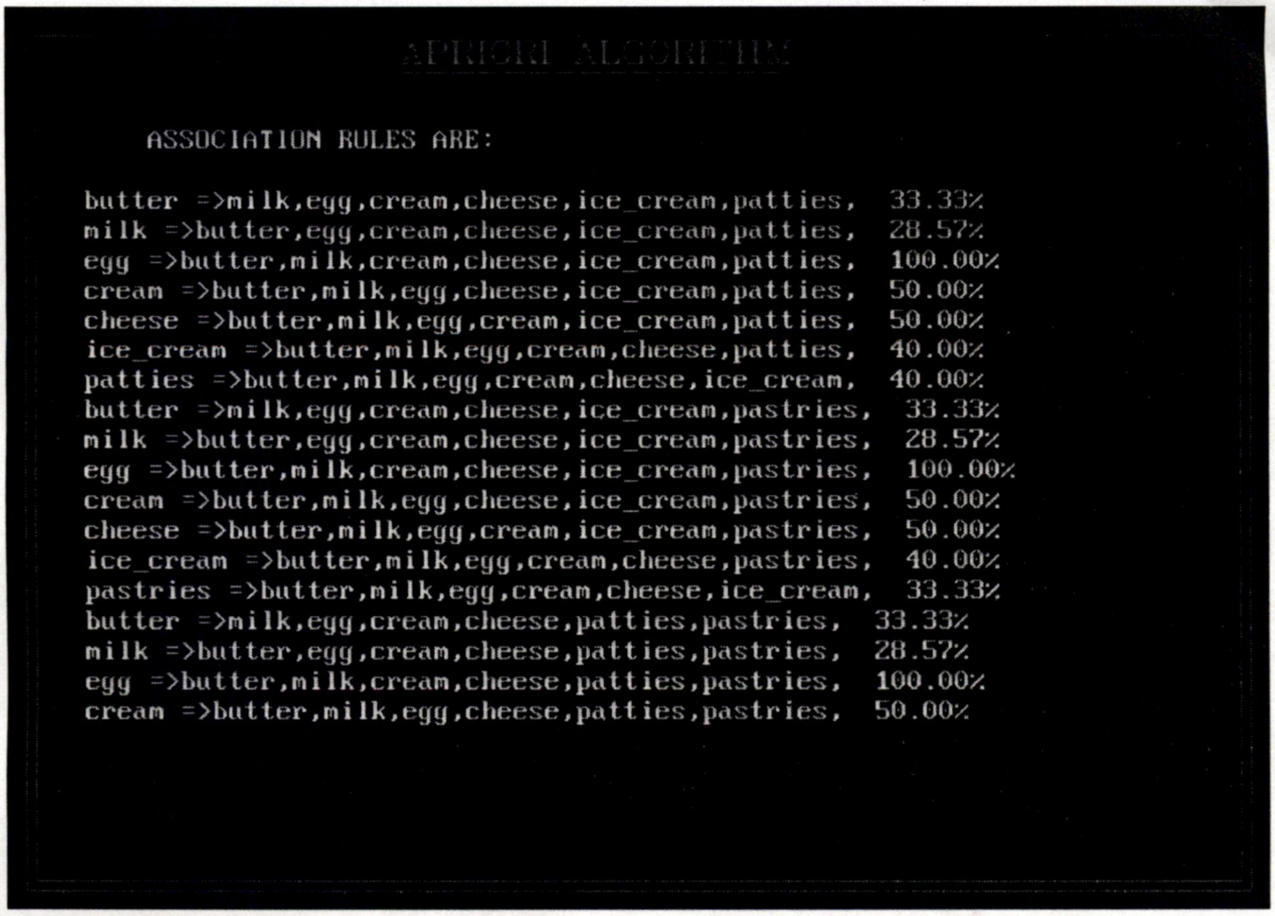
```
    CONT....

milk =>vegetables,  28.57%
vegetables =>milk,  100.00%
milk =>egg,  28.57%
egg =>milk,  100.00%
milk =>cream,  57.14%
cream =>milk,  100.00%
milk =>cheese,  42.86%
cheese =>milk,  75.00%
milk =>ice_cream,  57.14%
ice_cream =>milk,  80.00%
milk =>patties,  57.14%
patties =>milk,  80.00%
milk =>pastries,  57.14%
pastries =>milk,  66.67%
flavoured_milk =>cream,  33.33%
cream =>flavoured_milk,  50.00%
flavoured_milk =>ice_cream,  33.33%
ice_cream =>flavoured_milk,  40.00%
```

Fig 6.7: Association rules continued



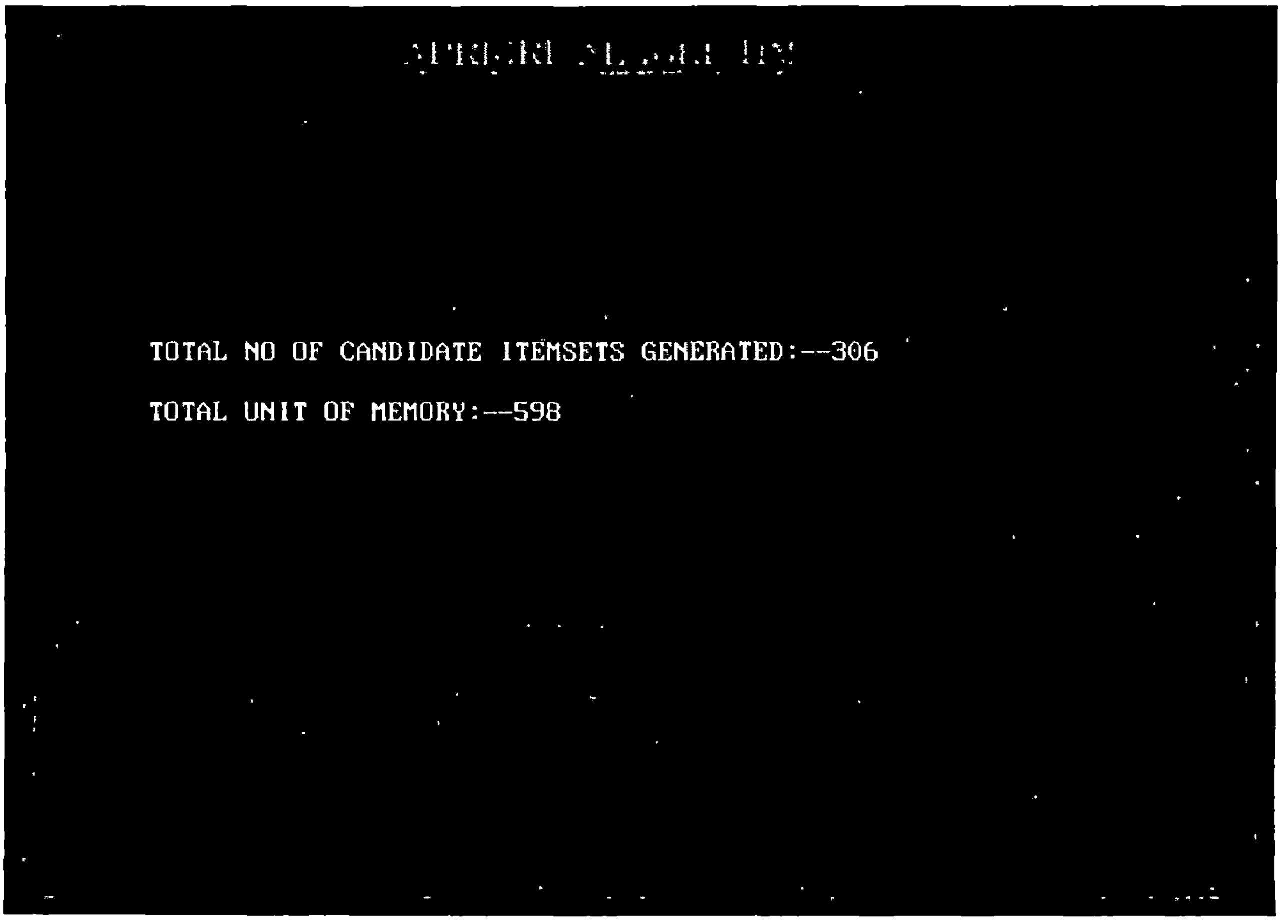
```
APRIORI ALGORITHM

    FREQUENT ITEMSETS OF CARDINALITY 7 ARE:

butter,milk,egg,cream,cheese,ice_cream,patties,
butter,milk,egg,cream,cheese,ice_cream,pastries,
butter,milk,egg,cream,cheese,patties,pastries,
butter,milk,egg,cream,ice_cream,patties,pastries,
butter,milk,egg,cheese,ice_cream,patties,pastries,
butter,milk,cream,cheese,ice_cream,patties,pastries,
butter,egg,cream,cheese,ice_cream,patties,pastries,
milk,egg,cream,cheese,ice_cream,patties,pastries,
```

Fig 6.8: frequent item sets of cardinality 7 generated by Apriori algorithm



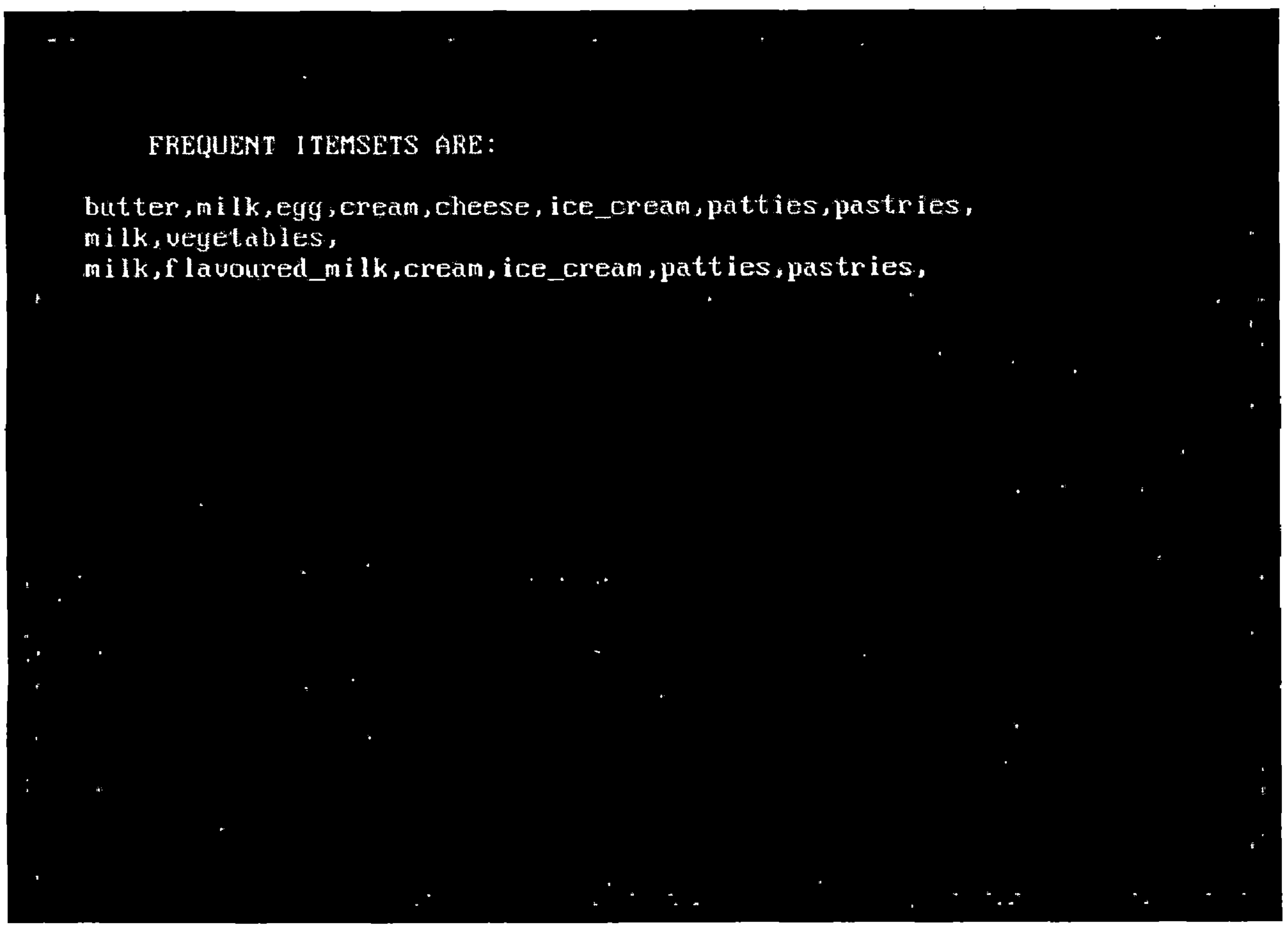
```
                          APRIORI ALGORITHM

    ASSOCIATION RULES ARE:

butter =>milk,egg,cream,cheese,ice_cream,patties,  33.33%
milk =>butter,egg,cream,cheese,ice_cream,patties,  28.57%
egg =>butter,milk,cream,cheese,ice_cream,patties,  100.00%
cream =>butter,milk,egg,cheese,ice_cream,patties,  50.00%
cheese =>butter,milk,egg,cream,ice_cream,patties,  50.00%
ice_cream =>butter,milk,egg,cream,cheese,patties,  40.00%
patties =>butter,milk,egg,cream,cheese,ice_cream,  40.00%
butter =>milk,egg,cream,cheese,ice_cream,pastries,  33.33%
milk =>butter,egg,cream,cheese,ice_cream,pastries,  28.57%
egg =>butter,milk,cream,cheese,ice_cream,pastries,  100.00%
cream =>butter,milk,egg,cheese,ice_cream,pastries,  50.00%
cheese =>butter,milk,egg,cream,ice_cream,pastries,  50.00%
ice_cream =>butter,milk,egg,cream,cheese,pastries,  40.00%
pastries =>butter,milk,egg,cream,cheese,ice_cream,  33.33%
butter =>milk,egg,cream,cheese,patties,pastries,  33.33%
milk =>butter,egg,cream,cheese,patties,pastries,  28.57%
egg =>butter,milk,cream,cheese,patties,pastries,  100.00%
cream =>butter,milk,egg,cheese,patties,pastries,  50.00%
```

Fig 6.9: Association rules generated by Apriori algorithm



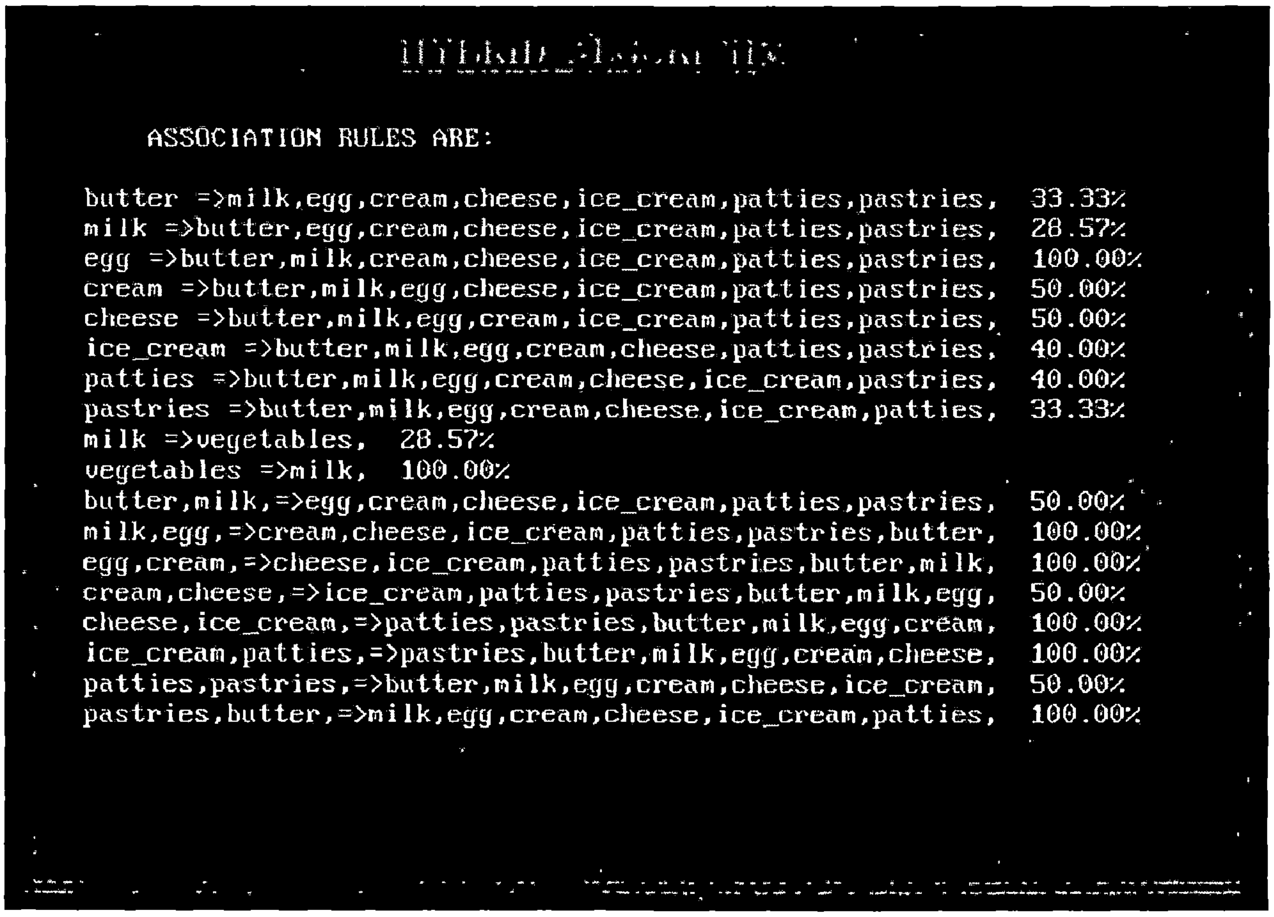
```
APRIORI ALGORITHM

TOTAL NO OF CANDIDATE ITEMSETS GENERATED:--306

TOTAL UNIT OF MEMORY:--598
```

Fig 6.10: Screen to show performance of Apriori algorithm



```
    FREQUENT ITEMSETS ARE:

butter,milk,egg,cream,cheese,ice_cream,patties,pastries,
milk,vegetables,
milk,flavoured_milk,cream,ice_cream,patties,pastries,
```

Fig 6.11: Most frequent item sets generated by Hybrid algorithm



```
                    [illegible]

     ASSOCIATION RULES ARE:

butter =>milk,egg,cream,cheese,ice_cream,patties,pastries,   33.33%
milk =>butter,egg,cream,cheese,ice_cream,patties,pastries,   28.57%
egg =>butter,milk,cream,cheese,ice_cream,patties,pastries,   100.00%
cream =>butter,milk,egg,cheese,ice_cream,patties,pastries,   50.00%
cheese =>butter,milk,egg,cream,ice_cream,patties,pastries,   50.00%
ice_cream =>butter,milk,egg,cream,cheese,patties,pastries,   40.00%
patties =>butter,milk,egg,cream,cheese,ice_cream,pastries,   40.00%
pastries =>butter,milk,egg,cream,cheese,ice_cream,patties,   33.33%
milk =>vegetables,  28.57%
vegetables =>milk,  100.00%
butter,milk,=>egg,cream,cheese,ice_cream,patties,pastries,   50.00%
milk,egg,=>cream,cheese,ice_cream,patties,pastries,butter,   100.00%
egg,cream,=>cheese,ice_cream,patties,pastries,butter,milk,   100.00%
cream,cheese,=>ice_cream,patties,pastries,butter,milk,egg,   50.00%
cheese,ice_cream,=>patties,pastries,butter,milk,egg,cream,   100.00%
ice_cream,patties,=>pastries,butter,milk,egg,cream,cheese,   100.00%
patties,pastries,=>butter,milk,egg,cream,cheese,ice_cream,   50.00%
pastries,butter,=>milk,egg,cream,cheese,ice_cream,patties,   100.00%
```

Fig 6.12: Association rules generated by Hybrid algorithm

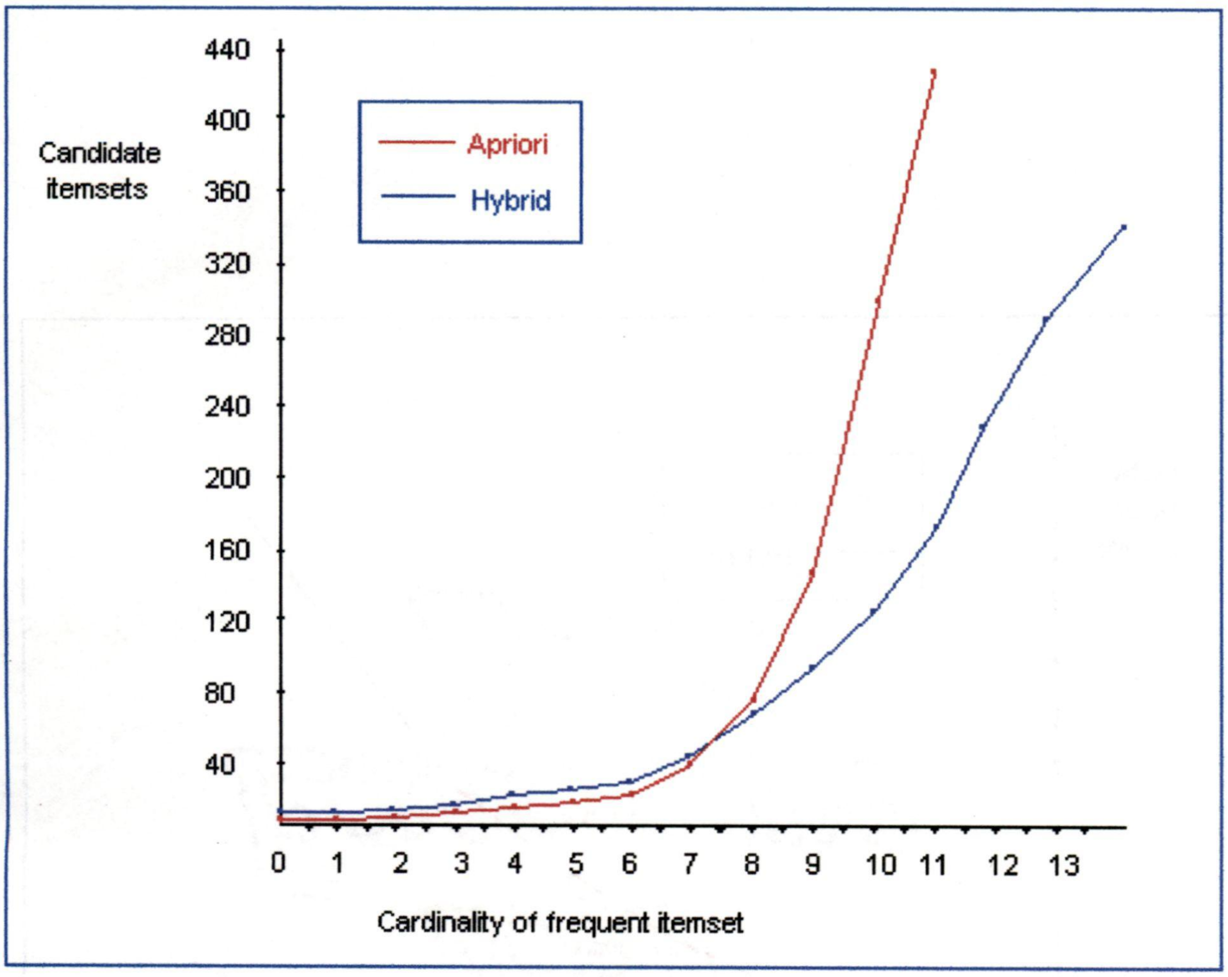


Fig 6.13: Graph shows dependence between no. of large frequent item sets and candidate item sets generated

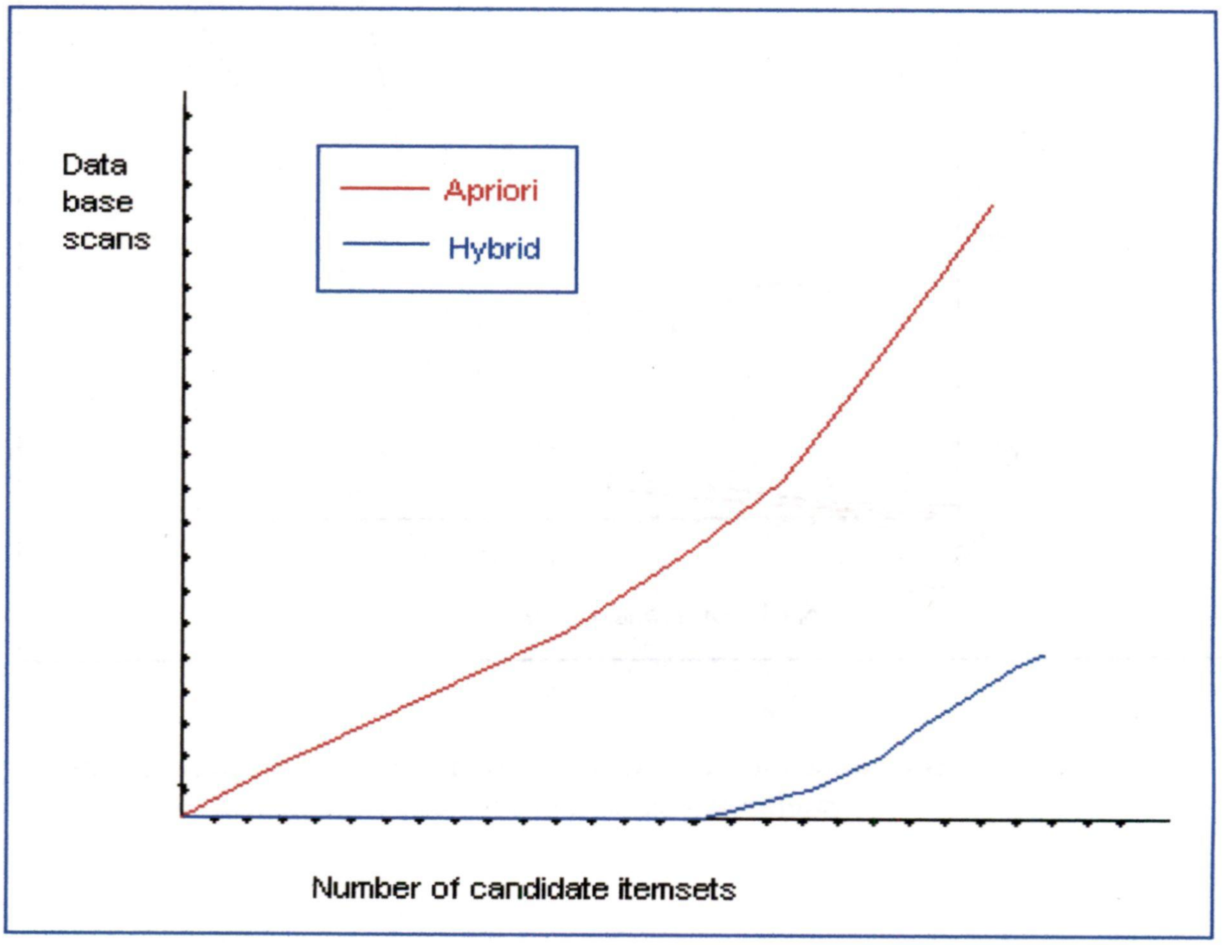


Fig 6.14: Graph shows relation between no. of data base scans and no. of candidate itemsets

Chapter 7

Concluding Remarks

An efficient way to discover the maximum frequent set can be very useful in various data mining problems, such as the discovery of the association rules. The maximum frequent set provides a unique representation of all the frequent itemsets. In many situations, it suffices to discover the maximum frequent set, and once it is known, all the required frequent subsets can be easily generated.

This work presents an algorithm that can efficiently discover the maximum frequent set. Hybrid-Search algorithm could reduce both the number of times the database is read and the number of candidates considered.

Experiments show that the improvement of using this approach can be very significant, especially when some maximal frequent itemsets are long. A popular assumption is that the maximal frequent itemsets are usually very short and therefore the computation of *all* (and not just maximal) frequent itemsets is feasible. Such assumption on maximal frequent itemsets does not need to be true in important applications. Hybrid algorithm may be useful in these applications such as the problem of discovering patters in price changes of individual stocks in a stock market. Prices of individual stocks are frequently quite correlated with each other. Therefore, the discovered patterns may contain many items (stocks) and the frequent itemsets are long.

The number of data base scans are reduced significantly, it depends upon the size of memory.

References

1. Agrawal R., Imielinski T., and Swami A. "Mining association rules between sets of items in very large databases". *Proceedings of the ACM SIGMOD Conference on Management of data,* pages 207-216, 1993.

2. Jiawei Han and Michaline Camber. "Data Mining Concepts and Techniques", Simon Fraser University, 2000.

3. Park J. S., Chen M. S., and Yu P. S. "An Effective Hash-based Algorithm for Mining Association Rules." *Proceedings of the ACM-SIGMOD Conference on Management of Data, 1995.* Extended version appears as: "Using a Hash-based Method with Transaction Trimming for Mining Association Rules." IEEE Trans-actions ob Knowledge and Data Engineering, Volume 9, no 5, September 1997, pages 813-825.

4. Charu C. Aggarwal and Philip S. Yu. "Mining Large Itemsets for Association Rules". IBM T. J. Watson Research Center.

5. Rakesh Agrawal and Ramakrishnan Srikant."Fast algorithm for mining association rules". *IBM Research report RJ9839, June 1994*, IBM Almaden Research Center, san Jose, CA.

6. Park J. S., Chen M. S., and Yu P. S. "An Effective Hash-based Algorithm for Mining Association Rules". *Proceedings of the ACM-SIGMOD Conference on Management of Data, 1995.* Extended version appears as: "Using a Hash-based Method with Transaction Trimming for Mining Association Rules." *IEEE Trans-actions ob Knowledge and Data Engineering*, Volume 9, no 5, September 1997, pages 813-825.

7. Savasere A., Omiecinski E., and Navathe S. B. "An efficient algorithm for mining association rules in large databases." *Proceedings of the 21st International Conference on Very Large Databases,* 1995.

8. Bayardo R. J. "Efficiently Mining Long Patterns from Databases." *Unpublished Research Report*.

9. M. Houtsma and A. Swami. Set-oriented mining of association rules. *Research Report RJ 9567*, IBM Almaden Research Center, Oct. 1993.

10. Agrawal R., and Srikant R. "Fast Algorithms for Mining Association Rules in Large Databases." *Proceedings of the 20th International Conference on Very Large Data base,* pages 478-499, 1994.

11. H. Mannila, H. Toivonen, and A. Verkamo. Improved methods for finding association rules. In *Proc. AAAI Workshop on Knowledge Discovery*, July 1994.

12. J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *Proc. ACM-SIGMO*D, May 1995.

13. A. Sarasere, E. Omiecinsky, and S. Navathe.Anefficient algorithm for mining association rules in large databases. In *Proc. 21st VLD*B, Sept. 1995.

14. H. Toivonen. Sampling large databases for association rules. In *Proc. 22nd VLD*B, Sept. 1996.

15. D. Guñopulos, H. Mannila, and S. Saluja. Discovering all most specific sentences by randomized algorithm. In *Proc. 13th ICD*T, Jan. 1997.

16. S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. SIGMO*D, May 1997.

17. M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proc. 3rd KD*D, Aug. 1997.

Appendix A

Pro C Commands Used In Project

To Declare The Variables

```
        EXEC SQL begin declare section;
        variable types and names;
        EXEC SQL end declare section;
```

To Handle Errors

```
    EXEC SQL WHENEVER SQLERROR DO sql_error("error message");
```

To Connect From Oracle

```
    EXEC SQL 1 begin declare section;
        char *username;
        EXEC SQL  end declare section;

    EXEC SQL CONNECT :username ;
```

To Display Attribute's Name

```
    EXEC SQL DECLARE contents CURSOR FOR
    select column_name from user_tab_columns where table_name="table";
    exec sql begin declare section;
    varchar name[17];
    exec sql end declare section;
    exec sql open contents;
    exec sql whenever not found do break;
```

TID	PIZZA	BURGER	PASTRIES	PATTIES	CORN_FLAKES	JAM
1	1	1	0	1	1	0
2	1	1	0	0	1	0
3	1	1	0	1	1	0
4	1	0	0	0	1	0
5	1	0	0	0	0	0
6	0	0	1	0	0	1
7	0	0	1	1	0	1
8	0	1	0	1	0	1
9	0	1	0	1	0	0
10	0	1	0	1	0	0
….	……	…….	……..	…….	……..	…….

Appendix A

Pro C Commands Used In Project

To Declare The Variables

```
        EXEC SQL begin declare section;
        variable types and names;
        EXEC SQL end declare section;
```

To Handle Errors

```
    EXEC SQL WHENEVER SQLERROR DO sql_error("error message");
```

To Connect From Oracle

```
    EXEC SQL l begin declare section;
        char *username;
        EXEC SQL  end declare section;

    EXEC SQL CONNECT :username ;
```

To Display Attribute's Name

```
    EXEC SQL DECLARE contents CURSOR FOR
    select column_name from user_tab_columns where table_name="table";
    exec sql begin declare section;
    varchar name[17];
    exec sql end declare section;
    exec sql open contents;
    exec sql whenever not found do break;
```

```
for(;;)
{
        exec sql fetch contents into :name;
        dbms_output.put_line(||name);
}
exec sql close contents;
exec sql commit work release;
```

To Find The Support Of Attributes

```
EXEC SQL DECLARE count CURSOR FOR
select tid
from "table"
where
for(int y=0;y<tempc->length;y++)
:tempc->candidate[y]=1;
exec sql open count;
exec sql whenever not found do break;
for(;;)
{
exec sql fetch count into :tid;
:tempc->support++;
}
exec sql close count;
exec sql commit work release;
```

Sample Database

TID	BREAD	BUTTER	BISCUITS	ICE_CREAM	MILK	EGG
1	0	0	0	1	1	0
2	1	1	0	1	1	0
3	1	1	0	1	1	0
4	1	1	0	1	0	0
5	0	0	0	1	0	1
6	1	0	0	1	0	1
7	0	0	1	1	1	1
8	1	1	1	0	1	1
9	0	1	1	0	1	1
10	1	1	1	0	0	1
…..	…..	….	……	……	…..	…..

TID	VEGITABLES	CREAM	CHEESE	FLAV_ MILK	NAMKEEN	COLD_DRI NKS
1	1	1	1	1	0	1
2	1	1	1	1	0	1
3	1	1	1	1	0	1
4	1	0	1	1	0	0
5	1	0	1	1	0	0
6	1	0	0	1	1	0
7	1	0	0	1	0	0
8	0	0	0	0	0	1
9	0	0	1	0	0	1
10	0	0	1	0	0	1
…	……	…….	…….	…….	…….	……

TID	PIZZA	BURGER	PASTRIES	PATTIES	CORN_FLAKES	JAM
1	1	1	0	1	1	0
2	1	1	0	0	1	0
3	1	1	0	1	1	0
4	1	0	0	0	1	0
5	1	0	0	0	0	0
6	0	0	1	0	0	1
7	0	0	1	1	0	1
8	0	1	0	1	0	1
9	0	1	0	1	0	0
10	0	1	0	1	0	0
….	……	…….	……..	…….	……..	…….