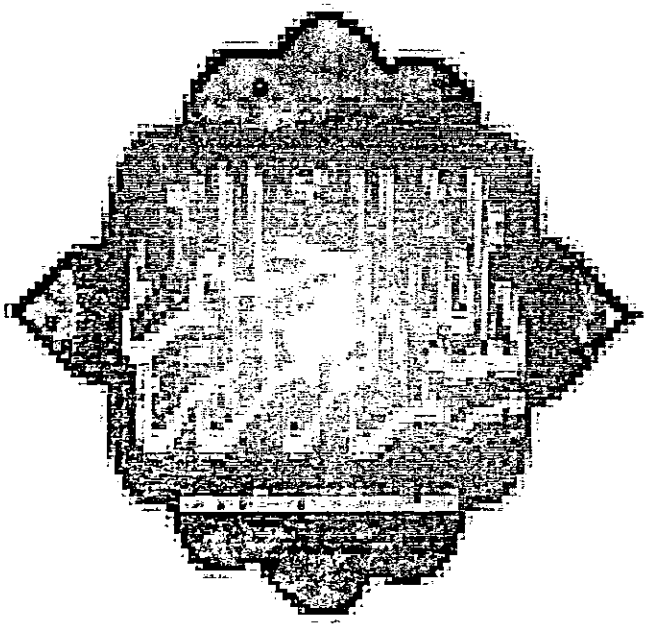
Frequent Itemset generation Using Cosine Measure

Undertaken by
Sobia Malik [330-FAS/MSCS/F06]

Supervisors
Mr. Muhammad Imran Saeed

Department of Computer Science
Faculty of Basic and Applied Sciences
International Islamic University, H-10, Islamabad
2008



In the Name of ALLAH ALMIGHTY, The most Merciful, The most Benefice

Final Approval

Dated: 20/03/2009

It is certified that we have read the project report submitted by Ms. Sobia Malik, Reg# 330-MSCS/FAS/F06. it is our judgment that this project is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad for Masters Degree in Computer Science.

Committee

External Examiner

Dr.Abdus Sattar
Director General,
Computer Bureau,
Islamabad, Pakistan.

Internal Examiner

Miss Zakia Jalil
Research Associate,
Department of Computer Science,
International Islamic University, Islamabad.

Supervisor

Mr. Muhammad Imran Saeed
Assistant Professor,
Department of Computer Science,
International Islamic University, Islamabad.

A dissertation submitted to the
Department of Computer Science,
Faculty of Applied Sciences,
International Islamic University, Islamabad, Pakistan,
As a fulfillment of the requirements for the award of the degree of

MS in Computer Science

To
Our Beloved Prophet MUHAMMAD(SAW)
Best role model for our lives

To
My Parents
Whose encouragement and prayers have always motivated me towards a successful life

To
My Teachers
Who not only paved way to my goals, but also inspired me their knowledge.

Declaration

I hereby declare that this software has not been copied as a whole from any source. Furthermore, the developed software and the accompanied thesis report is made entirely on the basis of personal efforts under my honest motivation, hard work and sincere guidance.

In addition, no portion of the work presented in this report has been submitted in support of an application for other degree or qualification of this or other university.

Sobia Malik
330-FAS/MSCS/F06

Acknowledgement

Many gratitudes to **Almighty ALLAH**, The most Merciful and Glorious, who gave me courage and potential to work hard and out up all my efforts throughtout my research work, who enabled me o perceive higher ideas in addition to bestowing me with his blessings and patience. I am deeply thankful to **Holy Prophet Muhammad (SAW)** who molded our hearts to recognize our Lord and created enthusiasm in us by his true teachings of Islam, taking us away from the shadows of disappointment.

I extend my greatest appreciation to my hardworking supervisor **Mr. Muhammad Imran Saeed**, who motivated me to work hard by his suggestions and appreciation. He put confidence in me and gave me courage to carry on with my project helping me a great deal with my project proposal. Special thanks to **Mr. Saif-ur Rehman**, who placed high hopes in me, sharing his knowledge with me and making me work to the best of my ability with a sincere approach and honesty of purpose.

I cannot deny the great generosity and continued support of **Miss Zakia Jalil**, who did her best to assist and guide me throughout the procedure of research as well as kept me informed of all circumstances regarding my project. Her efforts are really appreciated.

Last, but not the least, my deepest gratitude to my beloved parents, who prayed for my success day and night and gave me all the support and confidence I needed since my childhood. They were the one who paved way to my destination and lead me to the light of my dreams throughout all the shadows of hopelessness and disappointment rewarding me with a pessimistic approach towards life. I cannot forget the support and love I got from my brothers, sisters, in-laws and my husband throughout my project. Their support and courage will always be greatly appreciated.

Sobia Malik
330-FAS/MSCS/F06

Project in Brief

Project Title: "Frequent itemset generation using Cosine Measure".

Organization: International Islamic University, Islamabad.

Developed by: Sobia Malik
Reg. No: 330-FAS/MSCS/F06

Supervised by: Mr. Muhammad Imran Saeed.

Starting Date: January, 2008

End Date: October, 2008

Tools used: C++, Turbo C.

System used: Pentium 4.

Abstract

Traditionally, business analysts have performed the task of extracting useful information from recorded data, but the increasing volume of data in modern business and science calls for computer-based approaches. As datasets have grown in size and complexity, there has been a shift away from direct hands-on data analysis toward indirect, automatic data analysis using more complex and sophisticated tools. The modern technologies of computers, networks, and sensors have made data collection and organization much easier. However, the captured data needs to be converted into information and knowledge to become useful. Data Mining is the process of running data through sophisticated algorithms to uncover meaningful patterns and correlations that may otherwise be hidden.

Association rule mining finds interesting associations and/or correlation relationships among large set of data items. Association rules show attributes value conditions that occur frequently together in a given dataset. A typical and widely-used example of association rule mining is Market Basket Analysis. Association rule algorithm finds associations between the frequently sold items, so that the shopkeeper could put such items together for increased sales.

Our devised algorithm, **Cos_FIS generator** has certain advantages over previous algorithms such as it scans the database only once since it uses the vertical data layout of the database while scanning the database. Hence, not only the multiple scans of the database, but also the candidate generation was also avoided in our algorithm. Also, this algorithm makes use of clustering measure known as **"Cosine measure"** rather than using the Support or confidence measures. Moreover, the algorithm was tested on the synthetic database.

Chapter No **Contents** **Page No**

List of Tables

List of Figures

Chapter 1

****

INTRODUCTION

1. Introduction

Once the user got analysis, reporting, and dashboards deployed, it's time to take business intelligence (BI) to the next level by adding data mining and advanced analytics. This is a level of BI excellence that many organizations never manage to evolve to, however the importance of pushing ahead with advanced capabilities cannot be underestimated - they can provide a truly sustainable competitive advantage and enable user's organization to maximize both its efficiency and effectiveness.

Data Mining is the process of running data through sophisticated algorithms to uncover meaningful patterns and correlations that may otherwise be hidden. These can be used to help user understand the business better and also exploit to improve future performance through predictive analytics. For example, data mining can warn user there's a high probability a specific customer won't pay on time based on an analysis of customers with similar characteristics.

Data mining identifies trends within data that go beyond simple analysis. Through the use of sophisticated algorithms, non-statistician users have the opportunity to identify key attributes of business processes and target opportunities. However, abdicating control of this process from the statistician to the machine may result in false-positives or no useful results at all.

Although data mining is a relatively new term, the technology is not. For many years, businesses have used powerful computers to run through volumes of data such as supermarket scanner data to produce market research reports (although reporting is not always considered to be data mining). Continuous innovations in computer processing power, disk storage, and statistical software are dramatically increasing the accuracy and usefulness of data analysis.

The term data mining is often used to apply to the two separate processes of knowledge discovery and prediction. Knowledge discovery provides explicit information that has a readable form and can be understood by a user (e.g., association rule mining). Forecasting, or predictive modeling provides predictions of future events and may be

Questions such as "if a customer purchases product A, how likely is he to purchase product B?" and "What products will a customer buy if he buys products C and D?" are answered by association-finding algorithms.

Each item has a Boolean variable representing the presence or absence of that item. Each basket can then be represented by a Boolean vector of values assigned to these variables. The buying patterns can be represented in the form of association rules. For example, the information that customers who buy burgers also tend to buy coke at the same time is represented in association Rule below:

{Burger}-->{Coke}

An association rule has two numbers that express the degree of uncertainty about the rule namely Support and Confidence. The Support is simply the number of transactions that include all items in the antecedent and consequent parts of the rule. **Confidence** is the ratio of the number of transactions that include all items in the consequent as well as the antecedent (namely, the support) to the number of transactions that include all items in the antecedent. For example, if a supermarket database has 100,000 point-of-sale transactions, out of which 2,000 include both items A and B and 800 of these include item C, the association rule "If A and B are purchased then C is purchased on the same trip" has a support of 800 transactions (alternatively 0.8% = 800/100,000) and a confidence of 40% (=800/2,000). Association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold. Such threshold are user or expert specified. If an item set satisfies minimum support, then it is frequent item set (FIS).

One of the reasons behind maintaining any database is to enable the user to find interesting patterns and trends in the data. For example, in a supermarket, the user can figure out which items are being sold most frequently. But this is not the only type of 'trend' which one can possibly think of. The goal of database mining is to automate this process of finding interesting patterns and trends. Once this information is available, the user can perhaps get rid of the original database. The output of the data-mining process should be a "summary" of the database. This goal is difficult to achieve due to the vagueness associated with the term 'interesting'. The solution is to define various types of

trends and to look for only those trends in the database. One such type constitutes the association rule.

Association rule mining comprises of two steps i.e. finding ***frequent itemsets*** (FIS) and ***generating association rules***, based on the frequent itemsets. However, researchers have found numerous techniques to find FIS, mostly based on ***support*** measure. But, in this thesis, one of the similarity measures, known as ***"Cosine"*** measure has been used and only the first step of the association rule mining has been covered. While clustering data points are arranged in a way that the points nearest to each other are placed in one cluster. This can be done either by similarity or dissimilarity measures. Similar data items will be nearest to each other and dissimilar will be at distance far apart.

The problem of association rule mining is defined as: Let be a set of n binary attributes called *items*. Let be a set of transactions called the *database*. Each transaction in D has a unique transaction ID and contains a subset of the items in I. A *rule* is defined as an implication of the form $X \rightarrow Y$ where X is called *antecedent* (left-hand-side or LHS) and Y *consequent* (right-hand-side or RHS) of the rule.

transaction ID	**milk**	**bread**	**butter**	**beer**
1	1	1	0	0
2	0	1	1	0
3	0	0	0	1
4	1	1	1	0
5	0	1	0	0

Table 1-1 Example database with 4 items and 5 transactions

To illustrate the concepts, we use a small example from the supermarket domain. The set of items is I = {milk,bread,butter,beer} and a small database containing the items (1 codes presence and 0 absence of an item in a transaction) is shown in the table to the right. An example rule for the supermarket could be meaning that if milk and bread is bought, customers also buy butter.

Note: this example is extremely small. In practical applications, a rule needs a support of several hundred itemsets before it can be considered statistically significant, and datasets often contain thousands or millions of itemsets.

To select interesting rules from the set of all possible rules, constraints on various measures of significance and interest can be used. The best-known constraints are minimum thresholds on support and confidence. The *support* supp(X) of an itemset X is defined as the proportion of transactions in the data set which contain the itemset. In the example database, the itemset {milk,bread} has a support of 2 / 5 = 0.4 since it occurs in 40% of all transactions (2 out of 5 transactions).

The *confidence* of a rule is defined . For example, the rule has a confidence of 0.2 / 0.4 = 0.5 in the database, which means that for 50% of the transactions containing milk and bread the rule is correct. Confidence can be interpreted as an estimate of the probability $P(Y \mid X)$, the probability of finding the RHS of the rule in transactions under the condition that these transactions also contain the LHS.

In association rule mining, clustering is one of the most popular areas. Clustering is a process of partitioning a set of data (or objects) in a set of meaningful sub-classes, called clusters. Clustering is a discipline devoted to revealing and describing homogeneous groups of entities, that is, clusters, in data sets.

1.2 Clustering

Clustering is the method by which like records are grouped together. Usually this is done to give the end user a high level view of what is going on in the database. Clustering is sometimes used to mean segmentation - which most marketing people will tell is useful for coming up with a birds eye view of the business. Clustering is a data mining (machine learning) technique used to place data elements into related groups without advance knowledge of the group definitions.

1.2.1 A simple example of clustering

A simple example of clustering would be the clustering that most people perform when they do the laundry - grouping the permanent press, dry cleaning, whites and brightly colored clothes is important because they have similar characteristics. And it turns out they have important attributes in common about the way they behave (and can be ruined) in the wash. To "cluster" laundry most of the decisions are relatively straightforward. There are of course difficult decisions to be made about which cluster the white shirt with red stripes goes into (since it is mostly white but has some color and is permanent press). When clustering is used in business the clusters are often much more dynamic - even changing weekly to monthly and many more of the decisions concerning which cluster a record falls into can be difficult.

1.2.2 The Cosine Similarity Measure

Cosine similarity measure is one of the clustering measures. The purpose of clustering measure is to join together objects into successively larger clusters, using some measure of similarity or distance. A typical result of this type of clustering is the hierarchical tree. Cosine similarity is a measure of similarity between two vectors of n dimensions by finding the cosine of the angle between them, often used to compare documents in text mining. Given two vectors of attributes **A** and **B**, the cosine similarity, β is represented using a dot product and magnitude as:

$$\textbf{Similarity} = \textbf{Cos}(0) = \mathbf{A}.\mathbf{B} \,/\, \|\mathbf{A}\|.\|\mathbf{B}\|$$

For text matching, the attribute vectors **A** and **B** are usually the **tf** vectors of the documents. Furthermore, the cosine similarity of two vectors is an arbitrary mathematicl measure of how similar two vectors are on a scale of [0,1]. 1 being that the vectors are either identical, or that their values differ by a constant factor. The cosine similarity of two vectors (d1 and d2) is defined as:

Cos(d1, d2) = dot(d1,d2) / ||d1||.||d2||

Where dot(d1, d2) = d1[0]*d2[0]+d1[1]*d2[1]...

And where ||d1|| = sqrt (d1[0]^2+d1[1]^2...)

Moreover, the cosine similarity measure is a popular measure of similarity for text (which normalizes the features by covariance matrix) clustering. It captures a scale invariant understanding of similarity. An even stronger property is that the cosine similarity doesn not depend on the length. This allows documents with the same composition , but different totals to be treated identically which makes this the most popular measure for text documents. Also, due to this property, samples cane be normalized to the unit sphere for more efficient processing.

1.2.3 Nearest Neighbor

Clustering and the Nearest Neighbor prediction technique are among the oldest techniques used in data mining. Most people have an intuition that they understand what clustering is - namely that like records are grouped or clustered together. Nearest neighbor is a prediction technique that is quite similar to clustering - its essence is that in order to predict what a prediction value is in one record look for records with similar predictor values in the historical database and use the prediction value from the record that it "nearest" to the unclassified record.

1.2.4 A simple example of nearest neighbor

A simple example of the nearest neighbor prediction algorithm is that if the user looks at the people in his/her neighborhood (in this case those people that are in fact

geographically near to the user). The user may notice that, in general, the incomes of most people are somewhat similar. Thus if the user's neighbor has an income greater than 10,000 Rs. chances are good that he too has a high income. Certainly the chances that he has a high income are greater when all of his neighbors have incomes over 10,000 Rs. than if all of his neighbors have incomes of 5,000 Rs. Within his neighborhood there may still be a wide variety of incomes possible among even his "closest" neighbors but if the user had to predict someone's income based on only knowing their neighbors his best chance of being right would be to predict the incomes of the neighbors who live closest to the unknown person.

The nearest neighbor prediction algorithm works in very much the same way except that "nearness" in a database may consist of a variety of factors not just where the person lives. It may, for instance, be far more important to know which school someone attended and what degree they attained when predicting income. The better definition of "near" might in fact be other people that the user graduated from college with rather than the people that he lives next to.

Nearest Neighbor techniques are among the easiest to use and understand because they work in a way similar to the way that people think - by detecting closely matching examples. They also perform quite well in terms of automation, as many of the algorithms are robust with respect to dirty data and missing data.

1.2.5 How to use Nearest Neighbor for Prediction

One of the essential elements underlying the concept of clustering is that one particular object (whether they be cars, food or customers) can be closer to another object than can some third object. It is interesting that most people have an innate sense of ordering placed on a variety of different objects. Most people would agree that an apple is closer to an orange than it is to a tomato and that a Toyota Corolla is closer to a Honda Civic than to a Porsche. This sense of ordering on many different objects helps us place them in time and space and to make sense of the world. It is what allows us to build clusters - both in databases on computers as well as in our daily lives. This definition of nearness that seems to be ubiquitous also allows us to make predictions.

The nearest neighbor prediction algorithm simply stated is:

Objects that are "near" to each other will have similar prediction values as well. Thus if the user knows the prediction value of one of the objects he can predict it for it's nearest neighbors.

1.2.6 Where has the nearest neighbor technique been used in business?

One of the classical places that nearest neighbor has been used for prediction has been in text retrieval. The problem to be solved in text retrieval is one where the end user defines a document (e.g. Wall Street Journal article, technical conference paper etc.) that is interesting to them and they solicit the system to "find more documents like this one". Effectively defining a target of: "this is the interesting document" or "this is not interesting". The prediction problem is that only a very few of the documents in the database actually have values for this prediction field (namely only the documents that the reader has had a chance to look at so far). The nearest neighbor technique is used to find other documents that share important characteristics with those documents that have been marked as interesting.

1.2.7 Using nearest neighbor for stock market data

As with almost all prediction algorithms, nearest neighbor can be used in a variety of places. Its successful use is mostly dependent on the pre-formatting of the data so that nearness can be calculated and where individual records can be defined. In the text retrieval example this was not too difficult - the objects being documents. This is not always as easy as it is for text retrieval. Consider what it might be like in a time series problem - say for predicting the stock market. In this case the input data is just a long series of stock prices over time without any particular record that could be considered to be an object. The value to be predicted is just the next value of the stock price.

The way that this problem is solved for both nearest neighbor techniques and for some other types of prediction algorithms is to create training records by taking, for instance,

10 consecutive stock prices and using the first 9 as predictor values and the 10th as the prediction value. Doing things this way, if the user had 100 data points in his time series, he could create 10 different training records.

He could create even more training records than 10 by creating a new record starting at every data point. For instance, the user could take the first 10 data points and create a record. Then the user could take the 10 consecutive data points starting at the second data point, then the 10 consecutive data point starting at the third data point. Even though some of the data points would overlap from one record to the next the prediction value would always be different. In this example of 100 initial data points 90 different training records could be created this way as opposed to the 10 training records created via the other method.

Nearest Neighbor	Clustering
Used for prediction as well as consolidation.	Used mostly for consolidating data into a high-level view and general grouping of records into like behaviors.
Space is defined by the problem to be solved (supervised learning).	Space is defined as default n-dimensional space, or is defined by the user, or is a predefined space driven by past experience (unsupervised learning).
Generally only uses distance metrics to determine nearness.	Can use other metrics besides distance to determine nearness of two records - for example linking two points together.

Table 1-2 Some of the Differences Between the Nearest-Neighbor Data Mining Technique and Clustering

1.3 Data Mining Can Bring Pinpoint Accuracy to Sales

Data warehousing - the practice of creating huge, central stores of customer data that can be used throughout the enterprise - is becoming more and more commonplace. But data warehouses are useless if companies don't have the proper applications for accessing and using the data.

Two popular types of applications that leverage companies' investments in data warehousing are data mining and campaign management software. Data mining enables companies to identify trends within the data warehouse (such as "families with teenagers are likely to have two phone lines," in the case of a telephone company's data). Campaign management software enables them to leverage these trends via highly targeted and automated direct marketing campaigns (such as a telemarketing campaign intended to sell second phone lines to families with teenagers).

Data mining and campaign management have been successfully deployed by hundreds of Fortune 1000 companies around the world, with impressive results. But recent advances in technology have enabled companies to couple these technologies more tightly, with the following benefits: increased speed with which they can plan and execute marketing campaigns; increased accuracy and response rates of campaigns; and higher overall marketing return on investment.

Data mining automates the detection of patterns in a database and helps marketing professionals improve their understanding of customer behavior, and then predict behavior. For example, a pattern might indicate that married males with children are twice as likely to drive a particular sports car than married males with no children. A marketing manager for an auto manufacturer might find this somewhat surprising pattern quite valuable.

The data mining process can model virtually any customer activity. The key is to find patterns relevant to current business problems. Typical patterns that data mining uncovers

include which customers are most likely to drop a service, which are likely to purchase merchandise or services, and which are most likely to respond to a particular offer.

The data mining process results in the creation of a model. A model embodies the discovered patterns and can be used to make predictions for records for which the true behavior is unknown. These predictions, usually called scores, are numerical values that are assigned to each record in the database and indicate the likelihood that the customer will exhibit a particular behavior. These numerical values are used to select the most appropriate prospects for a targeted marketing campaign.

Campaign management and data mining, when closely integrated, are potent tools. Campaign management software enables companies to deliver to customers and prospects timely, pertinent, and coordinated offers, and also manages and monitors customer communications across all channels. In addition, it automates and integrates the planning, execution, assessment and refinement of possibly tens to hundreds of highly segmented campaigns running monthly, weekly, daily or intermittently.

1.3.1 Benefits of Data mining

Data mining being a very popular and interesting topic has number of advantages which are as follows:-

Provides insight into hidden patterns and relationships in user's data

- A classic example of data mining is a retailer who uncovers a relationship between sales of diapers and diaper rash cream – two items the user wouldn't normally consider as linked. The explanation is that husbands who are sent out to pick up a fresh supply of diapers are also likely to pick up diaper rash cream while they happen to be in the store – something that hadn't been recognized as a significant sales driver before data mining uncovered it.

Enables user to exploit the correlations to improve organizational performance

- Continuing the example above, very often retailers act on the relationships they discover by using tactics such as placing linked items together on end-of-isle displays as a way to spur additional purchases. All organizations can benefit from acting in a similar way – using newly discovered patterns and correlations as the basis for taking action to improve their efficiency and effectiveness.

Provides indicators of future performance

- "Those who do not learn from history are doomed to repeat it" is a famous quote from philosopher George Santayana. In the case of data mining, being able to predict outcomes based on historic data can dramatically improve the quality and outcomes of decision making in the present. As a simple example, if the best indicator of whether a customer will pay on time turns out to be a combination of their market segment and whether or not they have paid previous bills on time, then this is information the user can usefully benefit from in making current credit decisions.

Enables embedding of recommendations in user's applications

- The user can use the data mining results to display a simple summary statement and recommendations within operational applications. For example, on a credit screen user could add: "Based on this new account profile there is an 85% chance this customer will pay late. It is therefore recommended user requires a 50% prepayment on this order". Reporting on aggregate results such as Days Sales Outstanding (DSO) enables the user to measure business improvements based on when recommendations were followed and when they weren't so that the user can fine-tune his model and recommendations over time for optimal effect.

1.4 Existing techniques

Various algorithms have been developed to avoid the problems of association rule mining such as the multiple scans and generation of large candidate itemset. Following are some of these algorithms:-

Apriori is a seminal algorithm proposed by R.Agrawal[1] in May 1993. It uses *prior knowledge* of frequent itemset properties. Apriori uses breadth-first search and a hash tree structure to count candidate item sets efficiently. But its drawback is that the finding of each L_k requires one full scan of database. Hence, due to multiple scans we get wastage of resources like time and space /memory in addition to the counting of false candidates. Candidate generation generates large numbers of subsets (the algorithm attempts to load up the candidate set with as many as possible before each scan).

FP-growth algorithm[6] proposed by J. Han, J. Pei, Y. Yin, and R. Mao in 2004 adopts a *divide and conquer* strategy avoiding costly candidate generation. First, it compresses the database representing frequent items into a **frequent pattern tree or FP tree,** which contains the itemset association information. It then divides the compressed database into a set of *conditional databases.* FP-growth tree is memory resident and requires additional storage in every node of the FP-tree (Because of excessive pointers storage in every node) especially when the FP- tree is too large to fit in main memory.

Partition algorithm was proposed by **A. Savasere[2]** in 1996. This algorithm is used for partitioning the data to find candidate itemsets. A partitioning technique can be used that requires just two database scans to mine the frequent itemsets.

The problem of accurately estimating the number of partitions given the available memory, however, needs further work.

Sampling approach was proposed by **Toivonen[3]** in 1996. This algorithm is used for mining on a subset of the given data. The basic idea of the sampling approach is to pick a random sample S of the given data D, and then search for frequent itemsets in S instead of D. In this way, there is some tradeoff of accuracy against efficiency. The sample size

of S is such that the search for frequent itemsets in S can be done in main memory, and so only one scan of the transactions in S is required overall. We might miss some of the global frequent itemsets since we are searching for frequent itemsets in S than in D. The sampling approach is especially beneficial when efficiency is of utmost importance, such as in computationally intensive applications that must be run on a very basis. However, there is a tradeoff between accuracy and efficiency.

MaxMiner (Bayardo, 1998)[4] is another algorithm for finding the maximal elements. It uses Rymon R(1992)[10] "search through systematic set Enumeration" mechanism and efficient pruning techniques to quickly narrow the search. MaxMiner employs a breadth-first traversal of the search space; it reduces database scanning by employing a look ahead pruning strategy. Since MaxMiner uses the original horizontal database format, it can perform the same number of passes over a database as Apriori does. Hence, there will be need for scanning multiple times.

ECLAT (Equivalence CLASS transformation)[5] is an algorithm developed by M.J Zaki, which transforms a given data set of transactions in the horizontal data format of *TID-itemset* into the vertical format of *item-TID-set.* It mines the transformed data set by TID-set intersections based on Apriori property and additional optimization techniques such as *diffset.* However, the cost of registering long TID_sets is high.

The above literature shows that association rule mining is facing a number of problems currently such as multiple scans of database and generation of large candidate itemsets which needs to be solved.

1.5 Scope of the Project

Data mining has become very popular area for research where association rule mining plays a vital role. Association rule mining is not only used in businesses, retail sales but also in science and engineering, telecommunications, games, human resource departments etc. Data Mining is a highly effective tool in the catalog marketing industry.

Catalogers have a rich history of customer transactions on millions of customers dating back several years. Data mining tools can identify patterns among customers and help identify the most likely customers to respond to upcoming mailing campaigns.

In applying our devised Cos_FIS generator algorithm, the main problem that may be faced is limited memory and huge processing needed. Moreover, it needs a lot of time while scanning the database. The volume of the database if large may also create problems as it is not easy to handle it.

Chapter 2

****

LITERATURE SURVEY

2. Literature Survey

Market Basket Analysis is a modelling technique based upon the theory that if a costumer buys a certain group of items, he/she is more (or less) likely to buy another group of items. For example, if the user is in a restaurant and he orders apple juice and doesn't order pizza, he is more likely to order crisps at the same time than somebody who didn't order apple juice.

The set of items a customer buys is referred to as an **itemset**, and market basket analysis seeks to find relationships between purchases.

Typically the relationship will be in the form of a rule:

IF {Apple juice, no Pizza} THEN {crisps}.
The probability that a customer will order apple juice without a Pizza(i.e. that the antecedent is true) is referred to as the **support** for the rule. The conditional probability that a customer will purchase crisps is referred to as the **confidence**.

Consider a supermarket with a large collection of items. Typical business decisions that the management of the supermarket has to make include what to put on sale, how to design coupons, how to place merchandise on shelves in order to maximize the product sales etc. Analysis of past transaction data is a commonly used approach in order to improve the quality of such decisions. Until recently, however, only global data about the cumulative sales during sometime period a day, a week, a month, etc. was available on the computer. Progress in bar-code technology has made it possible to store the so called basket data that stores items purchased on a per-transaction basis. Basket data type transactions do not necessarily consist of items bought together at the same point of time. It may consist of items bought by a customer over a period of time. Examples include monthly purchases by members of a book club or a music club.

Following is some of the research work done previously in association rule mining. Each research paper represents an algorithm and its advantages and disadvantages.

database D must occur as a frequent itemset in atleast one of the partitions. Thus, all local frequent itemsets are candidate itemsets with respect to D. The collection of frequent itemsets from all partitions forms the **Global candidate itemsets** with respect to D. In phase II, a second scan of D is conducted in which the actual support of each candidate is assessed in order to determine the global frequent itemsets.

Moreover, in this paper, Partition algorithm has been described as not only efficient but also fast for discovering association rules in large databases. An important contribution of this algorithm is that it drastically reduces the I/O overhead associated with previous algorithms. This feature may prove useful for many real-life database mining scenarios where the data is most often centralized resource shared by many user groups, and may even have to support on-line transactions. Interestingly, this improvement in disk I/O is not achieved at the cost of CPU overhead. It is demonstrated with extensive experiments that the CPU overhead is actually less than the best existing algorithm for low minimum supports (i.e., cases which are computationally more expensive). In addition, the algorithm has excellent scale-up property. The problem of accurately estimating the number of partitions given the available memory, however, needs further work

2.3 Sampling large databases for association rules

Sampling approach was proposed by Toivonen[3] in 1996. This algorithm is used for mining on a subset of the given data. The basic idea of the sampling approach is to pick a random sample S of the given data D, and then search for frequent itemsets in S instead of D. In this way, there is some degree of tradeoff of accuracy against efficiency. The sample size of S is such that the search for frequent itemsets in S can be done in main memory, and so only one scan of the transactions in S is required overall. The user might miss some of the global frequent itemsets since he is searching for frequent itemsets in S than in D. To lessen this possibility, a lower support threshold is used than minimum support to find the frequent itemsets local to S (denoted as Ls). A mechanism is used to determine whether all of the global frequent itemsets are included in Ls. If Ls contains all of the frequent itemsets in D, then only one scan of D is required.

The sampling approach is especially beneficial when efficiency is of utmost importance, such as in computationally intensive applications that must be run on a very basis. However, there is a tradeoff between accuracy and efficiency. The penalty in partition/sampling [9] is that candidate set derived is necessarily a superset of the actual set of frequent itemsets and may contain many false positives.

2.4 Tree Structure for Mining Association rules

MaxMiner (Bayardo, 1998)[4] is another algorithm for finding the maximal elements. It uses Rymon's (1992)[10] "search through systematic set Enumeration" mechanism and efficient pruning techniques to quickly narrow the search. MaxMiner employs a breadth-first traversal of the search space; it reduces database scanning by employing a look ahead pruning strategy, i.e., if a node with all its extensions can determine to be frequent, there is no need to further process that node. It also employs item (re)ordering heuristic to increase the effectiveness of superset-frequency pruning.

Since MaxMiner uses the original horizontal database format, it can perform the same number of passes over a database as Apriori does. Hence, there will be need for scanning multiple times.

2.5 Efficiently mining long patterns from databases

ECLAT (Equivalence CLASS transformation)[5] is an algorithm developed by M.J Zaki, which transforms a given data set of transactions in the horizontal data format of *TID-itemset* into the vertical format of *item-TID-set.* It mines the transformed data set by TID-set intersections based on Apriori property and additional optimization techniques such as *diffset.* In this way the support of an itemset X can be easily computed by simply intersecting the covers of any two subsets Y,Z $\subseteq$ X, such that Y$\cup$Z=X. In this algorithm, for each frequent item I, the I-projected database D^i is created. This is done

by first finding every item j that frequently occurs together with i. The support of this set {I, J} is computed by intersecting the covers of both items. If {i,j} is frequent then j is inserted into D^i together with its cover then algorithm is called recursively to find all FIS in the new database D^i.

Éclat algorithm uses support based measure to find maximal frequent IS by using I-projected databases technique but this algorithm generates large number of candidate set to derive frequent item set at each iteration of the algorithm. Less memory is required as compare to FP-growth to find FIS. Moreover, the cost of registering long TID_sets is high.

2.6. Search through systematic set enumeration

FP-growth algorithm[6] proposed by J. Han, J. Pei, Y. Yin, and R. Mao in 2004 adopts a *divide and conquer* strategy avoiding costly candidate generation. First, it compresses the database representing frequent items into a **frequent pattern tree or FP tree,** which retains the itemset association information. It then divides the compressed database into a set of *conditional databases,* each associated with one frequent item or "Pattern fragment" and mines each such database separately.

First, it scans the Database D and collects F, the set of frequent items, and their support counts. F is sorted in support count descending order as L, the list of frequent items. Next, it creates the root of an FP-tree, and labels it as "NULL" for each transaction *Trans* in D it does the following:-

- Selects and sorts the frequent items in *Trans* according to the order of *L*. Let the sorted frequent item list in *Trans* be *[p|P], where p is the first element and P is the remaining list.* Then insert_tree(*[p|P], T*) is called which is performed as follows. If *T* has a child *N* such that N.item_name =p.item_name, then increment N's count by 1 else create a new node N and let its count b 1, its parent link be linked to *T* and it's node-link to the nodes with the same *item_name* via the node-link structure. Finally, the FP-tree is mined.

FP-growth tree is memory resident and requires additional storage in every node of the FP-tree (Because of excessive pointers storage in every node) especially when the FP-tree is too large to fit in main memory. However, it is efficient and scalable for mining both long and short frequent patterns.

2.7 New Algorithms for fast discovery of Associations Rules

Goethals (2004) presented MEDIC Algorithm[7] which generates all itemsets containing item i as soon as there can be no transaction anymore that contain i. One transaction processes at a time in lexicographic order. After generating all these itemsets, the cover of I can be removed from main memory. After that the transaction identifier of the current transaction is added to the cover of all items occurring in that transaction.

Medic is a frequent set mining algorithm. Medic is also based on support count measure and utilizes the ECLAT algorithm for mining the frequent item sets. Medic uses much less memory than Eclat because the database is never entirely loaded into main memory.

2.8 Similarity based mining for finding frequent itemsets

SB-Miner is novel algorithm to find FIS based on clustering measure i.e. jacquard similarity measure[11]. Jacquard similarity measure is based on calculating the distance between itemsets. Proposed technique makes use of prefix tree as data structure and vertical database layout to cluster related items together. The experimental results have proved that the same FIS can be generated by SB-miner technique as compared to other Apriori based algorithms. This also showed that various clustering measures can be applied for association rules mining. The research work in this paper is basically extension or improvement of the work to prove that clustering measure like cosine similarity is again a candidate clustering measure which can be used to generate FIS.

2.9 Selecting the right interestingness measure for association patterns

The DISJOINT and RANDOM algorithms are two table selection algorithms used to select a small set of tables [12] such that an expert can select a desirable measure by looking at just this small set of tables. Many techniques for association rule mining and feature selection require a suitable metric to capture the dependencies among variables in a data set such as support, confidence, lift etc are used to determine interestingness of association patterns. However, many such measures provide conflicting information about the interestingness of a pattern and best metric is rarely known. In this paper, an overview of various measures proposed in the statistics, machine learning and data mining literature, is presented. Moreover, there is a description of several key properties one should examine in order to select the right measure for a given application domain. Also, a comparative study of these properties is made using twenty one of the existing measures. Two scenarios are presented in which most of the existing measures agree with each other, namely support-based pruning and table standardization. The RANDOM algorithm randomly selects k out of the overall N tables and presents them to the experts. Whereas, DISJOINT algorithm selects k tables that are "furthest" apart according to their average rankings and would produce the largest amount of ranking conflicts i.e. large standard deviation in their ranking vector.

2.10 Problem Statement

In clustering, nearest data points are brought together. This can be done either using Cosine similarity or dissimilarity measures. Similar data items will be nearest to each other while Dissimilar will be at distance for apart. Association also finds most frequent items in a dataset. If a subset is found frequently in data set then it can be said that its similarity is high. So frequent item set can be found on the basis of Cosine similarity measures as well.

From the literature survey, it is observed that rules are generated on the basis of candidate itemsets, which are generated using support and confidence measures. Moreover, the database is scanned multiple times to generate candidate itemsets.

After scanning the database, the items are taken from database and their respective support count is stored in a table. Then, 2-itemsets are created and database is scanned till k-itemset is created. Multiple scanning is an extensive workload on the database specially when the mining of association rules is done on a huge database.

Apriori [1], while historically significant, suffers from a number of inefficiencies or trade-offs, which have spawned other algorithms. Candidate generation generates large numbers of subsets (the algorithm attempts to load up the candidate set with as many as possible before each scan).

The problem of accurately estimating the number of partitions given the available memory in PARTITION algorithm [2] needs further work.

The penalty in partition/sampling [3] is that candidate set derived is necessarily a superset of the actual set of frequent itemsets and may contain many false positives.

Éclat algorithm presented in [5] uses support based measure to find maximal frequent IS by using I-projected databases technique but this algorithm generates large number of candidate set to derive frequent item set at each iteration of the algorithm. Moreover, the cost of registering long TID_sets is high.

FP-growth tree [6] is memory resident and requires additional storage in every node of the FP-tree (Because of excessive pointers storage in every node) especially when the FP-tree is too large to fit in main memory.

In conclusion, the main problem in all the papers and algorithms is the multiple database scanning as well as the generation of large candidate itemsets. Whereas the *"Cos_FIS*

Chapter 3

PROBLEM DOMAIN & PROPOSED SOLUTION

3. Problem Domain and Proposed Solution

Data analysis is the process of looking at and summarizing **data** with the intent to extract useful information and develop conclusions. Data analysis is closely related to data mining, but data mining tends to focus on larger data sets, with less emphasis on making inference, and often uses data that was originally collected for a different purpose. In statistical applications, some people divide data analysis into descriptive statistics, exploratory data analysis(EDA) and confirmatory data analysis, where the EDA focuses on discovering new features in the data, and CDA on confirming or falsifying existing hypotheses.

Data mining is the process of sorting through large amounts of data and picking out relevant information. It is usually used by business intelligence organizations, and financial analysts, but is increasingly being used in the sciences to extract information from the enormous data sets generated by modern experimental and observational methods. It has been described as "the nontrivial extraction of implicit, previously unknown, and potentially useful information from data and the science of extracting useful information from large data sets or databases". Data mining in relation to enterprise resource planning is the statistical and logical analysis of large sets of transaction data, looking for patterns that can aid decision making

Another example of data mining, often called the market basket analysis, relates to its use in retail sales. If a clothing store records the purchases of customers, a data-mining system could identify those customers who favor silk shirts over cotton ones. Although some explanations of relationships may be difficult, taking advantage of it is easier. The example deals with association rules within transaction-based data. Not all data are transaction based and logical or inexact rules may also be present within a database. In a manufacturing application, an inexact rule may state that 73% of products which have a specific defect or problem will develop a secondary problem within the next six months.

Data Mining is a highly effective tool in the catalog marketing industry. Catalogers have a rich history of customer transactions on millions of customers dating back several years. Data mining tools can identify patterns among customers and help identify the most likely customers to respond to upcoming mailing campaigns.

In data mining, **association rule mining** is a popular and well researched method for discovering interesting relations between variables in large databases. Piatetsky-Shapiro describes analyzing and presenting strong rules discovered in databases using different measures of interestingness, among which Cosine measure is one of the most useful measure.

Association rules are required to satisfy a user-specified minimum support and a user-specified minimum confidence at the same time. To achieve this, association rule generation is a two-step process.

1. First, minimum support is applied to find all *frequent itemsets* in a database.
2. In a second step, these frequent itemsets and the minimum confidence constraint are used to form rules. While the second step is straight forward, the first step needs more attention.

3.1 Problem Domain

Most of the existing methods of finding FIS are based on support measure. The purpose of this thesis is to develop a novel data mining algorithm to find out association which will find FIS on the basis of Cosine similarity measure rather than on the basis of support count.

Although Support and confidence measures help exclude the exploration of a good number of uninteresting rules, many rules so generated are still not interesting to the users. Unfortunately, this especially true when *mining at low support threshold or mining for long patterns*. This has been one of the major bottlenecks for successful application of association rule mining. It is known that support and confidence measures are insufficient at filtering out uninteresting association rules. To tackle this weakness, a correlation measure such as COSINE measure can be used to augment the support-confidence framework

Moreover, the following challenges of frequent pattern mining need to be met:

- Multiple scans of transaction database / I/O overhead.
- Huge number of candidates

- Tedious workload of support counting for candidates.

3.1.1 Multiple Database scans

The major problem of Association rule mining is the multiple database scans, since first frequent itemsets are searched in the database and then 2-itemset and k-itemsets are created to find similarity between the two large itemsets which requires consulting the database again and again. Moreover, since the frequent itemset generation is also performed on huge databases and large datawarehouses, there is chance of multiple Disk I/Os which are the main obstacle in efficiency of database and association rule mining algorithms. Therefore, the primary goal in association rule mining should be to reduce database scans and the disk I/Os.

3.1.2 Large candidate set size

Creation of candidate itemsets resembles to a chain process i.e. 1-itemsets are used to create 2-itemsets and 2-itemsets are used to create k-itemsets and so on. Hence, the more the size of the candidate itemset, the more complicated will the rules be and the more time would it take to execute the algorithm for generating frequent itemsets. Also, since the number of rules wil be numerous then, finding interesting rules will be time consuming and hard. Moreover, the size and quantity of itemsets also leads to disk and I/O overhead. The number of database scans required by Apriori-based algorithms depends on the size of the largest large itemsets.

3.1.3 Algorithm execution time

One of the main challenges in database mining is developing fast and efficient algorithms that can handle large volumes of data as most mining algorithms perform computation over the entire database and often the databases are very large. Time management is the key factor in any algorithm for the fast retrieval of results of the

queries specially in todays world where there is huge amount of data and shortage of time. The faster the frequent itemsets are generated, the faster would the process of sales and promotion of products be, since the consequent rules would be generated and reviewed faster. For example, suppose If a clothing store records the purchases of customers, a data-mining system could identify those customers who favour silk shirts over cotton ones. The faster this system would identify such customers before the respective season comes, the more increase will be observed in the retail sales and the process of importing the demanded cloth would be faster and easier.

3.2 Proposed Solution

To achieve good runtime performance and efficient running of association rule mining algorithm, the above mentioned issues should be considered and solutions to these problems must be found to prevent performance degradation. For instance, lessen the number of database scans and reduce huge number of candidate itemsets.

3.2.1 Reduced Database scans

All the algorithms in the association rule mining need to scan the database multiple times, which not only causes overhead on the disk I/O but is also time consuming. Multiple scans are needed in order to create 1-itemset, 2-itemset and k-itemset as well as to keep track of support counts. However, the *Cos_FIS_generator* performs only a single scan of an item over the whole database while reading transactions' ID list from the .txt file and then stores the count of the transaction in the cache. Once the count of each transaction is maintained in the cache, the user can use a formula for Cosine measure which will show if a certain itemset is frequent or no where a certain Cosine measure threshold is given. This single scanning of a particular item will reduce the I/O overhead.

Moreover, among various layouts of the database, horizontal and vertical layouts are the most common. Horizontal layout consists of the list of transactions. Each transaction has an identifier followed by list of items. The vertical layout however consists of list of items. Each item has a transaction IDs list- the list of all transactions containing the item. Therefore, the algorithm in this thesis uses the vertical layout of the database since this format performs only a single scan while reading the transaction row by row and storing its support count meanwhile in cache. Also relevant transactions can be clustered together.

3.2.2 No candidate itemset generation

Another main problem with most of the association rule mining algorithms is the size of the candidate itmsets . The size of these itemsets is sometimes too large that its hard to find association rules. Moreover, it is very time consuming as well as storage of these candidates becomes harder. All these factors affect the whole process of association rule mining specially when the database is very huge. Finding candidate itemsets and then pruning them is a very tedious job. However, the *"Cos_FIS generator"* algorithm works without candidate generation. In this algorithm, only the pairs of items are created. Then, it is found out through the Cosine formula IS, whether a certain pair or itemset is frequent or no. If the pair is frequent, it is added in the Set Enumeration tree else it is discarded.

3.2.3 Reduced Algorithm Execution time

In today's era, fast and efficient algorithms are demanded as time has become a key element in one's life. Also, since such algorithms are required to handle large amounts of data in data warehouses and perform numerous computations, the faster the algorithm executes, the faster the Frequent itemsets are generated. Therefore, the

Cos_FIS generator creates the frequent itemsets faster. Also, the single scan of the database performed by this algorithm contributes to the speed of generation of FIS.

Chapter 4

****

SYSTEM DESIGN

4. System design

The Cos_FIS generator takes the .txt file of 0's and 1's as an input which shows absence and presence of data items respectively. The .asc file is first converted into .txt file. Association rule mining comprises of two steps:-

1. **Finding all frequent itemsets:** the itemsets which satisfy the given threshold will be frequent.
2. **Generating strong association rules from these frequent itemsets:** the rules must also satisfy the minimum support.

However, this research is limited to the first portion of the association rule mining that is the generation of the frequent itemsets.

4.1 Representation of the frequent itemsets

The frequent itemsets are represented into a node structure i.e. in form of a *set enumeration tree.* The *Set-Enumeration* (SE)-tree [10] is a vehicle for representing and/or enumerating sets in a best-first fashion. The *complete* SE tree systematically enumerates elements of a power-set using a pre-imposed order on the underlying set of elements. In problems where the search space is a subset of that power-set that is (or can be) closed under set-inclusion, the SE-tree induces a complete irredundant search technique.

4.1.1 Node structure

In this thesis, each node in SE tree represents a frequent itemset. Each node has **node_id** field, which shows the name of the node and also indicates frequent itemset. Node has a **count** field indicating the total number of transactions containing that frequent itemset. There are two node pointers down and right node pointer pointing to the node that is linked to current node in downward position level-wise and to the right position item-wise respectively. In other words, the down pointer points to the node in the next level where each level represents the k-itemset and the right pointer points to the

node of the same level. Moreover, there is a **transaction_id_list** field, which actually represents the transaction IDs of transactions containing the particular FIS represented by the **node_id** of the node.

4.1.2 Creation of Set Enumeration Tree

The Set Enumeration tree(SE tree) is created in a very systematic way. First node 'a' is created and then the column of dataset (text file) is scanned according to a pointer, hence filling the buffer with the support count. If the support count of the node is greater than or equal to the minimum user specified threshold, then the node is said to be frequent and hence is added in the SE tree. After the buffer is filled with count of the node, it is then copied into the info field of newly created node. It is then linked with the previous node. This way the first level of SE tree is generated.The second level of SE tree is created by finding Cosine similarity between every two itemsets or nodes present in the first level of SE tree i.e) between node 'a' and node 'b'. Only those itemsets whose similarity is greater than or equal to the user supplied minimum similarity threshold are declared frequent and are linked in the second level of the SE tree in lexicographic order. And the same process is repeated for levels 3 and 4.

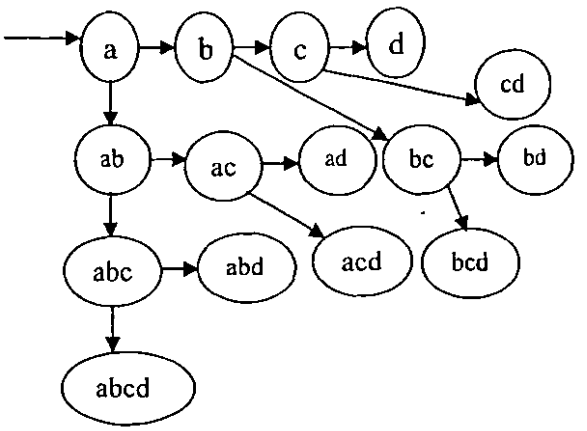


Fig 4.1 Set Enumeration Tree

4.2 The Cos_FIS generator algorithm

Input

Ds: Transactional Dataset
α : Threshold

Output:

FIS: Frequent itemsets
β : No. of FIS after each iteration

Step 1 : Scan transactional dataset DS.
Step 2: Construct first level of Prefix tree.
Step 3 :Construct second level of prefix tree by finding similarity between every 2 itemsets in the previous level of tree as shown in step 4.

Step 4: β = Generate_frequent_2_itemset(FIS, α)

Repeat Step 5 to 7 until $\beta = 0$.

Step 5: Scan each sub tree in the last level of prefix tree.
Step 6: Store starting node's address of each sub tree in S.
Step 7: β = Generate_next_frequent_itemset (S,FIS, α).
Step 8: Return FIS.

4.3 System's Major Modules

Division of any project into modules adds to its efficiency and overall performance. Hence this project is divided into following modules:-

1. Database conversion.
2. File reading.
3. Copying cache to node.
4. Frequent itemset generation.

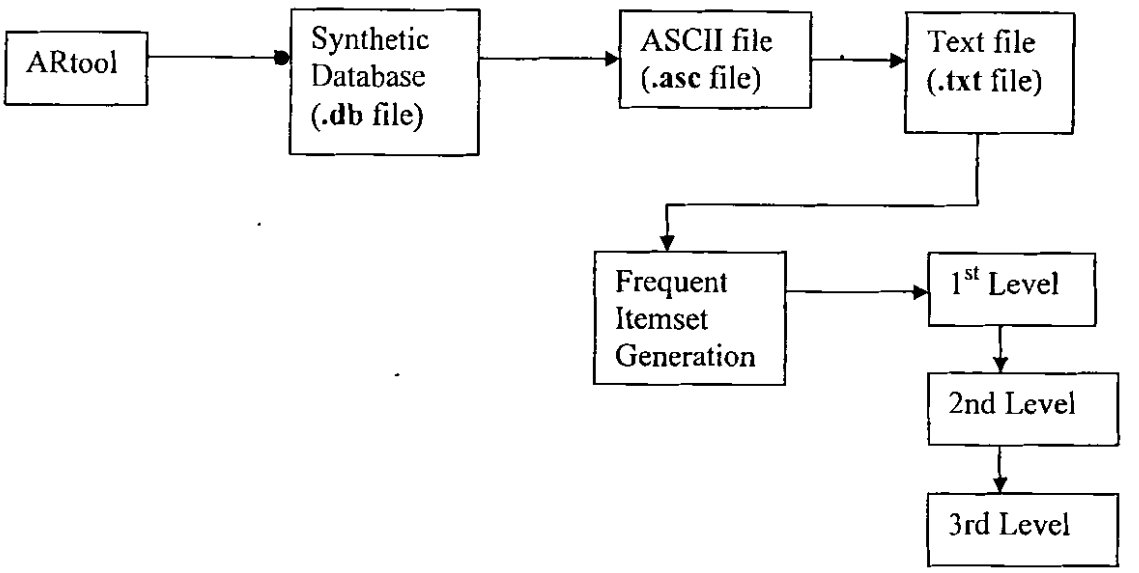


Fig 4.2 Architectural Diagram

4.4 Database Conversion

ARtool is an application for mining association in rules in binary databases. This tool has utility to generate ***synthetic binary databases***. The databases generated by using ARtool are in a specific format to be used only with this tool. But there was a need of having the synthetic database in format required to be used with Cos_FIS generator. Therefore first the database file is converted into ASCII format(.asc file) by the utility available in ARtool i.e. db2asc and then this ASCII format is converted into binary database format (.txt file) used in Cos_FIS_generator algorithm. By using the above technique, databases described in the table 4-1 are generated.

Database	T	AT	I	P	AP
T200AT6I10P5AP4.db	200	6	10	5	4
T500AT5I10P5AP3.db	500	5	10	5	3
T400AT6I10P5AP4.db	400	6	10	5	4

Table 4-1 Synthetic binary databases

In table 4-1, T is number of transaction, AT is average size of transaction, I is the number of items, P is number of patterns and AP is average size of patterns.

ARtool uses a custom format for its database files (which will be henceforth referred to as the .db format and is identical to the format used in ARMiner). The asc2db and db2asc are utilities that allow the conversion of a .db file to a specially formatted ASCII file (user will refer to this as .asc) and respectively the conversion of a .asc file into a .db file. The .asc files can be easily read and modified with any decent ASCII editor.

These formats can be best explained by taking a small example of supermarket data. Suppose the items sold by a (very, very small) shop are green apples, red apples, oranges, bananas, and grapes. Also suppose that the user had three customers, one bought green apples and grapes, one bought only oranges, and the last one bought oranges and grapes. This activity can be represented in the .asc format as follows:

```
1 green apples
2 red apples
3 oranges
4 bananas
5 grapes
```

Fig 4.3 Part 1 of ASCII file

```
BEGIN_DATA
1 5
3
3 5
END_DATA
```

Fig 4.4 Part 2 of ASCII file

There are two distinct parts of this file, the first one illustrated in fig 4.3 contains a listing of all the items user can sell, or otherwise said, of all the items that could participate in a transaction.

The format is pretty simple. It must consist of a positive number followed by a string (which can contain blank spaces). It is important that the numbers be assigned in increasing order starting from 1. Empty lines are allowed to appear in this section. This section enumerates all entities described by the data and between which ARtool will later be used to look for association rules The second part illustrated in fig 4.4 consists of the actual data.

In this case, there were 3 transactions and these are each represented on a separate line. The first transaction involved green apples and grapes and they are represented by the numbers associated in the first section, that is 1 for green apples and 5 for grapes.

The **db2asc** program in ARtool converts a .db file to .asc format. This can be useful if the user wants to read or verify the content of a .db file. He/she can also use it to modify by hand the contents of a .db file by first converting it to a .asc file, then editing the .asc file, and finally converting it back to a .db file.

db2asc is used in a similar way to its counterpart, **asc2db**. If the user needs to convert the artdata.db database to .asc format, then he/she can type the following command in the MS- DOS command prompt:

java db2asc artdata

which will produce an artdata.asc file. If the user wants a different name for the output, then he can pass it on the command line as a second argument:

java db2asc artdata arttxt

which will produce an arttxt.asc file representing the contents of the artdata.db database.

4.4.1 Database layout

It can be observed that after the conversion of ASCII file into text file, the database is being represented in **vertical format**. Among various layouts of the database horizontal and vertical layout are very much common layouts as shown in the Figure 4.3. Horizontal layout consists of list of transactions. Each transaction has an identifier followed by list of items. The vertical layout consists of list of items. Each item has a transaction IDs list- the list of all the transactions containing the item. Vertical database format has numerous advantages i.e. multiple scans of the database can be avoided and relevant transactions can be clustered together.

DATABASE	
TID	ITEMS
1	A, B, E
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	A, B, C, E
9	A, B, C

HORIZONTAL ITEM SET					
1	A	B	E		
2	B	D			
3	B	C			
4	A	B	D		
5	A	C			
6	B	C			
7	A	C			
8	A	B	C	E	
9	A	B	C		

VERTICAL TID SET				
A	B	C	D	E
1	1	3	2	1
4	2	5	4	8
5	3	6		
7	4	7		
8	6	8		
9	8	9		
	9			

Table 4-2 Database layouts

4.5 File reading

Once the ASCII file(.asc) is converted into text format (.txt) by the C code, it is read or scanned. The text file is in form 0's and 1's which show the absence or presence of a certain item in a particular transaction. This scanning is performed in order to keep track of the support count of each item i.e. the number of 1's in a certain item which shows the presence of that particular item in a certain transaction. The number of 1's is equal to the support count of an item. The probability of an item is found by dividing the

support count by the total number of transactions. This probability will then be used in the Cosine measure Formula to find the frequent itemset or to see if a certain itemset is frequent.

4.6 Copying Cache to Node

After first node is created, the columns of dataset are scanned and the buffer is filled with the information of a certain item or node. This buffer is then copied by a function named **copybuffer** in to newly created node info field. Each item presents a node in the set enumeration tree. Each node in SE-tree represents a frequent itemset. Each node has node_ID field, which shows the name of the node and also indicates frequent itemset. There is an transaction_ID_list field in every node containing the transaction IDs list, which actually represents the transaction IDs of the transaction containing the particular FIS represented by the tag of the node.

4.7 Frequent Itemset Generation

Once the user gets the support count of a certain itemset, he/she can easily calculate its probability by dividing the support count with the total number of transactions. After getting the probability of all the itemsets, he/she can easily find out whether a ceratin itemset is frequent or no. In other words, he can find the frequent itemsets /similarity between two items by applying the following ***Cosine masure*** formula to the itemsets:

$$IS_{ABsim}=P(AUB) / \sqrt{P(A)X}\sqrt{P(B)}$$

Where A is the first item and B is the second item, P(A) and P(B) is the probability of item A and item B respectively, P(AUB) is the combined probability.

After applying the formula to a certain itemset, if the similarity user gets from this Cosine formula is greater than or equal to the user specified similarity threshold, then that itemset is said to be frequent and will be linked in the 2^{nd} level of the SE tree in lexicographic order.

Chapter 5

****

IMPLEMENTATION

5. Implementation

This section covers all the aspects of the implementation of the ***Cos_FIS generator*** algorithm. This project is divided into functions or modules which are discussed one by one. Furthermore, the functionality of tools used in this project is also explained.

Efficient running of any system depends upon the software used in it and accurate input. The software used in the project should be capable of meeting not only the user's requirements but also of the proposed system. The tools/softwares used in this system are **ARtool, Notepad** for the text format files, **Wordpad** for the ASCII files and the Java platform for the running of **ARtool** i.e. sun's **JDK (Java Development Kit)or JRE(Java Runtime Environment).**

5.1 ARtool

ARtool v1.1.2 is a Java application for mining frequent itemsets and association rules in binary databases. ARtool is free software distributed under the GNU General Public License.

ARtool uses a custom format for its database files (which will henceforth referred to as the .db format and is identical to the format used in ARMiner). The asc2db and db2asc are utilities that allow the conversion of a .db file to a specially formatted ASCII file (user will refer to this as .asc) and respectively the conversion of a .asc file into a .db file. The .asc files can be easily read and modified with any decent ASCII editor.

ARtool comprises three components: a set of Java packages, a set of command line tools, and a graphical user interface (GUI).

Moreover, it also helps to generate a synthetic database first which is then converted into ASCII format (.asc file) by using the command line prompt as mentioned previously in section 4.2.

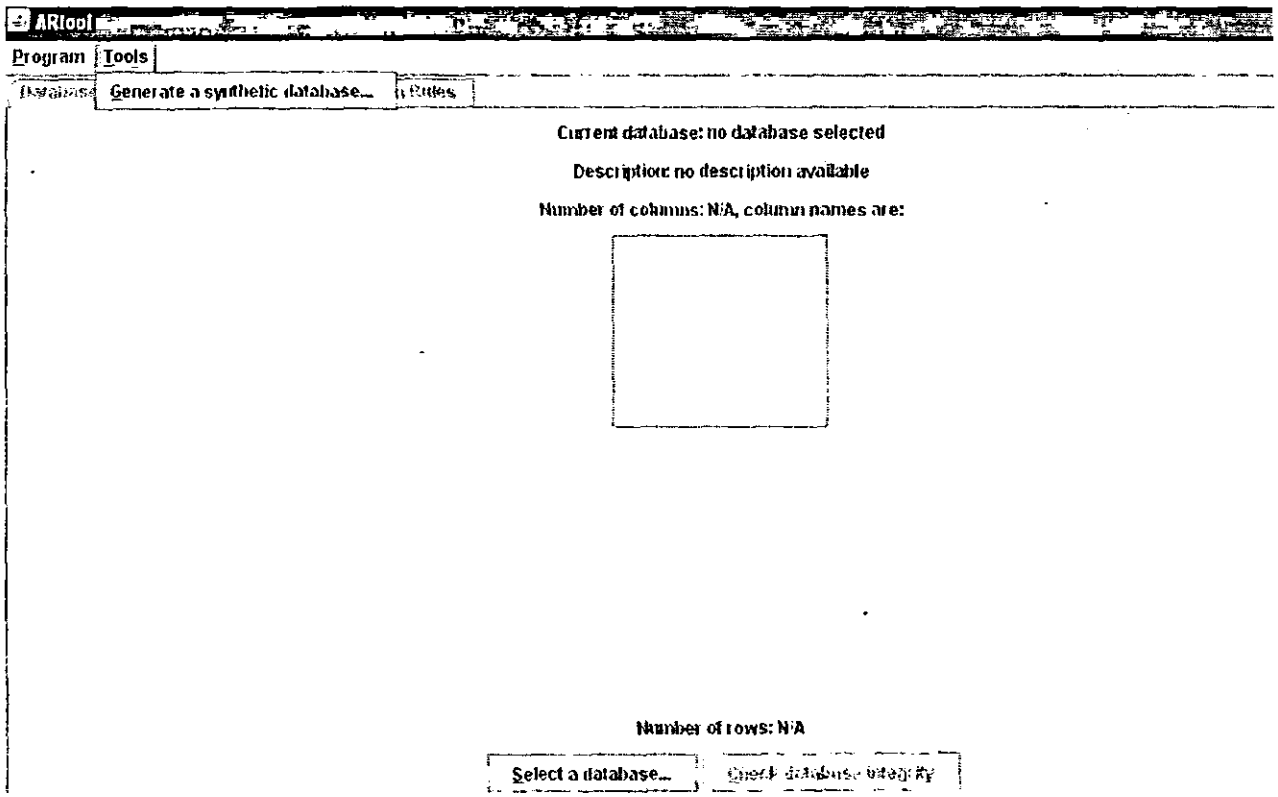


Fig 5.1 ARtool User Interface

5.1.1 How to install and execute ARtool

The user needs to have Sun's JDK or JRE installed, probably at least version 1.3.

To install ARtool, user will just unzip the ARtool binaries to some directory on his hard drive.

To run the ARtool GUI, user will just type:

java -jar ARtool.jar

or will simply double-click on ARtool.jar (works only if he has JRE installed).

5.1.2 Features of ARtool

The new features of ARtool are:

- a set of command line tools that allow mining, synthetic database generation, operations on databases, etc
- the GUI gives more information about a selected database
- the GUI gives more information about the frequent itemsets
- the GUI has a log window that keeps track of all operations performed
- algorithm execution and database generation can now be interrupted
- lengthy tasks are executed in threads and do not freeze the interface
- an online help system - provides a quick introduction to association rule mining and to using ARtool
- the GUI is easier to navigate since I use dialogs sparingly
- there are plenty of tooltips to help the novice user

5.2 Java Platform

As it is known that in order to run ARtool, user needs Java platform i.e Sun's JDK or JRE. Therefore, JDK (Java development Kit) was installed in this system.

5.2.1 JDK (Java Development Kit)

The **Java Development Kit (JDK)** is a **Sun** Microsystems product aimed at Java developers. Since the introduction of Java, it has been by far the most widely used Java SDK. On 17 November 2006, Sun announced that it would be released under the **GNU** General Public License (GPL), thus making it free software.

The JDK is a subset of what is loosely defined as a software development kit (SDK) in the general sense. In the descriptions which accompany their recent releases for Java SE, EE, and ME, Sun acknowledge that under their terminology, the JDK forms the subset of the SDK which is responsible for the writing and running of Java programs. The remainder of the SDK is composed of extra software, such as Application Servers, Debuggers, and Documentation.

The JDK also comes with a complete **Java Runtime Environment**, usually called a *private* runtime. It consists of a **Java Virtual Machine** and all of the class libraries that will be present in the production environment, as well as additional libraries only useful to developers, such as the internationalization libraries and the IDL libraries.

5.2.2 JRE (Java Runtime Environment)

The JVM Java virtual machine, which is the instance of the JRE (Java Runtime Environment), comes into action when a Java program is executed. When execution is complete, this instance is garbage-collected. JIT is the part of the JVM that is used to speed up the execution time. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.

A **Java Virtual Machine (JVM)** is a set of computer software programs and data structures which use a virtual machine model for the execution of other computer programs and scripts. The model used by a JVM accepts a form of computer intermediate language commonly referred to as Java bytecode. This language conceptually represents the instruction set of a stack-oriented, capability architecture. The JVM is a crucial

component of the Java Platform.Programs intended to run on a JVM must be compiled into a standardized portable binary format, which typically comes in the form of .class files. A program may consist of many classes in different files. For easier distribution of large programs, multiple class files may be packaged together in a .jar file (short for Java archive).The JVM runtime executes .class or .jar files, emulating the JVM instruction set by interpreting it, or using a just-in-time compiler (JIT) such as Sun's HotSpot. JIT compiling, not interpreting, is used in most JVMs today to achieve greater speed

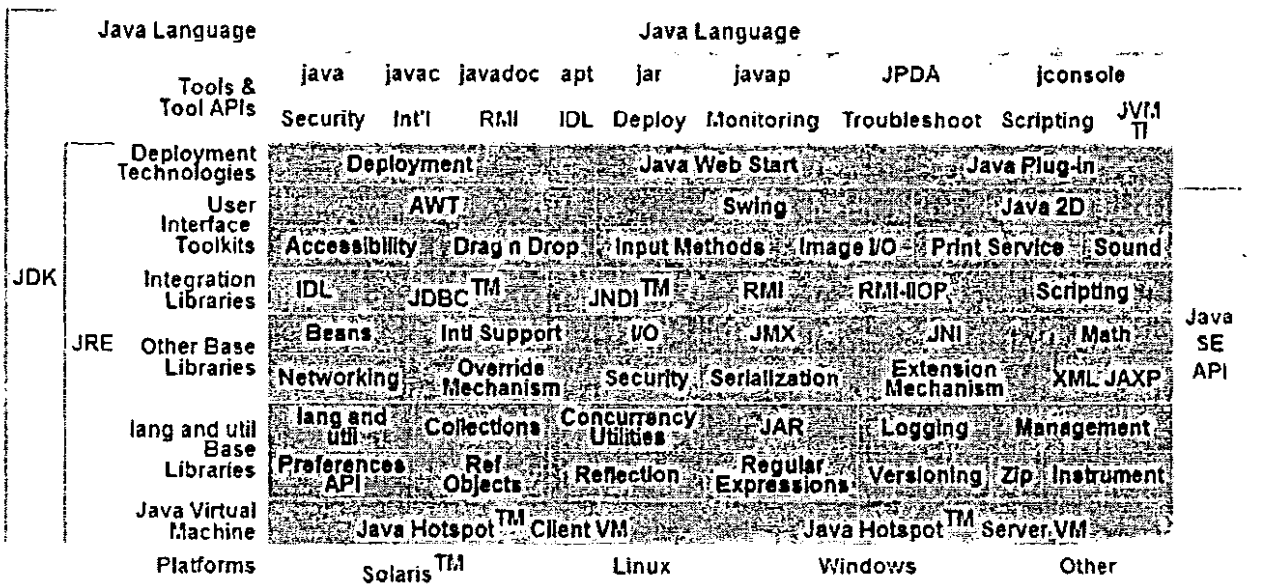


Fig 5.2 Java Platform Diagram from Sun

5.3 The C++ language

C++ ("C Plus Plus") is a general-purpose programming language. It is regarded as a *middle-level* language, as it comprises a combination of both high-level and low-level language features. It is a statically typed, free-form, multi-paradigm, compiled language where compilation creates machine code for a target machine hardware, supports

procedural programming, data abstraction, object-oriented programming, and generic programming.

The language was developed by Bjarne Stroustrup in 1979 at Bell Labs as an enhancement to the C programming language and originally named "*C with Classes*". It was renamed to *C++* in 1983. Enhancements started with the addition of classes, followed by, among other features, virtual functions, operator overloading, multiple inheritance, templates, and exception handling.

5.3.1 Conversion of Database file into ASCII file

The Database file (.db) is converted to ASCII file in the same folder where the ARtool application resides by typing the following command in the *MS-Dos Command Prompt:-*

java db2asc status.db status.asc

5.3.2 Node structure

Each node in SE tree represents a frequent itemset. The node structure is as follows:

```
class node{
public:
        node* down_ptr;
        node* right_ptr;
        int* tid_list;
        char* node_id;
        int count;
         node()
                {
                  down_ptr=0;
                  right_ptr=0
                } };
```

5.3.3 Conversion of ASCII file into Text format

Following is the C++ code of converting the ASCII file(.asc) or format of numbers into text format (.txt file) of 0's and 1's:

```
int main(void)
{
  clrscr();
 float sim_threshold=0.5;
  unsigned  int i=1; //EOF
  FILE *in;
  if ((in = fopen("C:\\TC\\arttxt.asc", "r"))== NULL)
  {
   fprintf(stderr, "Cannot open input file.\n");
   return 1;
  }
  FILE *out;
  if ((out = fopen("C:\\TC\\arttxt.txt", "w"))== NULL)
  {
   fprintf(stderr, "Cannot open input file.\n");
   return 1;
  }


  int tn=1;


  do
  {
```

```
    int fcont;

    fscanf(in,"%i",andfcont);

   for(;tn<fcont;tn++)

    {
         fprintf(out,"%i",0);


    }

   fprintf(out,"%i",1);
   tn=fcont+1;

   char c;
   fscanf(in,"%c",&c);
   if((c=='a')||(c=='b'))

    {
       i++;
       fprintf(out,"%c",'\n');
       tn=1;
    }

  }while (i!=9);//3197 11
  cout<<"number of Records="<<i<<endl;
  fclose(in);
  getch();
  return 0;
}
```

5.3.4 Reading of Text file

After the conversion of the .asc file into .txt file, following is the module that will read the text file of 0's and 1's showing the absence and presence of items. When the file is read, track of number of 1's is kept as the support count of a certain item in order to find out whether that item is frequent.

```
void database_scan(int record_length)
 {
    int current_ptr=0;
    int space=2;
    char node_no='a';
    int countt;
freecache();
start=createnode();
previous=start;

count=fill_buff(node_no,record_length,record_length);
start->count=countt;
cout<<"****COUNT****>>"<<start->count<<endl;

char character[2];
character[0]=node_no;
character[1]='\0';

copybuffer(start,size_filledbuff(),character);
node_no++;
delay(1000);
cout<<NODE NO"<<node_no<<endl;
```

```
for(int c=space;c<record_length-space;node_no++,c+=space)
  {
   current_ptr=c+record_length;
   current=createnode();
   countt= fill_buff(node_no,current_ptr, record_length);
   cout<<"$$$$$$$$$$COUNT$$$$$$$$"<<countt<<endl;
   current->count=countt;
   character[0]=node_no;
   character[1]='\0';
   copybuffer(current,size_filledbuff(),character);
   previous->right_ptr=current;
   previous=current;
  cout<<"NODE NO"<<node_no<<endl;
 }
}
```

5.3.5 Filling the buffer

The readfile_makelist() module makes use of the fill_buffer() function basically fills the buffer with the count of 1's i.e. number of 1's in an item, with help of a pointer named as current_ptr and the parameter record_length.

```
int fill_buff(int node_id,int current_ptr, int record_length)
{
   int count=0;
   freecache();
   int cachecounter=0;


for(int row=0;row<rows;current_ptr+=record_length,row++)
```

```
{

    fseek(infile, 0L, SEEK_SET);
    fseek(infile,current_ptr, SEEK_CUR);


    if(fgetc(infile)=='1')
     {

       count++;
       cache[cachecounter]=row;
       cachecounter++;
      delay(100);
      cout<<"Node Number"<<node_id<<endl<<"chread=="<<cachecounter<<endl;

  }
     current->count=count;
     cout<<"COUNTSS"<<current->count;

    //cout<<"The occurance of one is:"<<count<<endl;

}

return count;
}
```

5.3.6 Copying Buffer to node

Once a node is created, the relevant data or information of that node is copied from the cache/buffer to the newly created node's tid_list field. This information may include the count f 1's of that node. In other words, afer the buffer is filled it is then copied by a function named **copybuffer** in to newly created node tid_list (info) field.

This function takes the pointer to newly created node and size of the buffer and node_id to be copied into that node. The created node is then linked with the previous node and in this way the first level of set enumeration tree is created.

```
void copybuffer(node* temp,int sz, char* node_id)
{ temp->getsize(sz+1);
    int str_size=get_stringsize(node_id);
    temp->getsize_nodeid(str_size);
    strcpy(temp->node_id,node_id);
    for(int c=0;c<sz;c++)
      {
        temp->tid_list[c]=cache[c];
        temp->tid_list[sz]=-99
      }
       delay(50);
       cout<<"node start"<<endl;
       for(int t=0;t<sz;t++)
        {
          cout<<temp->tid_list[t]<<endl;
        }
          cout<<"node end"<<endl;
}
```

5.3.7 Finding the Second level

After first level of the SE tree is created, the second level is created by finding the similarity between every two items in the first level. If the similarity is equal to or greater than the user specified threshold, then that itemset will be added in the tree considering it as a frequent itemset.

```
int Generate_frequent_2_itemset()
{

   int nfis=0;
   node* outernode=start;
   node* innernode;
   node* temp;
   float similarity;

   while(outernode->right_ptr!=0)
   {
     node* strt=0;
     node* prs=0;
     int flag=1;
     innernode=outernode->right_ptr;
     char* newtag=new char[3];
      while(innernode > 0)
      {
        strcpy(newtag,outernode->node_id);
        strcat(newtag,innernode->node_id);
        freecache();
        intersectioninbuffer(outernode,innernode);
        int numerator=size_filledbuff();
        int product= (outernode->count-1)*(innernode->count-1);
        int denominator=sqrt(product);

       if(denominator==0)
        {
          goto nextnode;  }

similarity= (numerator*1.0) / denominator;
```

```
if(similarity >= sim_threshold)
  {
    if(flag==1)
    {
        cout<<"Similar tag="<<newtag<<"  "<<similarity<<endl;
        strt=createnode();
        prs = strt;
        copybuffer(strt,size_filledbuff(),newtag);
        freecache();//cache is freed after it is copied to node
        flag=0;
        nfis++;
    }
else{
        temp = createnode();
        cout<<"Similar NOde="<<newtag<<"  "<<similarity<<endl;
        copybuffer(temp,size_filledbuff(),newtag);
        freecache();
        prs->right_ptr = temp;
        prs = temp;
       nfis++;
    }
  }//if end

 nextnode :  innernode = innernode->right_ptr;
}//inner while loop

outernode->down_ptr=strt;
outernode=outernode->right_ptr;

}//outer loop
```

```
return nfis;
}
```

5.3.8 Finding the Next level

The next level or the third and other subsequent levels are found the same way as the second level is made. However, it uses the previous level to find similarity between every 2-itemsets and hence generate the next level.

```
int Generate_next_frequent_itemset (void){

  node* outernode;//=fnode;

 int ind=0;
// outernode=Generate_frequent_2_itemset();
  int nfis=0;

  while((outernode=nodeadd[ind])!=NULL)
{
 cout<<"outer node"<<outernode->node_id<<endl;
  ind++;
  node* innernode;
  node* temp;
  float similarity=0;
  delay(100);
  while(outernode->right_ptr!=0)
  {
    cout<<"entered nex level"<<endl;
    node* strt=0;
    node* prs=0;
    int flag=1;
    innernode=outernode->right_ptr;
```

```
    char* newtag=new char[10];
    while(innernode > 0)
     {

        strcpy(newtag,outernode->node_id);
        char* src=new char[10];
        strcpy(src,innernode->node_id);
        cout<<"src"<<src<<endl;
        char* str_bit=new char[10];
       SubString(src,str_bit,get_stringsize(innernode->node_id)
1,get_stringsize(innernode->node_id));


cout<<"SRC IN NEX LEBVEL-->>"<<src<<endl;
printf("STR_BITTT%s",str_bit);
cout<<"outernode...."<<newtag<<endl;
strcat(newtag,str_bit);
cout<<"NEW TAG INNEX LEVEL---->>>"<<newtag<<endl;
freecache();
intersectioninbuffer1(outernode,innernode,temp);
int numerator=size_filledbuff();
cout<<"intrsctn="<<intrsctn<<"tag="<<newtag<<endl;
int product= (outernode->count-1)*(innernode->count-1)*(temp->count-1);
int denominator = sqrt(product);
if(denominator==0)
{
   cout<<" node not found"<<endl;
 goto nextnode;
}
  similarity= (numerator*1.0) / denominator;
```

```
if(similarity >= sim_threshold)
 {
  if(flag==1)
   {
     strt=createnode();
     prs = strt;
     copybuffer(strt,size_filledbuff(),str_bit);
     cout<<"node tag="<<str_bit<<similarity<<endl;
     freecache();//cache is freed after it is copied to node
     flag=0;
     nfis++;
   }else
   {
    node* temp = createnode();
    copybuffer(temp,size_filledbuff(),str_bit);
    cout<<"node tag="<<str_bit<<similarity<<endl;
    freecache();
    prs->right_ptr = temp;
    prs = temp;
    nfis++;
   }
 }//if end
 nextnode :  innernode = innernode->right_ptr;
}//inner while loop

outernode->down_ptr=strt;
outernode=outernode->right_ptr;
}//outer loop
}
 return nfis;
```

}

5.3.9 Selecting the Previous FIS/candidate

This module will select the candidates for the third level, sot that it is easy to generate the 3-itemset, i.e. it will use 'ab' and 'ac' nodes in the second level to create the first node in the third level i.e. 'abc' after combination of these two nodes. Basically, it will return the starting pointer of the linked list i.e. the address of the first node in the linked list of the respective node. For example, under the node 'a', node 'ab' and node 'ac' if frequent are the children of the node 'a'. Thus they form a linked list and this module will return the address of the first node in the linked list i.e. 'ab' and pass this address to the Generate_next_frequent_itemset() module which will then concatenate the nodes 'ab' and 'ac' to generate next level node 'abc'.

```
void select_previous_FIS(){
stack s;
//node* start_node;
clearnodeadd();
int ind=0;
node* down_link;
node* current_node = start;

if( current_node->down_ptr == 0)
current_node = current_node->right_ptr;
else {
        s.push(current_node->right_ptr);
        current_node = current_node->down_ptr;
       down_link= current_node;


   }
```

```
while ((s.stackisnotempty()==1) || (current_node->right_ptr !=0) || (current_node->down_ptr !=0))
{

if( current_node->down_ptr!=0)
 {
    s.push(current_node->right_ptr);
   down_link = current_node->down_ptr;
   current_node = current_node->down_ptr;
}else
{
   if( current_node->right_ptr != 0 )
   current_node = current_node->right_ptr;
   else{
         if(( down_link==0) &&(s.stackisnotempty()==1))
          {
            while((current_node=s.pop())&&(current_node->down_ptr==0)&&
(current_node->right_ptr==0))
              {
                if(s.stackisnotempty()==0)
                return;
              }//while end
            }else
                 return;
 }
}

if(( down_link != 0)&&( current_node->down_ptr==0)&&( current_node->right_ptr==0))
{
```

```
    if(down_link!=current_node)
     {
        nodeadd[ind]=down_link;
        ind++;
     }

  if(s.stackisnotempty()==1)
  {
      down_link=0;

while((current_node=s.pop())&&(current_node->down_ptr==0)&&(current_node-
>right_ptr==0))
     {
        if(s.stackisnotempty()==0)
         return;//then exit from algorithm
      }//while end
 }else
  return;

}//if end

}
```

Chapter 6

****

RESULTS

6. Results

Following is the example of the sample database on which the ***Cos_FIS generator*** algorithm was applied. This sample database is generated by using the ARtool.

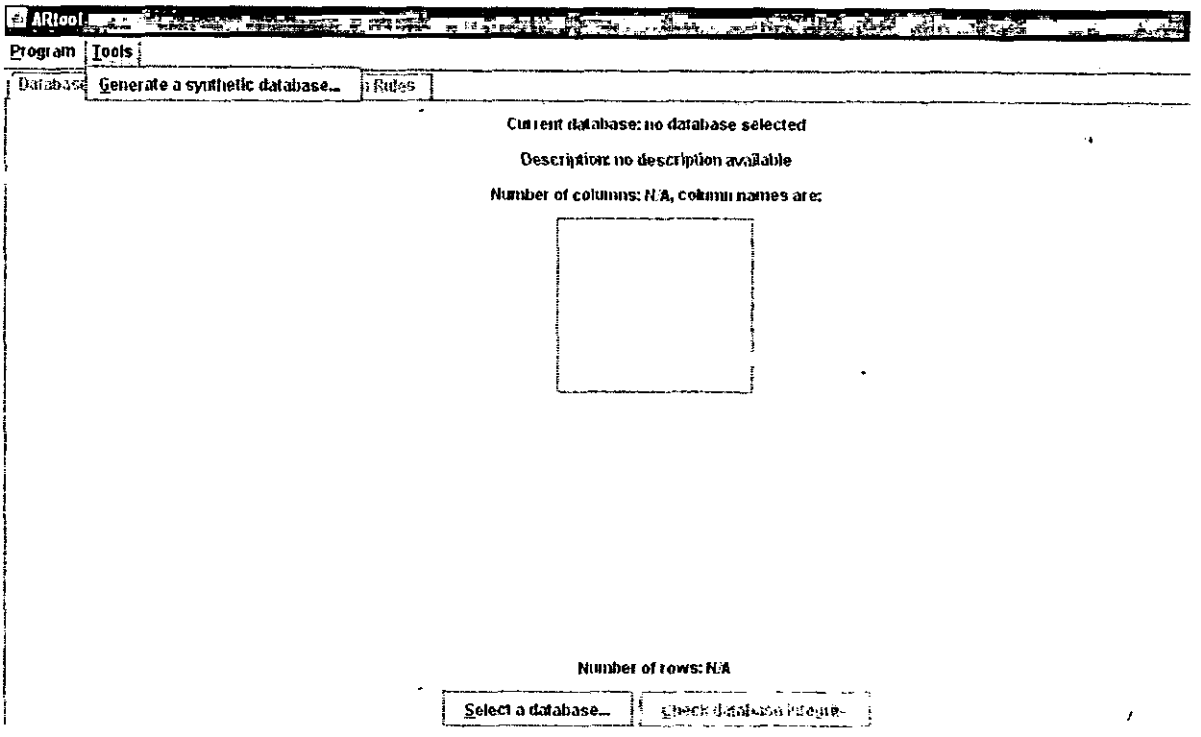


Fig 6.1 ARtool Graphical User Interface

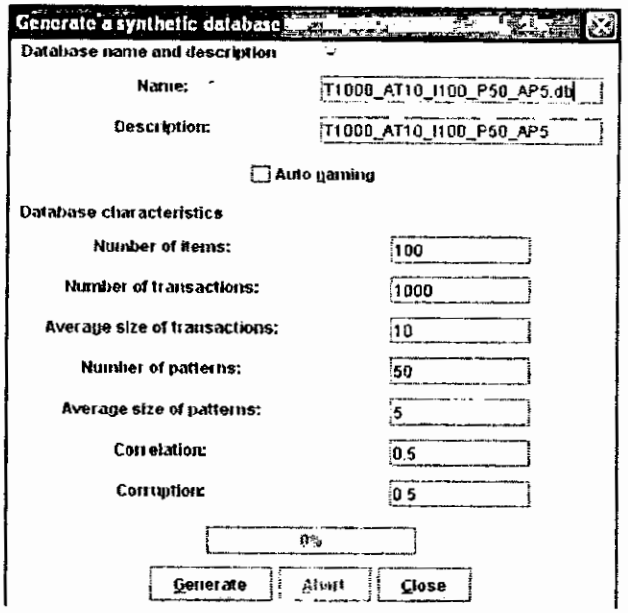


Fig 6.2 Generating a synthetic database

By using the tools menu in the ARtool application's user interface, user can generate a synthetic database i.e. T1000_AT10_I100_P50_AP5.db in our case.

1 A

2 B

3 C

4 D

5 E

6 F

BEGIN_DATA

1 3 5

3 4 6

1 2 3

END_DATA

Fig 6.4 Newdatabase.db

6.1 Conversion of Database file into ASCII file

The 'Newdatabase.db' file is converted to ASCII file 'Newdatabase.asc' in the same folder where the ARtool application resides by typing the following command in the *MS-Dos Command Prompt:-*

java db2asc Newdatabase.db Newdatabase.asc

6.2 Conversion of ASCII file into Text file

The ASCII file **'Newdatabase.asc'** is converted to text file **'Newdatabase.txt'** by using the C++ code. This file is in the form of 0's and 1's which shows the absence or presence of a certain item in columns where each row represents a specific transaction respectively.

1 3 4 5 6

3 4 5 6

1 2 3 4 5

Fig 6.5 Newdatabase.asc file

1 0 1 1 1 1

0 0 1 1 1 1

1 1 1 1 1 0

Fig 6.6 Newdatabase.txt file

6.3 Working of the Cos_FIS generator algorithm

The Cos_FIS generator algorithm begins by scanning the text format of the database in the form of vertical layout and keeping track of the support count of all the items in the cache i.e. the total number of transactions containing the item.

Transaction	A	B	C	D	E	F
1	1	0	1	1	1	1
2	0	0	1	1	1	1
3	1	1	1	1	1	0

Table 6-1 Vertical data Layout of the Synthetic Binary Database

6.3.1 Finding Support count

While the database scan is being performed, the ***database_scan()*** module keeps track of the support count of each item through the ***fill_buff()*** function

inside it. The support count refers to the number of 1's that come in the column of an item or the presence of the item in a certain transaction i.e. the number of transactions containing a specific item. The support count is saved in cache or buffer in our code. For example:- The support count of item 'A' is 2 .

6.3.2 Finding the First level of the SE tree

After the database in text format is scanned, the first level of the Set Enumeration tree is created through the module database_scan(). In this module first a node is created and then the column of data set is scanned according to the pointer ***Current_ptr*** with the help of a function named **fill_buff**, which requires *record_length* parameter, the ***fill_buff*** fills the buffer. This buffer is then copied by a function named ***copy_buffer*** in to newly created node info field. This function takes the pointer to newly created node and size of the buffer and node tag to be copied into that node. The created node is then linked with the previous node and in this way the first level of set enumeration tree is created.

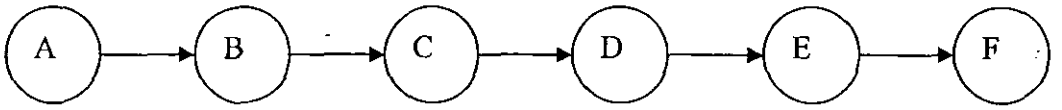


Fig 6.7 First level of SE tree

However, the output in C++ is as follows where each node or item has the same support in this case:-

Node no → a

$$$$$$$$$$COUNT$$$$$$$$ 2

node start

1

3

node end

NODE no = b

$$$$$$$$$$COUNT$$$$$$$$ 1

node start

3

node end

NODE no = c

$$$$$$$$$$COUNT$$$$$$$$ 3

node start

1

2

3

node end

NODE no = d

$$$$$$$$$$$COUNT$$$$$$$$$ 3

node start

1

2

3

node end

NODE no = e

$$$$$$$$$$$COUNT$$$$$$$$$ 3

node start

1

2

3

node end

NODE no = f

$$$$$$$$$$$COUNT$$$$$$$$$ 2

node start

1

2

node end

Fig 6.8 First level of SE tree in C++ output

6.3.3 Finding the Second level of the SE tree

The second level of the SE tree is created using its first level, by finding Cosine similarity between every two items in the first level. Each node in the first level has a subtree beneath it which is stored as a linked list i.e. if Node 'a' has node 'ab' and node 'ac' beneath it, then this forms a sub-tree as well as the linked list where the starting node is 'ab'. The support count found earlier in the first level formation is used in the Cosine measure formula to find the similarity. The Cosine formula is as follows:-

$$IS_{ABsim}=P(AUB) / \sqrt{P(A)X\sqrt{P(B)}}$$

Where P(A), the probability of the item 'A' is found by dividing the support count by the total number of transactions. i.e. 2/6 =0.3.

After putting all the relevant values into the *Cosine Similarity measure* formula, if the similarity found is greater than or equal to the user specified similarity threshold, then the itemset is said to be frequent and added in the SE tree as nodes of the second level.

Following is an example of itemsets in second level where the list of numbers shows the transactions containing the respective itemset:-

Starting Similar node= de >>> Similarity=1

node start

1

2

3

node end

Subsequent Similar node= df >>> Similarity= 1

node start

1

2

node end

Fig 6.9 Second level of SE tree in C++ output

6.3.4 Finding the Next level of the SE tree

After creating the second level of the SE tree, the subsequent levels are created using the previous level. For example, the third level is created using the itemsets of the second level. Each node in the first level has a subtree beneath it which is stored as a linked list i.e. if Node 'a' has node 'ab' and node 'ac' beneath it, then this forms a sub-tree as well as the linked list where the starting node is 'ab'. The ***select_previous_FIS()*** module will select candidates for the third level, by returning the address of the starting node in every linked list and passing this address to the

Generate_next_frequent_itemset() module. This module will then use the starting address to concatenate all the nodes in the respective linked list and hence construct the node(s) for the next level.

Following is an example of the third level nodes or itemsets where the nodes in the second level are concatenated i.e. 'cd' and 'ce' are combined to form 'cde':-

entered next level

Src ce

SUBSTRING;jSRC IN NEX LEVEL-->> ce

*****outernode*****cd

NEW TAG INNER LEVEL---->>> **cde**

Src cf

SUBSTRING; k SRC IN NEX LEVEL-->>cf

*****outernode*****cd

NEW TAG INNER LEVEL---->>>**cdf**

Src gi

SUBSTRING; k SRC IN NEX LEVEL-->>gi

*****outernode*****gh

NEW TAG INNER LEVEL---->>>**ghi**

Src gj

SUBSTRING; k SRC IN NEX LEVEL-->>gj

*****outernode*****gh

NEW TAG INNER LEVEL---->>>**ghj**

entered next level

src gk

SUBSTRING;jSRC IN NEX LEVEL-->>gk

*****outernode*****gh

NEW TAG INNER LEVEL---->>>**ghk**

Fig 6.10 Third level of SE tree in C++ output

Chapter 7

****

CONCLUSION & FUTURE ENHANCEMENT

7. Conclusion and Future Enhancement

In this section, the conclusions and future enhancements to the Cos_FIS generator algorithm will be discussed. This algorithm was implemented on the Pentium machine with the windows Xp version. However, further enhancements to the algorithm can be performed to increase efficiency of the software in order to cope with the slow processing and limited speed of the system.

7.1 Conclusion

All the previous techniques and algorithms of generating frequent itemsets use support and confidence measures as well as have some drawbacks such as multiple number of scans and large candidate itemsets. However, the Cos_FIS generator algorithm uses a new clustering measure 'Cosine measure' to generate frequent itemsets which has certain advantages i.e. there is no candidate generation using this measure. Also, this algorithm uses the vertical data format or vertical data layout for scanning the database which gives the benefit of a single scan of the database. This also establishes that the clustering measures can also be used for association rule mining. Furthermore, the same FIS that the user gets by applying Cosine similarity measure on transactional dataset, can be obtained by using the Apriori algorithm.

7.2 Future Enhancements

In this thesis, the application of the Cosine measure was studied as well as this measure was implemented for generation of frequent itemsets. The emphasis in this thesis was to observe that apart from support measure, other measures do exist and these can be used to generate FIS. In future, different clustering measures can be compared to decide which one is the best candidate for FIS generation in terms of accuracy, time and memory consumption. Moreover, the Set Enumeration tree and vertical database layout to arrange frequent itemsets were implemented. Therefore, the implementation of the algorithm in this thesis also shows that clustering measures can also be used for the creation of frequent itemsets. Also, the SE tree in this thesis could be represented graphically. In

future the Cos_FIS generator algorithm will be compared with the established algorithms of association rule mining for efficiency purpose.

References

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," ACM SIGMOD Conf. Management of Data, May 1993

[2] A. Savasere, E. Omiecinski, and S. Navathe, ™An Efficient Algorithm for Mining Association Rules in Large Databases,° Proc. 21st Very Large Data Bases Conf., 1995.

[3] Toivonen, H.1996, sampling large databases for association rules. In Proc.22nd VLDB Confrence, Bombay, pp. 134-145

[4] Bayadro, R,j. 1998 Efficiently mining long patterns from databases. In pro. ACM-SIGMOD Int Conf on management of data,pp.85-93

[5] M.J. Zaki, S Parthasarathy, M.Ogihara, "New Algorithms for fast discovery of Associations Rules", Third Int'l Conf. Knowledge Discovery and Data mining, Aug .1997

[6] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data* Mining and Knowledge Discovery, 2004.

[7] Bart Goethals SAC', March 14-17, 2004, Nicosia, Cyprus "Memory issues in Frequent Itemset mining".

[8] P. Cosine. Nouvelles recherches sur la distribution florale. Bulletin de la Societe Vaudoise de Sciences Naturelles, 44:223–270, 1908.

[9] Frans coenen, Graham GoulBourne, Paul Leng "Tree Structure for Mining Association rules" July 17 2002.

[10] Rymon, R.1992, search through systematic set enumeration. In Proc 3rd Int'l Conf.on principles of knowledge Representation and reasoning, pp 539-550

[11] M.S.H Khiyal, S Rahman, A Salam, D Khan. "Similarity based mining for finding frequent itemsets". International Conference on Computers, Communications and Systems.(ICCCS 2 Nov .2007) south Korea daegu university.

[12] Pang-Ning Tan, Vipin Kumar, Jaideep Srivastava "Selecting the right interestingness measure for association patterns" 2002.

ARtool v1.1.2 - Association Rule Mining Algorithms and Tools

B-1 Introduction

ARtool is a Java application for mining frequent itemsets and association rules in binary databases. ARtool is free software distributed under the GNU General Public License. ARtool uses a custom format for its database files (which will be henceforth referred to as the .db format and is identical to the format used in ARMiner). The asc2db and db2asc are utilities that allow the conversion of a .db file to a specially formatted ASCII file (we will refer to this as .asc) and respectively the conversion of a .asc file into a .db file. The .asc files can be easily read and modified with any decent ASCII editor.

B-2 Description and usage of the .asc format

A small example of supermarket data can be taken. Suppose the items sold by a (very, very small) shop are green apples, red apples,oranges, bananas, and grapes. Also suppose that in this morning user had three customers, one bought green apples and grapes, one bought only oranges, and the last one bought oranges and grapes. This activity can be represented in the .asc format as follows:

```
1 green apples
2 red apples
3 oranges
4 bananas
5 grapes
BEGIN_DATA
1 5
3
3 5
END_DATA
```

There are two distinct parts of this file, the first one contains a listing of all the items user can sell, or otherwise said, of all the items that could participate in a transaction. This part looks is:

```
1 green apples
2 red apples
3 oranges
4 bananas
5 grapes
```

The format is pretty simple. It must consist of a positive number followed by a string (which can contain blank spaces). It is important that the numbers be assigned in increasing order starting from1. Empty lines are allowed to appear in this section. This section enumerates all entities described by the data and between which ARtool will later be used to look for association rules.

The second part consists of the actual data:

```
BEGIN_DATA
1 5
3
3 5
END_DATA
```

In our case there were 3 transactions and these are each represented on a separate line. The first transaction involved green apples and grapes and they are represented by the numbers associated in the first section, that is 1 for green apples and 5 for grapes. The user can check the other transactions as an exercise. Note that this section must be enclosed between a BEGIN_DATA and END_DATA lines. Anything appearing after the END_DATA line will be ignored. Blank lines are allowed to appear in this section. Note that although the numbers appearing in each line are sorted, this is not required by the format. The user can list the numbers in any order and the file can still be processed correctly, however it is suggested to always list the numbers in a transaction in increasing order, because in this way asc2db will process the file more efficiently.

This concludes the supermarket data example as well as the description of the .asc format. However most of the time the data will not be similar to the one used in this example. If that happens, then the user will have to try to figure out some way in which he/she can express their data in the .asc format. To give an idea, following is another example:

Suppose the user has some sort of census data like the one below:

SSN#	Age	Sex	Married	Num_kids	Income
006	26	M	No	0	25000$
345	54	F	Yes	2	55000$
743	37	M	Yes	1	80000$

What can be done with it? Let's look at each column:

SSN#: this is unique for each entry, there is no sense to look for association rules involving SSN#, at least not in this data, since each SSN# appears only once in the whole data. So we can simply ignore this field for mining purposes.

Age: this attribute can take a variety of values. ARtool cannot handle such attributes easily, in fact it only considers binary attributes. The user needs to discretize this

attribute, replacing for example ages 0-21 with "very young age", 22-35 with "young age", 35-55 with "middle age", etc

Sex: this has two values: "male" and "female", so user could create two attributes out of it.

Married: again we can create two attributes: "married" and "not married"

Num_kids: this also has to be discretized, maybe in "no kids", "one kid", "several kids".

Income: we could also discretize this into "small", "average", and "high".

The discretization should be made such that it will identify clearly the ranges that present interest for the person who will do the mining of this data.

With these changes we could represent the above data in .asc format as:

```
1 very young age
2 young age
3 middle age
4 old age
5 male
6 female
7 married
8 not married
9 no kids
10 one kid
11 several kids
12 small income
13 average income
14 high income
BEGIN_DATA
2 5 8 9 12
3 6 7 11 13
3 5 7 10 14
END_DATA
```

From this file the user can now create a .db file and then mine it using ARtool or ARMiner.

B-2-1 Using asc2db

The asc2db program can be used to convert a correctly formatted .asc file to ARtool's .db format. Suppose user has a sample .asc file. Then he/she can create a .db file from it by typing:

java asc2db sample

which will create a sample .db file. If the user wants the .db file to have a different name then he/she can specify it on the command line as a second parameter:

java asc2db sample artdata

which will now produce an artdata.db file out of the sample.asc input. Note that the extensions .asc and .db do not have to be specified on the command line, they are automatically appended by asc2db.

B-2-2 Using db2asc

The db2asc program converts a .db file to .asc format. This can be useful if the user wants to read or verify the content of a .db file. The user can also use it to modify by hand the contents of a .db file by first converting it to a .asc file, then editing the .asc file, and finally converting it back to a .db file. ***db2asc*** is used in a similar way to its counterpart, ***asc2db***. If the user needs to convert the armdata.db database to .asc format, then he/she can type:

java db2asc artdata

which will produce an armdata.asc file. If the user wants a different name for the output, then you can pass it on the command line as a second argument:

java db2asc artdata arttxt

which will produce an arttxt.asc file representing the contents of the artdata.db database.

Again, the extensions .asc and .db should not be entered on the command line, since they are automatically appended by db2asc.

B-3 How to install and execute ARtool

The user needs to have Sun's JDK or JRE installed, probably at least version 1.3.

In order to install ARtool, the user needs to unzip the ARtool binaries to some directory on his/her hard drive.

To run the ARtool GUI, the user needs to type:

java -jar ARtool.jar

or double-click on ARtool.jar (works only if JRE is installed).

If ARtool runs out of memory during some mining operation (the user can see an OutOfMemoryException message), then he/she needs to allocate more memory to the JVM. In the case of Sun's JDK he/she can do this by typing:

```
java -Xmx512M -jar ARtool.jar
```

which will let JVM use 512MB of memory, assuming of course that the user has that much memory installed.

If the user wants to use the command line utilities, then he/she will have to add laur.zip to their class path. On Windows the user needs to have in his/her autoexec.bat a line like this:

```
SET CLASSPATH=.;C:\ARTOOL\BIN\LAUR.ZIP
```

If the user uses Unix, then he/she will have to add something like

```
setenv CLASSPATH .:~/ARtool/bin/laur.zip
```

to their shell configuration file.

Frequent Itemset Generation Using Cosine Measure

Sobia Malik
Department of computer science
International Islamic University (IIU)
Islamabad, Pakistan.
smz_techno@yahoo.com

ABSTRACT

Generating frequent itemsets (FIS) is the first step of association rule mining. Existing techniques/algorithms for generating FIS used the well known support and confidence measures. However, we introduced a novel algorithm which makes use of a clustering measure i.e. Cosine measure for the generation of FIS. This algorithm presents the FIS in the form of a Set Enumeration tree in addition to the use of vertical database layout for clustering the items together. Furthermore, the results show that the same FIS that the user gets by applying Cosine similarity measure on transactional dataset, can be obtained by using the Apriori algorithm.

Keywords: Association rule mining, Frequent Itemsets, Cosine similarity measure.

1. INTRODUCTION

Data Mining is the process of running data through sophisticated algorithms to uncover meaningful patterns and correlations that may otherwise be hidden. These can be used to help user understand the business better and also exploit to improve future performance through predictive analytics. In data mining, association rule mining plays a vital role which discovers interesting relations between variables in large databases. Piatetsky-shapiro describes analyzing and presenting strong rules discovered in databases using different measures of interestingness. For example, the information that customers who buy burgers also tend to buy coke at the same time is represented in association Rule below:

{Burger}--> {Coke}

Where burger is the antecedent and coke is the consequent of the rule. An association rule has two numbers that express the degree of uncertainty about the rule namely ***Support*** and ***Confidence***. Researchers have discovered numerous techniques to find FIS, mostly based on these measures. However, we introduce a new clustering measure for the same purpose known as ***"Cosine"*** measure. While clustering, data points are arranged in a way that the points nearest to each other are placed in one cluster. This can be done by similarity or dissimilarity measures. Similar data items will be nearest to each other and dissimilar will be at distance far apart. Cosine similarity measure is one of the clustering measures. The purpose of clustering measure is to join together objects into successively larger clusters, using some measure of similarity or distance. A typical result of this type of clustering is the hierarchical tree. Cosine similarity is a measure of similarity between two vectors of n

dimensions by finding the cosine of the angle between them, often used to compare documents in text mining. The cosine similarity of two vectors (d1 and d2) is defined as:

$$\mathbf{Cos(d1, d2) = dot(d1,d2) / ||d1||.||d2||}$$

Where *dot(d1, d2) = d1[0]*d2[0]+d1[1]*d2[1]...*

And where *||d1|| = sqrt (d1[0]^2+d1[1]^2...).*

2. LITERATURE REVIEW

Apriori is a seminal algorithm proposed by R.Agrawal[1] in May 1993. It uses *prior knowledge* of frequent itemset properties. Apriori uses breadth-first search and a hash tree structure to count candidate item sets efficiently. **FP-growth algorithm[6]** proposed by J. Han, J. Pei, Y. Yin, and R. Mao in 2004 adopts a *divide and conquer* strategy avoiding costly candidate generation. FP-growth tree is memory resident and requires additional storage in every node of the FP-tree (Because of excessive pointers storage in every node) especially when the FP- tree is too large to fit in main memory. **Partition algorithm** was proposed by **A. Savasere[2]** in 1996. This algorithm is used for partitioning the data to find candidate itemsets. A partitioning technique can be used that requires just two database scans to mine the frequent itemsets. The problem of accurately estimating the number of partitions given the available memory, however, needs further work. **Sampling** approach was proposed by **Toivonen[3]** in 1996. This algorithm is used for mining on a subset of the given data. The basic idea of the sampling approach is to pick a random sample S of the given data D, and then search for frequent itemsets in S instead of D. In this way, there is some tradeoff of accuracy against efficiency. **MaxMiner (Bayardo, 1998)[4]** is another algorithm for finding the maximal elements. It uses Rymon R(1992)[10] "search through systematic set Enumeration" mechanism and efficient pruning techniques to quickly narrow the search. **ECLAT (Equivalence CLASS transformation)[5]** is an algorithm developed by **M.J Zaki**, which transforms a given data set of transactions in the horizontal data format of *TID-itemset* into the vertical format of *item-TID-set*. The above literature shows that association rule mining is facing a number of problems currently such as multiple scans of database and generation of large candidate itemsets which needs to be solved.These problems cann be solved by using clustering measures such as Jacquard and Cosine measure etc. **SB-Miner[11]** developed by **S.Rahman** is novel algorithm to find FIS based on clustering measure i.e. jacquard similarity measure. Jacquard similarity measure is based on calculating the distance between itemsets.

3. THE COS_FIS GENERATOR ALGORITHM

The Cos_FIS generator algorithm uses the clustering measure i.e. Cosine measure to generate frequent itemsets. Therefore, this algorithm makes use of the SE tree which arranges the k-itemsets according to their specific levels.

3.1 MAJOR MODULES

Division of any project into modules adds to its efficiency and overall performance. Hence this project is divided into following modules:-

1. Database conversion.
2. File reading.
3. Copying cache to node.
4. Frequent itemset generation.

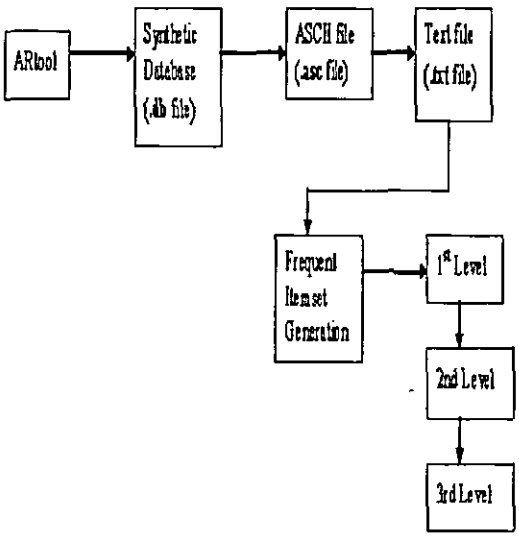


Fig 1 Architectural Diagram

3.2 THE SET ENUMERATION TREE

The Set Enumeration tree is a lattice structure used to enumerate all possible itemsets. In general, a data set that contains *k* items can potentially generate up to 2^K-1 frequent itemsets, excluding the null set. Because *k* can be very large in many practical applications, the search space of itemsets that need to be explored is exponentially large. In this paper, the SE tree as illustrated in Fig 1 is the data structure used to represent the frequent itemsets in a lexicographic order. Each node represents a frequent itemset and each level represents k-itemset i.e. level one will have 1-itemset and level two will have 2-itemset and so on. Every subtree as shown in Fig 1 represents an equivalence class of its root node.

Definition 1 **[Equivalence class]**

Equivalence class of node A consists of all elements containing node A. For example, following is the equivalence class of node A:-

A = {AB, AC, AD}

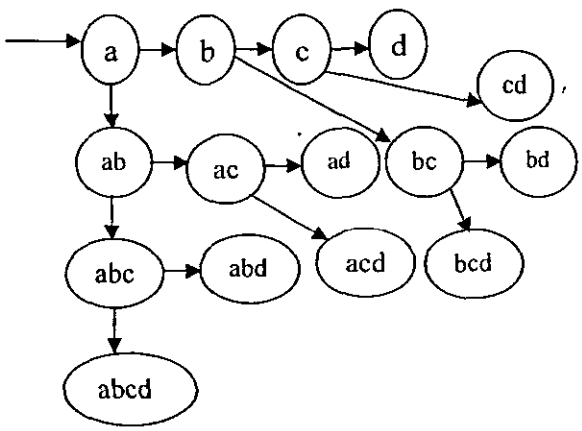


Fig 2 Set Enumeration Tree

3.3 THE NODE STRUCTURE OF SE TREE

Each node in SE tree represents a frequent itemset. Each node has a unique **node_id** field, which shows the name of the node. Node has a **count** field indicating the total number of transactions containing that frequent itemset. There are two node pointers down and right node pointer pointing to the node that is linked to current node in downward position level-wise and to the right position item-wise respectively. In other words, the down pointer points to the node in the next level where each level represents the k-itemset and the right pointer points to the node of the same level. Moreover, there is a **transaction_id_list** field, which actually represents the transaction IDs of transactions containing the particular FIS represented by the **node_id** of the node.

3.4 CREATION OF THE SE TREE

A brute force approach for finding frequent itemsets is to determine the support count of every candidate itemset in the lattice structure. To do this, we need to compare each candidate against every transaction. If the candidate is contained in the transaction, its support count will be incremented. The algorithm shown in Fig 2 creates the SE tree for the purpose of generation of frequent itemsets. The Set Enumeration tree(SE tree) is created in a very systematic way. It begins by execution of the first module of the code i.e. **database_scan**. In this module, first node '**a**' is created and then the column of dataset (text file) is scanned according to a pointer, hence filling the buffer with the support count. If the support count of the node is greater than or equal to the minimum user specified threshold, then the node is said to be frequent and hence is added in the SE tree. After the buffer is filled with count of the node using **fill_buff** function, it is then copied into the info field of newly created node using **copybuffer** module. It is then linked with the previous node. This way the first level of SE tree is generated. The second level of SE tree is obtained by execution of the **Generate_frequent_2_itemset** module. This level is created by finding Cosine similarity between every two itemsets or nodes present in the first level of SE tree i.e) between node 'a' and node 'b'. Only those itemsets whose similarity is greater than or equal to the user supplied minimum similarity threshold are declared frequent and are linked in the second level of the SE tree in lexicographic order. And the same process is repeated for levels 3 and 4:
Each equivalence class is represented in form of a linked list.The **select_previous_FIS** module selects the FIS generated in the previous level and returns address of the starting node in the linked list of equivalence classes. Furthermore, the **Generate_next_frequent_itemset** module uses these addresses to generate the next subsequent levels.

EXAMPLE:-

Is {A,B,C} itemset frequent?

This question can be answered by applying the Cosine similarity measure on the itemset as follows:-

A	B	C
1	1	0
0	1	0
0	1	1
1	1	0
1	0	1
0	1	1
1	0	1
1	1	1
1	1	1

Minimum threshold = 0.2
AUBUC = A.B.C

A.B.C = 1*1*0+0*1*0+0*1*1+1*1*0+1*0*1+0*1*1+1*0*1+1*1*1+1*1*1
= 1+1= 2.

P(A.B.C) = 2/9
P(A) = 6/9
P(B) = 7/9
P(C) = 6/9

$IS_{ABsim}=P(AUBUC)/\sqrt{P(A)X\sqrt{P(B)}}$
$X\sqrt{P(C)}$

IS_{ABsim} = 2/9 / √6/9.√7/9.√6/9
= 0.22 / 0.59
= 0.3

Since 0.3 > 0.2, therefore the *{A, B, C}* itemset is frequent.

Algorithm **Cos_FIS_generator**

Input

Ds: Transactional Dataset
α : Threshold

Output:

FIS: Frequent itemsets
β : No. of FIS after each iteration

Step 1 : Scan transactional dataset DS.
Step 2: Construct first level of Prefix tree.
Step 3 :Construct second level of prefix tree by finding similarity between every 2 itemsets in the previous level of tree as shown in step 4.

Step 4: β = Generate_frequent_2_itemset(FIS, α) by using the ***Cosine similarity measure***

Repeat Step 5 to 7 until *β = 0.*

Step 5: Scan each sub tree in the last level of prefix tree.
Step 6: Store starting node's address of each sub tree in S.
Step 7: β = Generate_next_frequent_itemset (S,FIS, α).
Step 8: Return FIS.

Fig 3 Pseudo code of the COS_FIS generator algorithm

of scans and large candidate itemsets. However, the Cos_FIS generator algorithm uses a new clustering measure 'Cosine measure' to generate frequent itemsets which has certain advantages i.e. there is no candidate generation using this measure. This also establishes that the clustering measures can also be used for association rule mining. In future, different clustering measures can be compared to decide which one is the best candidate for FIS generation in terms of accuracy, time and memory consumption. Also, the SE tree in this thesis could be represented graphically. In future the Cos_FIS generator algorithm will be compared with the established algorithms of association rule mining for efficiency purpose.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," ACM SIGMOD Conf. Management of Data, May 1993

[2] A. Savasere, E. Omiecinski, and S. Navathe, ™An Efficient Algorithm for Mining Association Rules in Large Databases," Proc. 21st Very Large Data Bases Conf., 1995.

[3] Toivonen, H.1996, sampling large databases for association rules. In Proc.22nd VLDB Confrence, Bombay, pp. 134-145

[4] Bayadro, R,j. 1998 Efficiently mining long patterns from databases. In pro. ACM-SIGMOD Int Conf on management of data,pp.85-93

[5] M.J. Zaki, S Parthasarathy, M.Ogihara, "New Algorithms for fast discovery of Associations Rules", Third Int'l Conf. Knowledge Discovery and Data mining, Aug .1997

[6] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data* Mining and Knowledge Discovery, 2004.

[7] Bart Goethals SAC', March 14-17, 2004, Nicosia, Cyprus "Memory issues in Frequent Itemset mining".

[8] P. Cosine. Nouvelles recherches sur la distribution florale. Bulletin de la Societe Vaûdoise de Sciences Naturelles, 44:223–270, 1908.

[9] Frans coenen, Graham GoulBourne, Paul Leng "Tree Structure for Mining Association rules" July 17 2002.

[10] Rymon, R.1992, search through systematic set enumeration. In Proc 3rd Int'l Conf.on principles of knowledge Representation and reasoning, pp 539-550

[11] M.S.H Khiyal, S Rahman, A Salam, D Khan. "Similarity based mining for finding frequent itemsets". International Conference on Computers, Communications and Systems.(ICCCS 2 Nov .2007) south Korea daegu university.

[12] Pang-Ning Tan, Vipin Kumar, Jaideep Srivastava "Selecting the right interestingness measure for association patterns" 2002.