Frequent Pattern Mining using CATSIM Tree

Ketan Modi * and B.L.Pal
*: M.Tech Student,
Mewar University, Chittorgarh
Email: modiketanit@gmail.com
**: Assistant Professor, Mewar University, Chittorgarh,
Email: contact2bl@rediffmail.com

Abstract: Efficient algorithms to discover frequent patterns are essential in data mining research. Frequent pattern mining is emerging as powerful tool for many business applications such as e-commerce, recommender systems and supply chain management and group decision support systems to name a few. Several effective data structures, such as two-dimensional arrays, graphs, trees and tries have been proposed to collect candidate and frequent itemsets. It seems as the tree structure is most extractive to storing itemsets. The outstanding tree has been proposed so far is called FP-tree which is a prefix tree structure. Some advancement with the FP tree structure is proposed as CATS tree. CATS Tree extends the idea of FP-Tree to improve storage compression and allow frequent pattern mining without generation of candidate itemsets. It allows to mine only through a single pass over the database. The efficiency of Apriori, FP-Growth, CATS Tree for incremental mining is very poor. In all of the above mentioned algorithms, it is required to generate tree repeatedly to support incremental mining. The implemented CATSIM Tree uses more memory compared to Apriori, FP-Growth and CATS Tree, but with advancement in technology, is not a major concern. In this work CATSIM Tree with modifications in CATS Tree is implemented to support incremental mining with better results.

I. INTRODUCTION

Data mining has attracted a great compact of interest in the information industry in recent years is because of wide availability of large amounts of data and the imminent need for turning such data into useful information and knowledge. The information and knowledge gained can be used for applications ranging from business management, production control, and market analysis, to engineering design and science exploration. Data Mining refers to extracting or mining knowledge from large

amounts of data. Data mining is also known as Knowledge discovery in database [8] [4].

Data mining involves an integration of techniques from multiple disciplines such as database technology, statistics, machine learning, high-performance computing, pattern recognition, neural networks, information retrieval, image and signal processing [16], and spatial data analysis. By performing data mining, interesting knowledge, regularities, or high-level information can be extracted from databases and viewed from different angles. The discovered knowledge can be applied to decision making, process control, information management and query processing [17]. Therefore, data mining is considered one of the most promising interdisciplinary developments in the information industry [9] [12].

A procedure so called mining frequent itemsets is a foundation step of association rules discovering. By using basket data in supermarket, we can gain valuable information regarding to a relationship of items or products buying together in the form of association rules. Such information can be used to make dictions, for example designing weekly catalog, customer segmentation, cross-marketing etc. Frequent pattern mining for large databases of business data, such as transactional records, is of great interest. A number of efficient algorithms have been proposed in this area [1] [8] .

Apriori is the basic data-mining algorithm as discussed in [2] [8] [11] [12]. The second class of algorithms is based on frequent pattern growth (FP-growth), which generates frequent patterns without candidate generation [1] [3] [9] [19] [20]. The extension to the FP-growth algorithm is known as CATS (Compressed Arranged Transaction Sequences) Tree [5] [9]. When the data is dense where patterns within the dataset have high correlation with one another, e.g., medical data, CATS Tree allows data to be sorted with smaller space. However, most of successful algorithms in this area have been bounded by their limitations of making the frequent pattern mining an offline analytical task [1] [9] [14].
The contribution of this work is the development of a simple, but yet powerful and efficient, novel tree structure for maintaining frequent patterns found in the incremental database. The CATS tree is for interactive mining, not for incremental mining [20]. The CATSIM Tree performs the frequent pattern mining with support of incremental mining.

This paper is organized as follows. Section II discusses about some previous work related with the frequent pattern mining. Section III discusses the CATSIM Tree method. Section IV shows the experimental results, and Conclusion and future scope is discussed in section V.

II. PREVIOUS WORK

Let I = {i1, i2, . . . , im} be a set of items and let D be a database having a set of transactions where each transaction T is a subset of I. An association rule is an association relationship of the form: $X \rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \phi$. The support of rule $X \rightarrow Y$ is defined as the percentage of transactions containing both X and Y in D. The confidence of $X \rightarrow Y$ is defined as the percentage of transactions containing X that also contain Y in D. The task of association rules mining is to find all strong association rules that satisfy a minimum support threshold (*min_sup*) and a minimum confidence threshold (*min_conf*)[13] [15]. Given a user specified support threshold min_sup, X is called frequent itemset if sup(X) is greater than min_sup. From frequent itemsets association rules can be derived. This section provides discussion of some existing algorithms for frequent itemset mining [15].

A. Apriori based Algorithms

Apriori is the basic known algorithm for frequent pattern mining. To achieve efficient mining frequent patterns, an anti-monotonic property of frequent itemsets, called the Apriori heuristic was identified. It works on the principle of candidate generation and test. It is based on the downward closure property of itemset that if an itemset of length k is not frequent, none of its superset patterns can be frequent [10]. Before each data scan, candidate frequent item sets are verified whether they are frequent or not during the next data scan. There are many variations proposed to improve the efficiency of the Apriori algorithm, they are as follows [8].

1. **Hash-based technique:** To reduce the size of the candidate k-itemsets, a hash based technique can be used.
2. **Transaction reduction:** Here the anti monotone property of Apriori algorithm will be worked. In this case a transaction that does not contain any frequent k-itemsets can not contain any frequent (k+1) itemsets. Therefore, such a transaction can be removed from further consideration since subsequent scans of the database for j-itemsets, where j>k, will not require it [16].
3. **Partitioning:** Requires just two database scans to mine the frequent itemsets. It consist of two parts [17]. In Part I, the algorithm subdivides the transactions of D into n overlapping partitions. In Part 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.
4. **Sampling:** The basic idea of 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.
5. **Dynamic itemset counting**: A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by start points. In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately prior to each complete database scan [10].

However, the Apriori algorithm suffers from the following problems [9]:

1. To handle a huge number of candidate sets is costly. For, example if there are 10^5 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^9 length-2 candidates and test their occurrence frequencies.
2. To frequently scan the database and check a large set of candidates by pattern matching is complex task.

B. Pattern Growth Methods

To avoid the candidate generation and test procedure, a new method to find the frequent itemsets is found in [7] known as FP-Growth. A divide and conquer strategy is adapted by FP-Growth as follows: compress the database representing frequent items into a frequent pattern tree, but preserve the itemset association information, and then divide such a compressed database into a set of conditional databases, each associated with one frequent item, and mine each such database separately. This compact data structure can be designed on the following observations[16] [17].

1. If the set of frequent items of each transaction cam be stored in some compact data structure, it may be possible to avoid repeatedly scanning the original transaction database.
2. Because only frequent items will play a role in the frequent pattern mining, it is to required to perform one scan of transaction database to identify the set of frequent items.
3. If multiple transactions share a set of frequent items, it may be possible to merge the shared sets with the number of occurrences registered as count.

4. If two transactions share a common prefix, according to some sorted order of frequent items, the shared parts can be merged using one prefix structure as long as the count is registered properly.

Although FP-growth is more efficient than Apriori in many cases, it may still face problems in some cases as mentioned below [7]:

1. Space is requirement is higher to serve the mining. If the database is huge and sparse, the FP-tree will be large and the space requirement for recursion is a challenge.
2. Database contains all the cases. Real data sets can be sparse or dense in different applications.
3. Large applications need more scalability. Many existing methods are efficient when the data set is not very large.

C. CATS Tree and FELINE Algorithm

CATS Tree extends the idea of FP-Tree to improve the storage compression and allow frequent pattern mining without generation of candidate itemsets. The differences of FP-tree and CATS Tree are discussed here.

CATS Tree contains all items in every transaction. FP-Tree contains only frequent items. Single scan data mining is supported by CATS Tree, as compared to FP-Tree which supports two scan data mining. In CATS Tree sub-trees are locally optimized to improve compression, while sub-trees are locally optimized in FP-Tree. Items within a transaction do not need to be sorted in CATS Tree while items within a transaction are sorted in FP-Tree [19].

The CATS Tree satisfies the following properties.

1. No item of the same kind, i.e., nodes containing the same item label, could appear on the lower right hand side of that level item.
2. The compactness of CATS Tree measures how many transactions are compressed at a node. The compactness of CATS Tree is the highest at the root and the compactness decreases as a node is further away from the root.

The algorithm for the CATS Tree builder is briefly discussed in [19].The FELINE algorithm takes the CATS Tree as an input and mines the frequent items according to the algorithm given in [19]. However the CATS Tree and FELINE algorithm suffers from the following problems [19]:

I. It is expensive to build the tree without removing the items with frequency less than minimum support.

II. Swapping will take more times than the normal FP-Tree nodes.

III. CATSIM TREE

In the CATS Tree, all the items are stored as they are appearing in the sequence of particular transaction. The sequence may be changed if any of the lower leaf appears more time than the upper leaf. The procedure is continuous up to the last transaction of the database. So, in CATS Tree we can not predict which item will remain on top of the tree up to the last transaction. While in the CATSIM Tree all the items are stored either in the alphabetical or any other order as related with items.

As an example, suppose first transaction in any database contains D, F, H, B as items. So the tree will be formed with B as a root than D, F and H as leaves respectively as shown in the figure 1.

In the same database if the second transaction contains the items J, F, E, A than as per the alphabetic order A is the first in the second transaction. So if try to match with the first transaction than A is not present in the first transaction and also as per alphabetic order A comes before B, so new root will be created and on one side First transaction will come and on the other side second transaction will come as shown in the following figure.

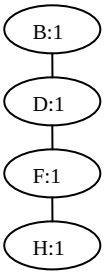


Figure 1

In the same database if the second transaction contains the items H, F, D, A than as per the alphabetic order A is the first in the second transaction. So if try to match with the first transaction than A is not present in the first

transaction and also as per alphabetic order A comes before B, so new root will be created and on one side First transaction will come and on the other side second transaction will come as shown in the following figure 2.

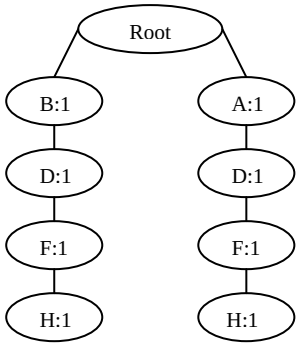


Figure 2

CATSIM Tree satisfies the following properties.

1. Items ordering is unaffected by the changes in frequency caused by incremental updates.
2. Node frequency in the CATSIM tree is at least as high as the sum of frequencies of its children

CATSSIM Tree algorithm is presented in the figure 3.

Input: a transaction database, minimum support threshold min_sup.

Output: The complete set of frequent patterns.

Method:

(a) CATSIM Tree is constructed in the following steps:

1. **sort** transaction according to the ascending order or alphabetical order
2. **create** the root node and arrange the elements according to the first transaction
3. **PROCEDURE** add(transaction t)
4. **if** (there is a common item between children nodes and t)
5. child_node.merge(t)
6. **else** t is added as a new child_node
7. **repeat** the procedure for all transaction
8. **PROCEDURE**merge(transaction t)
9. **increase** frequency of the node
10. **remove** item from t and call node.add(t)

(b) Mining of Frequent patterns is done in the following steps:

1. **calculate** the frequency of each item and sort according to the descending order

2. **if** (freq_item < min_sup)
   than remove the specified item from the tree
3. **repeat** the same procedure for each item.
4. **design** the new tree after the min_sup
5. **PROCEDURE** CATSIM(Tree, α)
6. **if** Tree contains a single path P than for each combination (denoted as β) of the nodes in the path P
7. **generate** pattern $\beta \cup \alpha$ with support = minimum support of nodes in β
8. **else** for each α_i in the header of the tree {
9. **generate** pattern $\beta = \alpha_i \cup \alpha$ with support =α_i .support
10. **construct** β's conditional pattern base and then β's conditional FP_tree $Tree_\beta$
11. **if** $Tree_\beta \neq \Phi$
12. **then** call CATSIM ($Tree_\beta$, β); }

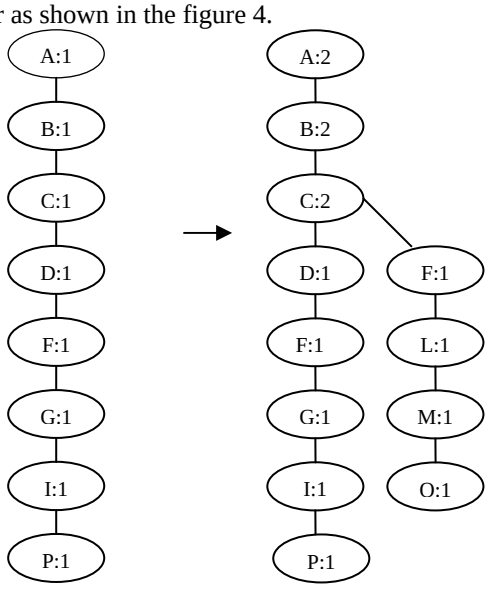
Figure 3: CATSIM Tree Algorithm

To illustrate, how the CATSIM tree is working, take the following database as an example.

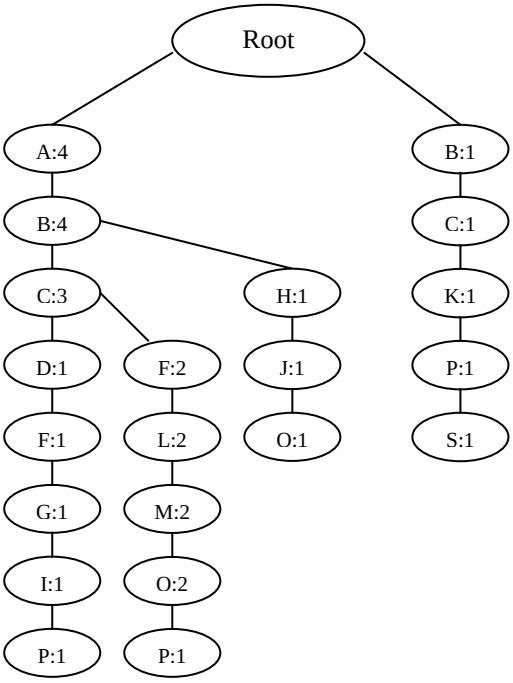
TID	Original Transaction
1	A,F,C,D,G,I,B,P
2	A,B,C,F,L,M,O
3	B,A,H,J,O
4	B,C,K,S,P
5	A,F,C,E,L,P,M,N

Table 1: Transaction Database

First all the elements of the transaction will be sorted according to the alphabetical order, and then the tree will be formed according to that order as shown in the figure 4.



(a) TID - 1 (b) TID - 2



(c) Final CATSIM Tree
Figure 4: CATSIM Tree construction

From the above tree, the frequency of all the elements will be found. Suppose min_sup=2

B : 5 , A : 4 , C : 4 , F : 3 , O : 3 , P : 3 , L : 2 , M: 2 , D : 1 , G : 1 , H: 1 , I: 1 , J: 1 , K : 1 , S : 1

Now the elements whose frequency is less than the min_sup will be removed from the tree. So the new tree will be formed as shown in Figure 5.

After that the mining of the frequent items will be done according to the FP-growth [6] approach as follows.

Step 1: The Conditional pattern base will be formed according to the ascending order of the items.

Step 2: Conditional FP-tree will be generated according to the same order as in step 1 by removing the items with the frequency less than the min_sup from the conditional pattern base.

Step 3: Finally frequent patterns will be generated from the conditional FP-tree.

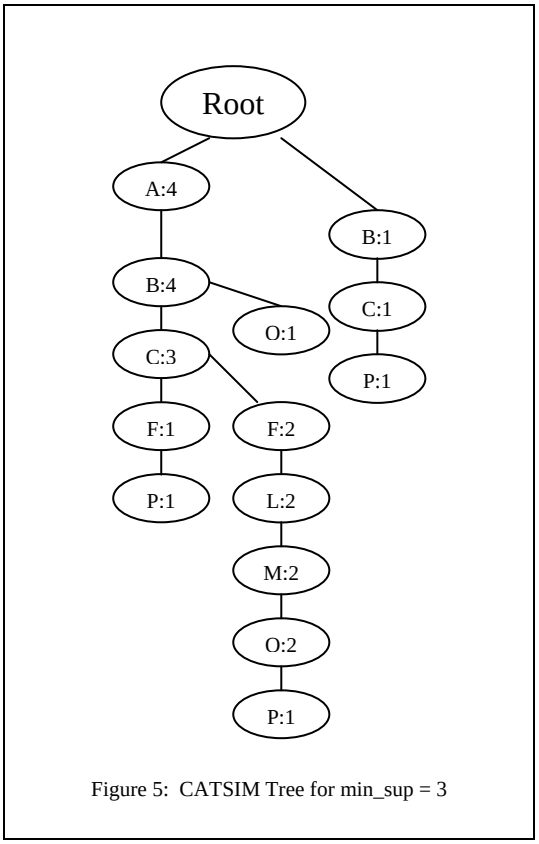


Figure 5: CATSIM Tree for min_sup = 3

IV. EXPERIMENTAL RESULTS

In the experiments, several transactions databases generated by the program developed at IBM and some real life databases from UC Irvine machine Learning Depository are used [6] [7]. The configuration of the machine used for experiments is Dual Core 2.0 GHz, with 160 GB Hard disk, 2 GB Expanded RAM, 945 Motherboard with Multimedia keyboard. OS used for the experiments is Windows XP. The Result produced are consistent.

Performance Comparison: Comparison of CATSIM with the CATS Tree and FP-growth algorithm is shown here. Apriori and FP-growth standard implementation is available [21]. In too many experiments it has been tested that CATS Tree is better than the Apriori and FP-growth algorithm, so here only comparison between the CATS and CATSIM Tree and FP-Growth has been shown. In the initial stage CATSIM Tree is much better than the CATS Tree as shown in Figure 6.

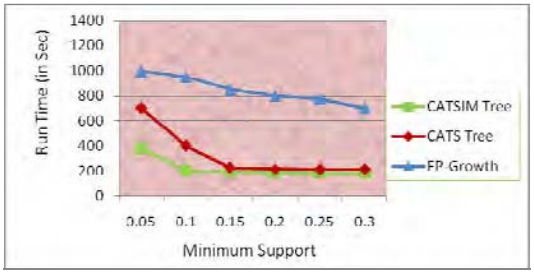


Figure 6: Comparisons for Runtime Vs MinSup

In CATS Tree algorithm it has been assumed that unlimited amount of memory is available, but it is always not true, so disk based memory has been used to store all the swapping required.

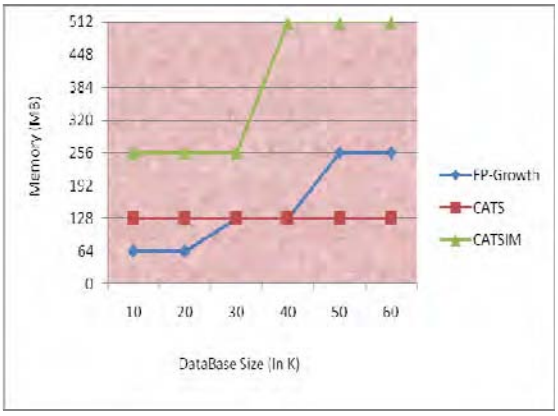


Figure 7: Comparison for Database size Vs Memory requirement

From the graph shown in the Figure 7, as the database size is changed there is no change in the CATS Tree because it contains the whole tree. In the comparison for FP-growth, CATS and CATSIM Tree the CATSIM Tree requires more memory because it is required to rearrange the transaction in alphabetical order. As per mathematical analysis for any minimum support if CATSIM Tree gives n frequent patterns, the CATS Tree always gives n or less than n frequent patterns.

Due to some movement in the upward and downward direction, the CATS Tree some times misses some of the frequent pattern, but the CATSIM Tree cannot miss any of the frequent patterns. So the overall result of the CATSIM Tree is better than CATS Tree.

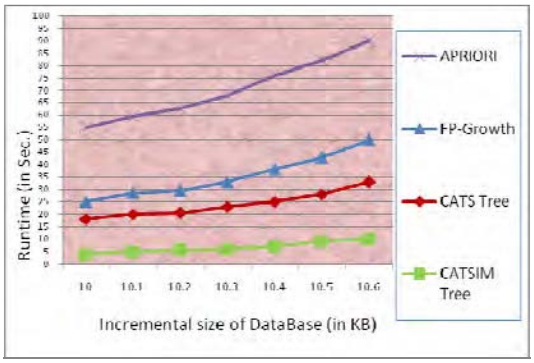


Figure 8: Incremental size of Database Vs. RunTime

Figure 8, shows the comparison for the four algorithms for incremental database. The Apriori algorithm works on the principle of candidate generate and test, so it requires the maximum execution time. The FP-Growth and CATS Tree algorithms are working for static database. If any modifications are proposed, as per the algorithm of FP-Growth and CATS, the tree generation procedure has to be started from the scratch. In the CATSIM Tree , if any transaction is going to be added or deleted there is a provision to make changes directly in the existing tree. So for incremental size of the database, CATSIM Tree is better than any of the existing algorithms.

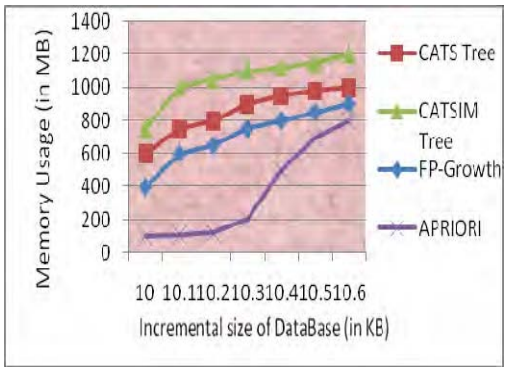


Figure 9: Incremental size of Database Vs Memory Usage

Experiment for the incremental size of database and memory usage is shown in figure 9. The Apriori, FP-Growth, and CATS are working on the principle of regeneration of the tree, so these three algorithms use the same memory that had been used previously to construct the tree. While in the case of CATSIM Tree, it requires

more memory in the normal static database conditions, so also in the incremental size of the database it requires more memory.

V. CONCLUSION AND FUTURE SCOPE

Implementation of all the previously studied algorithms uses main memory to reduce the execution time, but due to growth of database size now it is recommended to use disk based memory to store the tree of exponential size. CATS tree and FELINE algorithm are implemented with the use of disk memory and experiments show that the execution time delay is negligible. The CATSIM tree provides users with efficient incremental mining. The tree size can be exponential for the case of dense data, so there is a need in the improvement in the tree structure which reduces the tree size and make it scalable to handle large database which is highly incremental in nature. To use FP-Tree based algorithm for variety of databases is a open research problem. So, attempt may be made to use concept of CATSIM for heterogeneous databases.

REFERENCES

[1] A.Erwin, R.P.Gopalan and N.R.Achuthan, "CTU-Mine: An efficient High Utility Itemset Mining Algorithm using Pattern growth approach" Seventh International conference on Computer and Information Technology,2007.

[2] C. Borgelt, " Recursive pruning for the Apriori Algorithm", School of computer science, Germany

[3] C. Borgelt, " An Implementation of the FP-growth Algorithm" OSDM'05, August 21, 2005, Chicago, Illinois, USA, www.cs.rpi.edu/~zaki/OSDM05/papers/p1-borgelt.pdf.

[4] D. J. Chai, B. Hwang, K.H. Ryu, "Frequent pattern mining using Bipartite Graph ", 18th international workshop on Database and Expert Systems Applications , IEEE 2007

[5] H. T. HE, S.L. Zhang, "A New Method for Interactive Frequent Itemsets Mining", IJCSES International Journal of Computer Sciences and Engineering Systems, Vol.1, No.3, July 2007

[6] http://fimi.cs.helsinki.fi/data

[7] http://www.almaden.ibm.com/cs/quest//syndata.html#assocSynData

[8] J. Han and M. Kamber, "Data Mining, Concept and Techniques", 578 pages, books.google.co.in.

[9] J. Pei, "Pattern-Growth methods for frequent pattern mining " Ph. D. Thesis, Simon Fraser University, 2002, ftp://fas.sfu.ca/pub/cs/theses/2002/JianPeiPhD.pdf.

[10] J. Su, W. Lin, "CBW: An Efficient Algorithm for Frequent Itemset Mining", Proceedings of the 37th Hawaii International Conference on System Sciences – 2004.

[11] M.C. Fernandez, E. menasalvas, O.Marban, J.M. Pena, S.Millan, "Minimal Decision Rules based on the Apriori algorithm", International Journal of Applied Mathematics and Computer Science , 2001, Volume 11, No. 3, 691-704.

[12] M. Hegland, " The Apriori Algorithm- A Tutorial", Lecture Notes Series, WSPC, March 2005

[13] M. Song and S. Rajasekaran , " A Transaction Mapping Algorithm for Frequent Itemsets Mining" IEEE Transactions on Knowledge and Data Engineering (TKDE) *www.engr.uconn.edu/~rajasek/JArticles.htm*

[14] Q. I. Khan, T. Hoque and C.K. Leung, " CanTree : A Tree structure for Efficient Incremental mining of Frequent Patterns". Proceeding of the Fifth International conference on Data Mining (ICDM'05), csdl2.computer.org/.../&toc=comp/proceedings/icdm/2005/2278/00/2278toc.xml&DOI=10.1109/ICDM.2005.

[15] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules", Proceedings of the 20th VLDB Conference Santiago, Chile, 1994.

[16] R. Dass and A. Mahanti, " An efficient heuristic search for Real-Time frequent pattern mining". International Conference on System Sciences – 2006, ieeexplore.ieee.org/iel5/10548/33362/01579371.pdf

[17] R. Dass, A. Mahanti, " An Efficient Algorithm for Real-Time Frequent Pattern Mining for Real-Time Business Intelligence analytics", Proceedings of the 39th Hawaii International Conference on System Sciences – 2006.

[18] S. Patel and S. Garg, " Basic Frame work of CATSIM Tree for efficient frequent pattern mining" (In Communication with "Infocomp journal of Computing " , Bazil)

[19] W. Cheung, "Frequent Pattern mining without candidate generation or support constraint." Master's thesis, University of Alberta, 2002, SPRING '03, doi.ieeecomputersociety.org/10.1109/IDEAS.2003.1214917.

[20] W. Cheung and O. R. Zaiane, " Incremental Mining of Frequent Patterns without candidate Generation or Support Constraint", IDEAS'03, doi.ieeecomputersociety.org/10.1109/IDEAS.2003.1214917.

[21] www.cse.cuhk.edu.hk/~kdd/program.