Comparative Analysis of Various Approaches Used in Frequent Pattern Mining

Deepak Garg, Hemant Sharma
Thapar University, Patiala

***Abstract*—Frequent pattern mining has become an important data mining task and has been a focused theme in data mining research. Frequent patterns are patterns that appear in a data set frequently. Frequent pattern mining searches for recurring relationship in a given data set. Various techniques have been proposed to improve the performance of frequent pattern mining algorithms. This paper presents review of different frequent mining techniques including apriori based algorithms, partition based algorithms, DFS and hybrid algorithms, pattern based algorithms, SQL based algorithms and Incremental apriori based algorithms. A brief description of each technique has been provided. In the last, different frequent pattern mining techniques are compared based on various parameters of importance. Experimental results show that FP- Tree based approach achieves better performance.**

Keywords- Data mining; Frequent patterns; Frequent pattern mining; association rules; support; confidence; Dynamic item set counting.

I. INTRODUCTION

Frequent patterns are item sets, subsequences, or substructures that appear in a data set with frequency no less than a user-specified threshold. Frequent pattern mining is a first step in association rule mining. One of the major uses with association rules is to analyze large amount of supermarket basket transactions [1-3]. Recently, association rules have been applied to other areas like outlier's detection, classification, clustering etc [4, 6, 8].

Association rules mining can formally be defined as follows. Let I = $\{i_1, i_2, i_3, .., i_m\}$ be a set of attributes called items. Let D be a set of transactions. Each transaction t ϵ D consists of a set of items such that t $\subseteq$ I. A transaction t is said to contain an item set X if and only if all items within X are also contained in t. Each transaction also contains a unique identifier called transaction identification (TID). Support of an item set is normalized number of occurrences of the item set within the dataset. An item set is considered as frequent or large, if the item set has a support that is greater or equal to the user specified minimum support [25-26].

The most common form of association rules is implication rule which is in the form of X $\Rightarrow$ Y, where X $\subset$ I, Y $\subset$ I and X $\cap$ Y = Φ. The support of the rule X Y is equal to the percentage of transactions in D containing X $\Rightarrow$ Y. The confidence of the rule X Y is equal to the percentage of transactions in D containing X also containing Y. Once the required minimum support and confidence are specified, association rule mining becomes finding all association rules that satisfy the minimum support requirements. The problem can be further broken down into two steps: mining of frequent item sets and generating association rules.

The number of possible combinations of item sets increases exponentially with I and the average transaction length. Therefore it is infeasible to determine the support of all possible item sets. When counting the supports of item sets, there are two strategies. The first strategy is to count the occurrences directly, whenever an item set is contained in a transaction, the occurrence of the item set is increased. The second strategy is to count the occurrences indirectly by intersecting TID set of each component of the item set. The TID set of a component X, where X can be either item or item set, is denoted as X.TID. The support of an item set S = X $\cup$ Y is obtained by intersecting X.TID $\cap$ Y.TID = S.TID and the support of S equals S.TID [28, 29].

II. VARIOUS FREQUENT PATTERN MINING TECHNIQUES

A. *Apriori-based Algorithms*

The first published frequent item set mining algorithm is Apriori [1]. Apriori uses breadth first search (BFS). At each level, Apriori reduces the search space by using downward closure property of item set. If an item set of length k is not frequent, none of its superset patterns can be frequent. Candidate frequent item sets, C_k where k is the length of the item set, are generated before each data scan. The supports of candidate frequent item sets are counted. Candidate k item sets, C_k are generated with frequent (k – 1) item sets. Apriori algorithm achieves good performance by reducing the candidate item sets iteratively. The problem, however, associated with Apriori is it requires k data scans to find all frequent k-item sets. It is very much expensive to scan the large data base. Dynamic Item set Counting, (DIC) relaxes the strict separation between generating and counting of item sets [4]. DIC starts counting the support of candidate frequent item sets as soon as they are being generated. By overlapping counting and candidate item set generation, DIC reduces the overall data scans required. Orlando et al. [13] proposed an algorithm that combines transaction reduction and direct data access. At the end of each scan, transactions that are potentially useful are used for the next iteration. A technique called scan reduction uses candidate 2 item sets to generate subsequent candidate item sets [12]. If all intermediate data can be held in the main memory, only one scan is required to generate all candidate frequent item sets. Another data scan is required to verify whether the candidate frequent item sets are frequent or not.

With all of those improvements, the number of data scans required by Apriori based algorithms has been reduced significantly. However, the cost of generating candidate frequent item sets has not been fully addressed by Apriori based algorithms. This problem becomes visible when there are huge numbers of frequent 1 or 2 item sets.

B. Partition-based Algorithms

Partition-based Algorithms [15] solves the problem of high number of database scans, associated with Apriori-based algorithm. It requires two complete data scan to mine frequent item sets. The Partition algorithm divides the dataset into many subsets so that each subset can be fitted into the main memory. The basic idea of the Partition-based algorithm is that a frequent item set must be frequent in at least one subset. Partition-based algorithm generates local frequent item sets for each partition during the first data scan. Since the whole partition can be fitted into the main memory, the complete local frequent item sets can be mined without any disk I/O operations. The local frequent item sets are added to the global candidate frequent item sets. In the second data scan, false candidates are removed from the global candidate frequent item sets. In a special case where each subset contains identical local frequent item sets, Partition algorithm can mine all frequent item sets with a single data scan. However, when the data is distributed unevenly across different partitions, this algorithm may generate a lot of false candidates from a small number of partitions. By employing the knowledge collected during the mining process, false global candidate frequent item sets are pruned when they are found that they cannot be frequent. In addition, those algorithms reduce the number of scans in the worse case to (2b- 1)/b where b is the number of partitions.

C. DFS and Hybrid Algorithms

Eclat and Clique [16] combine both depth first search (DFS) and intersection counting. Since intersection counting is used, no complicated data structure is required. These hybrid algorithms reduces the memory requirement, since only the TID sets of the path item sets from the root to the leaves have to be kept in the memory simultaneously. Intersection of TID sets can be stopped as soon as the remaining length of the shortest TID set is shorter than the required support minus the counted support value. The intersection of TID sets of 1-item set to generate frequent 2 item sets is expensive.. The maximal hyper graph clique clustering is applied to 2-frequent item sets to generate a refined set of maximal item sets. Hipp et al. [10] pointed out that DFS cannot prune candidate k item sets by checking frequent (k – 1) item sets, because DFS searches from the root to the leaves of the tree without using any subsets relationship. A hybrid approach of BFS and DFS is proposed in [11]. It is cheaper to use item set counting with BFS to determine the supports, when the number of candidate frequent item sets is small. When the number of candidate frequent item sets is relatively large, the hybrid algorithm switches to TID set intersection with DFS, since simple TID set intersection is more efficient than occurrence counting when the number of candidate frequent item sets is relatively large. This results in additional costs to generate TID sets. The authors proposed [11] to use hash-tree-like structure to minimize the cost of transition. However, the authors do not provide an algorithm to determine the best condition to switch the strategy. In the evaluation, the authors provide parameters to change in strategy. However, those parameters may not be generalized enough for all kinds of datasets. Incorrect timing of changing strategy may decrease the performance of hybrid algorithm.

D. Pattern-Growth Algorithms

Two major costs of Apriori based algorithms are the cost to generate candidate frequent item sets and the cost associated with I/O operations. The issues related to I/O have been addressed, but the issues related to candidate frequent item sets generation remain open. If there are n frequent 1 item sets, Apriori based algorithms would require to generate approximately $n^2/2$ candidate frequent item sets. Secondly, the memory required to hold the candidate frequent item sets and their supports could be substantial. For example, when n equals 10,000, there would be more than 10^8 length 2 candidate frequent item sets. If we assume that it requires 4 bytes to hold the support and 4 bytes to hold the item sets, approximately 0.5 gigabytes of main memory would be needed to store the information [18]. Furthermore, the memory required does not include the overhead associated with the data structure. Also the cost required to count the support of candidate item sets may be large. As far as Apriori-based algorithms are concerned, the run time increases as the support value decreases. Therefore, the cost of candidate frequent item sets generation of Apriori based algorithms will exceeds than the cost of I/O [24]. Han et al. [9] proposed a data structure called frequent pattern tree or FP Tree. FP-growth mines frequent item sets from FP-Tree without generating candidate frequent item sets. FP-Tree is an extension of prefix tree structure. Only frequent items get stored in the tree. Each node contains the item's label along with its frequency. The paths from the root to the leaves are arranged according to the support value of the items with the frequency of each parent is greater than or equal to the sum of its children's frequency. The construction of FP-Tree requires two data scans. In the first scan, the support value of each item is found. This calculated support values are used in the second scan to sort the items within transactions in descending order. If two transactions share a common prefix, the shared portion is merged and the frequencies of the nodes are incremented accordingly. Nodes with the same label are connected with an item link. The item link is used to facilitate frequent pattern mining. In addition, each FP-Tree has a header that contains all frequent items and pointers to the beginning of their respective item links. FP-growth partitions the FP-Tree based on the prefixes. FP-growth traverses the paths of FP-Tree recursively to generate frequent item sets. Pattern fragments are concatenated to ensure all frequent item sets are generated properly. Thus FP-growth avoids the costly operations for generation and testing operations of candidate item sets. When the data is sparse, the compression achieved by the FP-Tree is small and the FP Tree is bushy. As a result, FP-growth would spend a lot of effort to concatenate fragmented patterns with no frequent item sets being found. A new data structure called H-struct is introduced in [14]. In this, transactions are sorted with an arbitrary ordering scheme. Only frequent items are projected in the H-struct. H-struct consists of projected transactions and each node in the projected transactions contains item label and a hyper link pointing to the next occurrence of the item. A header table is created for H-struct. The header contains frequencies of all items, their supports and hyper link to the

first transaction containing given item. H-mine mines the H-struct recursively by building a new header table for each item in the original header with subsequent headers omitting items that have been mined previously. For each sub- header, H-mine traverses the H-struct according to the hyper links and finds frequent item sets for the local header. At the same time, H-mine builds links for items that have not been mined in the local header. Those links are used to find conditional frequent patterns within the local header. The process is repeated until all frequent item sets have been mined. In case of a dense dataset, H-struct is not as efficient as FP-Tree because FP-Tree allows compression.

E. Incremental Update with Apriori-based Algorithms

Complete dataset is normally huge and the incremental portion is relatively small compared to the complete dataset. In many cases, it is not feasible to perform a complete data mining process while transactions are being added continuously. Therefore, incremental data mining algorithms have to reuse the existing information as much as possible, so that either computational cost and/or I/O cost can be reduced. A general incremental mining algorithm called Fast Update 2 (FUP2), that allows both addition and deletion of transactions was proposed in [7]. The major idea of FUP2 is to reduce the cost of candidate frequent item sets generation. Incremental portion of the dataset is scanned; frequent patterns in the incremental data are compared with the existing frequent item sets in the original dataset. Previous frequent item sets are removed if they are no longer frequent after the incremental portion of the data is added or removed. The supports of previous frequent item sets that are still frequent are updated to reflect the changes. In those ways, previous frequent item sets that are still frequent are not required to be checked for their supports again. New (k + 1) candidate frequent item sets are generated from frequent k item sets. The entire updated dataset is scanned to verify those newly added candidate item sets if they are indeed frequent. The process is repeated until the set of candidate frequent item set becomes empty. FUP2 offers some benefits over the original Apriori algorithm. However, it still requires multiple scans of the dataset. Another incremental Apriori based algorithm is called Sliding Window Filtering (SWF) [12]. SWF incorporates the main idea of Partition algorithm with Apriori to allow incremental mining. SWF divides the dataset into several partitions. During the scan of partitions, a filtering threshold is employed in each partition to generate candidate frequent 2 item sets. When a candidate 2 item set is found to be frequent in the newly scanned partition, the partition number and the frequency of the item set are stored. Cumulative information about candidate frequent 2 item sets is selectively carried over toward subsequence partition scans. Cumulative frequencies of previous generated candidate frequent 2 item sets are maintained as new partitions are being scanned. False candidate frequent item sets are pruned when the cumulative support of the candidate frequent item sets fall below required proportional support since they have become frequent. Once incremental portion of the dataset is scanned, scan reduction techniques are used to generate all subsequence candidate frequent items sets [5]. Another data scan over the whole dataset is required to confirm the frequent item sets. In the case of data removal, the partition to be removed are scanned, the cumulative count and the start partition number of candidate length 2 item sets are modified accordingly. Although SWF achieves better performance than pervious algorithms, the performance of SWF still depends on the selection of partition size and removal of data can only be done at partition level.

F. SQL-based algorithms

DBMS can facilitate data mining to become an online, robust, scalable and concurrent process by complementing the existing querying and analytical functions. The first attempt to the particular problem of integrated frequent item set mining was the SETM algorithm [10, 17], expressed as SQL queries working on relational tables. The Apriori algorithm [1] opened up new prospects for FIM. The database- coupled variations of the Apriori algorithm were carefully examined in [19]. The SQL-92 based implementations were too slow, but the SQL implementations enhanced with object-relational extensions (SQL-OR) performed acceptable. The so- called Cache-Mine implementation had the best overall performance, where the database-independent mining algorithm cached the relevant data in a local disk cache [21-23]. SQL based frequent mining using FP-tree provide best performance than other SQL based techniques [20]. Although an FP-tree is rather compact, it is unrealistic to construct a main memory- based FP-tree when the database is large. However using RDBMSs provides us the benefits of using their buffer management systems specially developed for freeing the user applications from the size considerations of the data. And moreover, there are several potential advantages of building mining algorithms to work on RDBMSs. An interesting alternative is to store a FP-tree in a table. There are two approaches in this category - FP, EFP (Expand Frequent Pattern). They are different in the construction of frequent pattern tree table, named FP. FP approach checks each frequent item whether it should be inserted into a table FP or not one by one to construct FP. EFP approach introduces a temporary table EFP, thus table FP can generate from EFP. According to the properties of FP-tree, FP-tree can be presented by a table FP with three column attributes: item identifier (item), the number of transactions that contain this item in a sub- path (count), and item prefix sub-tree (path). The field path is beneficial not only to construct the table FP but also to find all frequent patterns from FP. In the construction of table FP, the field path is an important condition to judge if an item in frequent item table F should be insert into the table FP or update the table FP by incrementing the item's count by 1. If an item does not exist in the table FP or there exist the same items as this item in the table FP but their corresponding path are different, insert the item into table FP. In the process of constructing conditional pattern base for each frequent item, only need to derive its entire path in the table FP as a set of conditional paths, which co-occurs with it.

III. COMPARISON OF VARIOUS FREQUENT PATTERN MINING TECHNIQUES

Comparison of different FPM techniques is given in Table 1, where A is length of maximal frequent item set and B is number of partitions. As Shown In the table, various algorithms are compared against four parameters, number of database scans required for the generation of frequent item set, the candidate generation technique used, whether the frequent item generation approach is incremental or not, and how the

algorithm is sensitive to the change in user parameters. Apriori-based methods use efficient technique for pruning the candidate item sets, but they require lots of computational time as well as multiple database scans to generate candidate item sets. Partition-based methods limit the size of candidate item sets. Partition algorithm may generate a lot of false candidates from a small number of partitions. FP-Tree based methods require only two database scans in order to generate frequent patterns. These methods use a compact tree- structure to represent the entire database. They do not require candidate generation, reducing the computational cost.

IV. Comparison of Apriori and Primitive Association Rule Mining

Comparison of the algorithms, Apriori and Primitive Association Rule Mining is given in this section. There are many advantages of Primitive Association Rule Mining over Apriori.

Apriori uses candidate Generate function for generating every candidate k-item sets and it takes enormous amount of time to generate candidate k+1-item sets from large k item sets. However, Primitive Association Rule Mining does not use this function; instead it uses graph based approach after generating of large 2–item sets.

In primitive association, a graph is constructed with large two item sets. Using graph, large three item sets can be generated easily without scanning the database.

At each pass in primitive association, it is enough to use graph with k large item sets for generating k+1 candidate item sets. Traversal of one link list (adjacency list) takes less time as compared to Apriori generation function.

Secondly in Apriori approach we are accessing transaction as a whole or we can divide into parts but it takes lot of memory whereas in Primitive Association Rule Mining, transactions are converted into bit vector which is based on items. Bit vector representation takes very less times as well as memory, theoretically 32 times less. Primitive Association Rule Mining takes less time since transactions are represented in bit vector form, and we are using logical AND, OR operation which is very fast. Further the bit representation consumes less memory also.

A. Comparison of AprioriTid and Apriori Hybrid

AprioriTid and AprioriHybrid are just variations of Apriori. In Apriori, at every step, we have to find candidate k-item sets, and we have to scan whole database at each k, which is time consuming. So, AprioriTid algorithm has given a solution for finding candidate k-item sets without scanning whole transaction. This algorithm works on the basis of transaction Id that is associated with every transaction. Apriori Hybrid is combined approach of Apriori and Apriori Tid, in which if some part of transactions (which is stored in other place) do not fit into the memory then use Apriori algorithm, otherwise swap Apriori algorithm to Apriori Tid.

In general, using Apriori AprioriTid and AprioriHybrid algorithms we can find frequent item sets, whereas we assume that items and transactions have equal weights. Sometimes, it is important to know that whether every items have equal weights or not, if all items have equal weights then Apriori and their variation can do good job for finding frequent item sets, and if weights of items are not equal, then Apriori and their variations do not work. So, to solve this problem, we have two solutions, whether we can assign weights to items and transactions, or to use some algorithms, so that it can give weights of items and weights of transactions. If we have weights of items in the beginning in the database then we can find frequent item sets using weighted association rule of mining, otherwise we can use association rule of mining without pre-assign weights, which gives weights of items and weights of transactions using HITS algorithm.

TABLE I. Comparison Of Various Frequent Pattern Mining

	Apriori-Based	Partition Based	Incremental Apriori	FP Tree	SQL Based
Number of Database Scan For Best Case Scenario	2	1	2	2	1
Number of Database Scan For Worst Case	A+1	(2B-1)/B	A+1	2	1
Candidate Generation Needed or Not	Yes	Yes	Yes	No	No
Incremental Mining Possible	No	No	Yes	No	No
Sensitive to Change in User Parameter	Yes	Yes	Yes	Yes	Yes

B. Experimental Results

Following are real life datasets which were taken, these are:

Kosarak- The kosarak dataset comes from the click-stream data of a Hungarian online news portal, Number of Instances =990,002, Number of Attributes= 41,270.

TABLE II. KOSARAK DATABASE

Large Item Sets	Time taken by Apriori	Time taken by Primitive
3	0.46	0.29
4	2.166	1.86
5	12.04	11.11

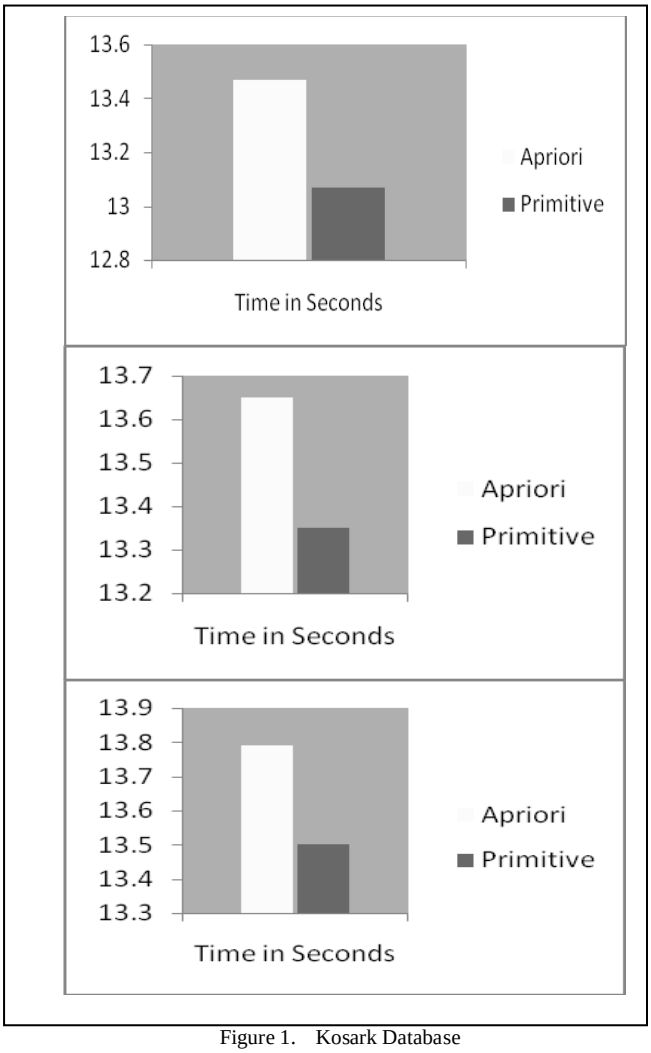

Figure 1. Kosark Database

The results clearly show that Primitive algorithm is taking less time as compared to the apriori algorithm.

Mushroom- This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom. Number of Instances = 8124, Number of Attributes = 22.

TABLE III. FOR MUSHROOM DATABASE

Large Item Sets	Time taken by Apriori	Time taken by Primitive
3	13.47	13.07
4	13.65	13.35
5	13.79	13.5

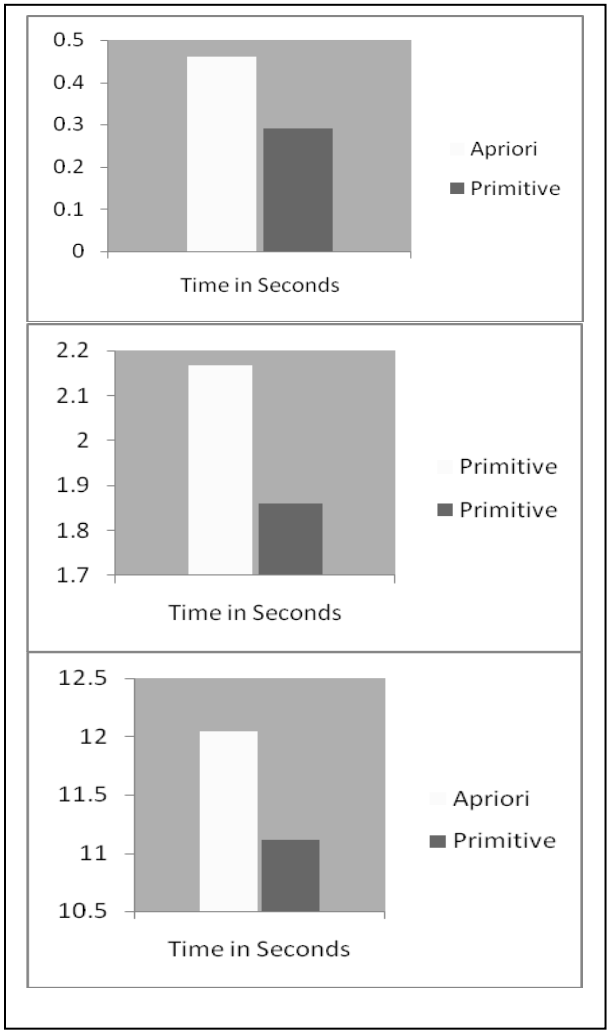

Figure 2. Mushroom Database

Experimental result clearly shows that Apriori is taking more time.

Chess - A game datasets.

Attribute Information: Classes (2): -- White-can-win ("won") and White-cannot-win ("nowin"). It believes that White is deemed to be unable to win if the Black pawn can safely advance. Number of Instances= 3196, Number of Attributes=36.

TABLE IV. FOR CHESS DATABASE

Large Item Sets	Time taken by Apriori	Time taken by Primitive
3	0.41	0.37
4	4.64	4.04
5	50.43	43.43

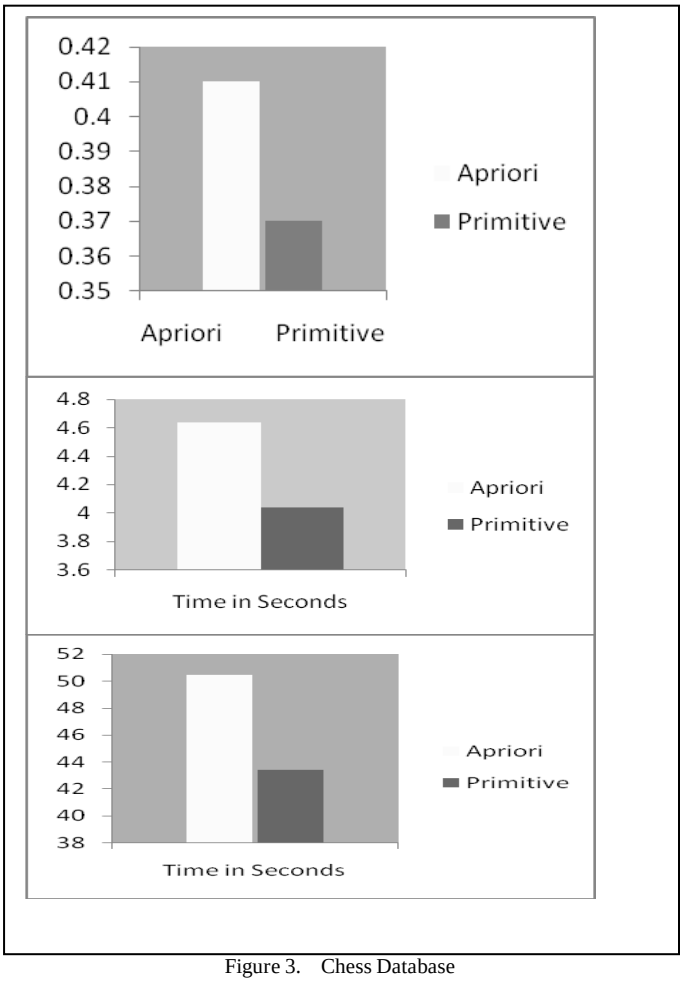

Figure 3. Chess Database

By looking at the above results it is clear that FP- Tree based approach are showing a clear edge because the number of database scans required are less which in turn reduces the computational time. Because the database is represented in tree structures which are taking less space so the overall memory requirement reduces.

CONCLUSION

Frequent pattern mining is the first step for association rule mining. Association rule mining has found many applications other than market basket analysis, including applications in marketing, customer segmentation, medicine, e-commerce, classification, clustering, web mining, bioinformatics and finance. Various techniques have been found to mine frequent patterns.

Each technique has its own pros and cons. Performance of particular technique depends on input data and available resources. Among all of the techniques discussed above, FP-Tree based approach achieves better performance by requiring only two database scans hence reducing the computational time. It takes less memory by representing large database in compact tree-structure. But a word of caution here that association rules should not be used directly for prediction without further analysis or domain knowledge. They are, however, a helpful starting point for further exploration & understanding of data. Experimental results have shown advantages of Primitive Association Rule Mining over Apriori.

REFERENCES

[1] Agrawal Rakesh, Imilienski T., and Swami Arun. Mining association rules between sets of items in large datasets. SIGMOD, 207-216, 1993

[2] Bayardo Roberto J. Efficiently Mining Long Patterns from Databases. SIGMOD, 83-93, Seattle, Washington, June 1998

[3] Brin Sergey, Motwani Rajeev, and Silverstein Craig. Beyond market baskets: Generalizing association rules to correlations. SIGMOD, 265-276, Tucson, AZ, USA, May 1997

[4] Brin Sergey, Motwani Rajeev, Ullman Jeffrey D., and Tsur Shalom. Dynamic itemset counting and implication rules for market basket data. SIGMOD, Tucson, AZ, USA, May 1997

[5] Chen Ming Syan, Park J. S., and Yu P. S. Efficient Data Mining for Path Traversal Patterns. IEEE Transactions on Knowledge and Data Engineering 10(2), 209-221, 1998

[6] Chen Xiaodong and Petrounias Ilias. Discovering temporal association rules: Algorithms, language and system. 2000 IEEE 16th International Conference on Data Engineering,San Diego, CA, USA, February 2000

[7] Cheung David W., Lee S. D., and Kao Benjamin. A General Incremental Technique for Maintaining Discovered Association Rules. Proc.International Conference On Database Systems For Advanced Applications, April 1997

[8] Han Jiawei, Pei Jian, Mortazavi-Asl Behzad, Chen Qiming, Dayal Umeshwar, and Hsu Mei-Chun. FreeSpan: Frequent pattern-projected sequential pattern mining. Boston, Ma, August 2000

[9] Han Jiawei, Pei Jian, and Yin Yiwen. Mining Frequent Patterns without Candidate Generation. SIGMOD, 1-12, Dallas, TX, May 2000

[10] Hipp Jochen, Güntzer Ulrich, and Nakhaeizadeh Gholamreza. Algorithms of Association Rule Mining - A General Survey and Comparison. SIGKDD Explorations 2(1), 58-64, 2000

[11] Hipp Jochen, Güntzer Ulrich, and Nakhaeizadeh Gholamreza. Mining Association Rules: Deriving a Superior Algorithm by Analyzing Today's Approaches. 159-168, Lyon, France, September 2000

[12] Lee Chang Hung, Lin Cheng Ru, and Chen Ming Syan. Sliding Window Filtering: An Efficient Method for incremental Mining on a Time-Variant Database. Proceedings of 10th International Conference on Information and Knowledge Management, 263-270, November 2001

[13] Orlando Salvatore, Palmerini P., and Perego Raffaele. Enhancing the Apriori Algorithm for Frequent Set Counting. 3rd International Conference on Data Warehousing and Knowledge Discovery, Germany, September 2001

[14] Pei Jian, Han Jiawei, Nishio Shojiro, Tang Shiwei, and Yang Dongqing. H-Mine: Hyper- Structure Mining of Frequent Patterns in Large Databases. Proc.2001 Int.Conf.on Data Mining, San Jose, CA, November 2001

[15] Savasere Ashok, Omiecinski Edward, and Navathe Shamkant. An Efficient Algorithm for Mining Association Rules in Large Databases. Proceedings of the Very Large Data Base Conference, September 1995

[16] Zaïane Osmar R. and Oliveira Stanley R. M. Privacy preserving frequent itemset mining. Workshop on Privacy, Security, and Data Mining, in conjunction with the IEEE International Conference on Data Mining, Japan, December 2002

[17] M. Houtsma and A. Swami. Set-oriented data mining in relational databases. Data Knowl. Eng.,245–262, 1995

[18] Christian Borgelt, An Implementation of the FP-growth Algorithm, OSDM'05, 2005

[19] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: alternatives and implications. In SIGMOD, International conference on Management of data, pages 343–354, 1998

[20] X. Shang, K.-U. Sattler, and I. Geist. SQL based frequent pattern mining with fp-growth. In INAP/WLP, pages 32–46, 2004

[21] R. Agrawal and K. Shim. Developing tightly-coupled data mining application on a relational database system. In Proc.of the 2nd Int. Conf. on Knowledge Discovery in Database and Data Mining, Portland,Oregon, 1996

[22] J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: A data mining query language for relational database. In Proc. Of the 1996 SIGMOD workshop on research issues on data mining and knowledge discovery, Montreal, Canada, 1996

[23] R. Meo, G. Psaila, and S. Ceri. A new SQL like operator for mining association rules. In Proc. Of the 22nd Int. Conf. on Very Large Databases, Bombay, India, 1996

[24] Wang Ke, Tang Liu, Han Jiawei, and Liu Junqiang. Top down FP-Growth for Association Rule Mining. Proc.Pacific- Asia Conference, PAKDD 2002, 334-340, Taipei, Taiwan, May 2002

[25] Ozden Banu, Ramaswamy Sridhar, and Silberschatz Avi. Cyclic association rules. The 1998 14th International Conference on Data Engineering, 412-421, Orlando, FL, USA, February 1998

[26] Wang Jiinlong, Xu Conglfu, Cben Weidong, Pan Yunhe, Survey of the Study on Frequent Pattern Mining in Data Streams, 5917-5920, IEEE International Conference on Systems, Man and Cybernetics, 2004

[27] Jiawei Han, Hong Cheng, Dong Xin, Xifeng Yan, Frequent pattern mining: current status and future directions, 57-60, Data Mining Knowledge Discovery, 2007

[28] R. srikant, R. Agarwal, Mining sequential patterns: generalization and performance improvements, 1-15, IBM Research report, 1996

[29] Balazs Racz,Ferenc Bodon,Lars Schmidt-Thieme., On Benchmarking Frequent Itemset Mining Algorithms, from Measurement to Analysis, 37-42, Chicago, Illinois, USA, august 2005.