International Journal of Soft Computing and Engineering (IJSCE)
ISSN: 2231-2307, Volume-1, Issue-2, May 2011

# A Survey of Efficient Algorithms and New Approach for Fast Discovery of Frequent Itemset for Association Rule Mining (DFIARM)

**Anurag Choubey, Ravindra Patel, J.L. Rana**

**Abstract—The problem of mining association rules has attracted lots of attention in the research community. Several techniques for efficient discovery of association rule have appeared. With abundant literature published in research into frequent itemset mining and deriving association rules, if the question is raised that whether we have solved most of the critical problems related to frequent itemset mining and association rule discovery. Based on the scope of the recent literature, the answer will be negative. The most time consuming operation in discovering association rule, is the computation of the frequency of the occurrences of interesting subset of items (called candidates) in the database of transactions. Can one develop a method that may avoid or reduce candidate generation and test and utilize some novel data structures to reduce the cost in frequent pattern mining? This is the motivation of my study for mining frequent-itemsets and association rules. In this paper we review some existing algorithms for frequent itemset mining and present a proposal of our new approach.**

*Index Terms* — **Data mining, Frequent Item-set mining, Association Rule Mining.**

## I. INTRODUCTION

The discovery of association rules in transaction databases is an important data-mining problem because of its wide application in many areas, such as market basket analysis, decision support, financial forecast, collaborative recommendation, and prediction. Prediction is a process, for example, given a set of rules that describe the shopping behavior of the customers in a store over time, and some purchases made by a particular customer, we wish to predict what other purchases will be made by that customer. Many techniques have been proposed for prediction in the past. In addition to the classical decision-tree induction approach, there are Bayesian classifications, neural network, nearest neighbor classifiers, case-based reasoning, genetic algorithm, rough set, fuzzy set, and data mining approaches.

**Manuscript received May 13, 2011**.

**Anurag Choubey**, Dean Academic, Technocrats Institute of Technology, Bhopal (Madhya Pradesh), India, (E-mail: choubeyanurag1@gmail.com)

**Dr. Ravindra Patel**, Reader & Head, Department of Computer Application, UIT-RGPV, Bhopal(M.P.), India Bhopal (Madhya Pradesh), India (E-mail: ravindra@rgtu.net)

**Dr. J.L. Rana**, Ex. Professor & Head, Department of Computer Science & Engineering, MANIT, Bhopal (Madhya Pradesh), India (E-mail: jl_rana@yahoo.com).

For data mining approach, the association rule set is usually used for prediction. However, traditional association rule algorithms typically generate a large number of rules, most of which are unnecessary when used for prediction. Enhancements on simplifying the association rule set directly and indirectly have been therefore studied extensively [15, 16]. Most indirect algorithms simplify the set by post-pruning and re-organization of association rules [12, 13, 14]. The direct algorithms attempts to reduce the number of association rules directly, for example, the constraint association rule sets, non-redundant rule sets, and informative rule sets.

In recent years, association analysis [10, 13, 16] has attracted a lot of attention for research and applications. Frequent-Item-set mining [2, 4, 7] is one sub-problem and the key task of association analysis. Many research works focus on mining frequent Item-sets as it is a hard problem when data is large. There are proposals on reduction of such a huge set, including closed Item-sets, maximal Item-sets, approximate Item-sets, condensed Item-set bases, representative Item-sets, clustered Item-sets, and discriminative frequent Item-sets. However, most of the previously developed frequent and closed Item-set mining algorithms work under the candidate maintenance-and-test paradigm which is inherently costly in both runtime and space usage when the support threshold is low or the patterns become long Thus much research is still needed to substantially reduce the size of derived Item-set sets and enhance the quality of retained Item-sets. In order to find more algorithms to solve the frequent Item-set mining problem in a given range of the parameters, the behavior of the most representative algorithms should be investigated. After detecting their drawbacks a novel method can be developed which aims to avoid the disadvantages found by the algorithms examined. The objectives of the investigation are the memory requirements of each method. In this work, we are particularly interested in improving the efficiency of mining association rule sets by targeting improvement in mining of frequent Item-set.

## II. PROBLEM DEFINITIONS

The association rule mining has received a great deal of attention since its introduction [1]. Today the mining of such rules is still one of the most popular and attracting pattern-discovery methods in Knowledge Discovery and Data mining (KDD) [5]. Association rule mining

[15] is a popular data mining technique because of its wide application in marketing and retail communities as well as other more diverse fields [18]. Association rule mining is a method of finding relationships of the form X→Y amongst Item-sets that occur together in a database where X and Y are disjoint Item-sets [17]. Support and confidence measures serve as the basis for customary techniques in association rule mining. The support and confidence are predefined by users to drop the rules that are not so interesting or useful. The association rule indicates that the transactions that contain X tend to also contain Y. Suppose the support of an item is 0.1%, it means only 0.1 percent of the transaction contain purchasing of this item [16]. The task of mining association rules is defined as follows:

Let IS ={i1,i2,i3,…,im} a set of items and TDI ={t1,t2,t3, …,tn}be a set of transaction data items, where ti = {ISi1, ISi2, ISi3, ……, ISip}, P≤ m and ISij ∈ IS, if X ⊆ I with k=|X| is called a k-item-set or simply an item-set. An An expression, where X, Y are item-sets and association rule X X∩Y= Φ holds is called an association rule X → Y. The measure of number of transactions T supporting an item set X with respect to TDI is termed as the Support of an item-set.

$$\text{Support}(X) = |\{T \in TDI |\ X \subseteq T\}|\ TDI| \quad \ldots\ldots\ldots\ldots(1)$$

The ratio of the number of transactions that hold X U Y to the number of transactions that holds X is said to be the confidence of an association rule X → Y.

$$\text{Conf}(X \rightarrow Y) = \text{Support}(X \cup Y) / \text{Support}(X) \quad \ldots\ldots\ldots\ldots(2)$$

Informally, the prediction using association rule set can be described as follows. For a given association rule set R and an Item-set P, we say that the predictions for P from R is a sequence of items Q. The sequence of Q is generated by using the rules in R which is descending order of confidence. For each rule r that matches P (i.e. for each rule whose antecedent is a subset of P), each consequent of r is added to Q. After adding a consequence to Q, all rules whose consequences are in Q are removed form R. The following example shows the association rules of a simple data set and its application to prediction.

Example 1:

Consider a small database shown in table-1. For minimum support 0.5 and minimum confidence 0.5. For the rule: a b, the 67% is called the support of the rule is the percentage of transactions that contain both a and b. The 80% here called the confidence of the rule, which means that 80% of transaction that contains X also contains Y. Therefore, set of 12 association rules can be found, as shown in Table-2.

**[Table-1: A Simple Database D**

| TID | Items |
|---|---|
| 1 | abc |
| 2 | abc |
| 3 | abc |
| 4 | abd |
| 5 | acd |
| 6 | bcd |

**[Table-2: Association Rule Set obtained from Table-1]**

| No. | AR | Support | Confidence |
|---|---|---|---|
| 1 | a=>b | 0.67 | 0.8 |
| 2 | a=>c | 0.67 | 0.8 |
| 3 | b=>a | 0.67 | 0.8 |
| 4 | b=>c | 0.67 | 0.8 |
| 5 | c=>a | 0.67 | 0.8 |
| 6 | c=>b | 0.67 | 0.8 |
| 7 | ab=>c | 0.5 | 0.75 |
| 8 | ac=>b | 0.5 | 0.75 |
| 9 | bc=>a | 0.5 | 0.75 |
| 10 | a=>bc | 0.5 | 0.6 |
| 11 | b=>ac | 0.5 | 0.6 |
| 12 | c=>ab | 0.5 | 0.6 |

For prediction, given an Item-set P = {a, b}, the predicted sequence of items will be Q = {b, c, a}. It can be observed that not all association rules are used to produce the predicted sequence Q.

The problem of mining association rules can further be decomposed into two sub-problems:

1. discovering the frequent Item-sets, and
2. Using the frequent Item-sets to generate the association rules for the database.

### *A. Frequent Item-set Mining (FIM):*

The task of frequent-Item-set mining, first sub-problem was first introduced in [1] discovering the large Item-sets, is said to generate all combinations of items that have fractional transaction support above a certain threshold. All other combinations that fall below the support threshold are called small Item-sets. Finding all frequent Item-sets in a database is difficult since it involves searching all possible Item-sets (item combinations). The set of possible Item-sets is the power set over I and has size $2^n - 1$ (excluding the empty set which is not a valid Item-set). Although the size of the power-set grows exponentially in the number of items n in I, efficient search is possible using the downward-closure property of support [2, 4] which guarantees that for a frequent Item-set, all its subsets are also frequent and thus for an infrequent Item-set, all its supersets must also be infrequent. Exploiting this property, efficient algorithms can find all frequent Item-sets.

A frequent Item-set is a set of items that appears at least in a pre-specified number of transactions. Frequent Item-sets are typically used to generate association rules. The task of frequent-Item-set mining is defined as follows:

Let I be a set of items. A set X = {ii ,...., ik } ⊆ I is called an item set, or a k-item-set, if it contains k items. A transaction over I is a couple T=(tid, I) where tid is the transaction identifier and I is Item-set. A transaction T= (tid, I) is said to support an item-set X⊆I, if X⊆I. A transaction database D over I is a set of transactions over I. The support of an Item-set X in D is the number of transactions in D that supports X:

$$\text{Support}(X,D) = |\{tid |\ (tid,I) \in D, X \subseteq I\}| \quad \ldots\ldots\ldots\ldots(3)$$

The frequency of an item-set X in D is the probability of X

occurring in a transaction T∈ D.

Frequency (X, D) =P(X) = Support(X,D) / | D| ……(4)

Note that | D| = support({},D). An item-set is called frequent if its support is no less than a given absolute minimal support threshold $\sigma_{abs}$, with $0 \leq \sigma_{abs} \leq |D|$. The frequent itemsets discovered does not reflect the impact of any other factor except frequency of the presence or absence of an item.

## III. Analysis of Some of the Efficient Existing Algorithms for FIM

The goal of frequent-Item-set mining is to discover sets of items that frequently co-occur in the data. The problem is non-trivial because datasets can be very large, consisting of many distinct items, and contain interesting Item-sets of high cardinality. Existing approaches can be classified into three major categories. First, bottom-up algorithms such as the well-known Apriori algorithm [1, 2] repeatedly scan the database to build Item-sets of increasing cardinality. They exploit monotonicity proper- ties between frequent Item-sets of different cardinalities and are simple to implement, but they suffer from a large number of expensive database scans as well as costly generation and storage of "candidate Item-sets". Second, top-down algorithms such as Top Down [11]—proceed the other way around: The largest frequent Item-set is built first and item- sets of smaller cardinality are constructed afterwards, again using repeated scans over the database. Finally, prefix-tree algorithms [7, 8] operate in two phases. In the first phase, the database is transformed into a prefix tree designed for efficient mining. The second phase extracts the frequent item- sets from this tree without further base data access. Algorithms of this class require only a fixed number of database scans, but may require large amounts of memory.

The major emphasis has been placed on finding efficient algorithm in finding the large Item-sets. The core algorithms and their performance will be discussed in this sections, are the AIS algorithm [1], Apriori and AprioriTid [2], DHP [5], Partition Algorithm [6], and FP-growth and Dyn FP-growth Algorithm [7, 8].

### A. *AIS Algorithm*

The AIS algorithm put forth by [1] was the forerunner of all the algorithms used to generate the frequent Item-sets and confident association rules, the description of which has been given along with the introduction of mining problem. The algorithm comprises of two phases. The first phase constitutes the generation of the frequent Item-sets. This is followed by the generation of the confident and frequent association rules in the second phase. Before explaining the AIS algorithm, there are two terms needed to be defined. The "frontier set" of a pass is a set which consists of Item-sets that are extended during the pass. During each pass, a measurement of the support for certain Item-sets is taken. These Item-sets are called "candidate Item-sets." They are derived from the tuples in the database and the Item-sets are contained in the frontier set. The AIS algorithm makes multiple passes over the database. Initially, the frontier set is empty. During each pass over the database, the candidate sets are generated from taking the extensions of the frontier set and the tuples in the database. At the end of each pass, if the support for a candidate set is equal or above the minimum support required, this candidate set is kept and considered as a large Item-set which will then be used in the following passes. Meanwhile, this Item-set will be determined whether it should be added to the frontier set for the next pass. That is, those candidate Item-sets that were expected to be small but turned out to be large in the current pass would be included in the frontier set for the next pass. The entire algorithm terminates when the frontier set becomes empty. At the end, the remaining candidate Item-sets is the large Item-sets that we are supposed to discover.

### B. *Apriori, AprioriTid, and AprioriHybrid Algorithms*

The exploitation of the monotonicity property of the support of Item-sets and the confidence of association rules led to the enhancement of the algorithm and it was later renamed as Apriori [2]. Though a number of algorithms were put forth following the introduction of Apriori algorithm, a majority of them dealt with the optimization of one or more steps of the Apriori bearing the similar general structure. The Apriori algorithm first counts occurrences of items to determine the large 1-Item-sets (Item-set with the cardinality of 1). Then there are two phases in the subsequent passes. First of all, the large Item-sets Lk-1 found in the (k-1)th pass are used to generate the candidate Item-sets Ck, using the "apriori-gen" function. Next, the support of candidates in Ck is counted by scanning the database. The final answer is obtained by taking the union of all Lk Item-sets. The "apriori-gen" function takes Lk-1 as argument and returns a superset of the set of all large k-Item-sets. Firstly, the join step is taken by joining Lk-1 with Lk-1. The next step is the prune step, which deletes all Item-sets c□Ck such that some (k-1)-subset of c is not in Lk-1. The intuition behind the "apriori-gen" function is that every subset of a large Item-set must be large; thus we can combine almost-matching pairs of large (k-1)-Item-sets, i.e. Lk-1, and then prune out those with non-large (k-1)-subsets.

The AprioriTid algorithm is a variation of the Apriori algorithm. The AprioriTid algorithm also uses the "apriori-gen" function to determine the candidate Item-sets before the pass begins. The main difference from the Apriori algorithm is that the AprioriTid algorithm does not use the database for counting support after the first pass. Instead, the set <TID, {Xk}> is used for counting. (Each Xk is a potentially large k-Item-set in the transaction with identifier TID.) The benefit of using this scheme for counting support is that at each pass other than the first pass, the scanning of the entire database is avoided. But the downside of this is that the set <TID, {Xk}> that would have been generated at each pass may be huge. Another algorithm, called AprioriHybrid, was introduced in [AS94]. The basic idea of the AprioriHybird algorithm is to run the Apriori algorithm initially, and then switch to the AprioriTid algorithm when the generated database (i.e. <TID, {Xk}>) would fit in the memory.

AprioriTid performed equivalently well as Apriori for smaller problem sizes however performance

degraded twice slow when applied to large problems. Scaleup experiments demonstrated that AprioriHybrid scales linearly with the number of transactions. In addition, the execution time decreases a little as the number of items in the database increases. As the average transaction size increases (while keeping the database size constant), the execution time increases only gradually.

### C. DHP Algorithm

The DHP (Direct Hashing and Pruning) algorithm is an effective hash-based algorithm [5] for the candidate set generation. The DHP algorithm consists of three steps. The first step is to get a set of large 1-Item-sets and constructs a hash table for 2-Item-sets. (The hash table contains the information regarding the support of each Item-set.) The second step generates the set of candidate Item-sets Ck, but it only adds the k-Item-set into Ck if that k-Item-set is hashed into a hash entry whose value is greater than or equal to the minimum transaction support. The third part is essentially the same as the second part except it does not use the hash table in determining whether to include a particular Item-set into the candidate Item-sets. The second part is designed for the use in early iteration, whereas the third part should be used for later iterations when the number of hash buckets with a support count greater than or equal to s (the minimum transaction support required) is less than a pre-defined threshold. Note that the DHP algorithm has two major features: one is its efficiency in generation of large Item-sets; the other is effectiveness in reduction on transaction database size. The generation of smaller candidate sets is the key to effectively trim the transaction database size at earlier stages of the iterations such that the computational cost for the later iterations is significantly reduced. Therefore, the DHP algorithm is particularly powerful to determine large Item-sets in early stages, and improves the performance bottleneck in a great deal.

### D. Partition Algorithm

The Partition algorithm logically partitions the database D into n partitions, and only reads the entire database at most two times to generate the association rules [6]. The reason for using the partition scheme is that any potential large Item-set would appear as a large Item-set in at least one of the partitions. The algorithm consists of two phases. In the first phase, the algorithm iterates n times, and during each iteration, only one partition is considered. At any given iteration, the function "gen_large_Item-sets" takes a single partition and generates local large Item-sets of all lengths from this partition. All of these local large Item-sets of the same lengths in all n partitions are merged and then combined to generate the global candidate Item-sets. In the second phase, the algorithm counts the support of each global candidate Item-sets and generates the global large Item-sets. Note that the database is read twice during the process: once in the first phase and the other in the second phase, which the support counting requires a scan of the entire database. By taking minimal number of passes through the entire database drastically saves the time used for doing I/O.

Although the covers of all items can be stored in the main memory, during the generation of all local frequent sets for every part, it is still possible that the covers of all local candidate k-sets can not be stored in main memory. Also, the algorithm is highly dependent on the heterogeneity of the database and can generate too many local frequent sets, resulting in a significant decrease in performance. However, if the complete database fits into main memory and the total of all covers at any iteration also does not exceed main memory limits, then the database must not be partitioned at all.

### E. The FP-Growth Algorithm

In general there is no “best" approach for frequent-Item-set mining, but the prefix-tree algorithm FP-growth [7, 8, 9 ] is usually considered as one of the fastest available algorithms [4]. The key advantage of FP-growth is that it requires only two passes over the database; it is thus very I/O efficient. The two passes are used to build an FP-tree, which can be viewed as a compressed representation of the frequent items and their co-occurrence in the data. Based on the initial FP-tree, FP-growth recursively builds smaller FP-trees that are eventually used to obtain the actual frequent Item-sets. Many optimizations to FP-growth have been proposed as shown in [7, 8, 9], the main bottleneck of the Aprioi-like methods is at the candidate set generation and test. This problem was dealt with by introducing a novel, compact data structure, called frequent pattern tree, or FP-tree then based on this structure an FP-tree-based pattern fragment growth method was developed, FP-growth. The definition, according to [7] is as follows.

Definition 1 (FP-tree) A frequent pattern tree is a tree structure defined below.

1. It consists of one root labeled as “root”, a set of item prefix sub-trees as the children of the root, and a frequent-item header table.
2. Each node in the item prefix sub-tree consists of three fields: item-name, count, and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, and node-link links to the next node in the FP-tree carrying the same item-name, or null if there is none.
3. Each entry in the frequent-item header table consists of two fields, (1) item-name and (2) head of node-link, which points to the first node in the FP-tree carrying the item-name.

The main disadvantage of FP-growth is its excessive memory requirement. Even for moderately sized datasets, the FP-tree may easily grow to billions of nodes.

### F. The DynFP-Growth Algorithm

As shown in [2] the main bottleneck of the Aprioi-like methods is at the candidate set generation and test. This problem was taken into consideration by introducing a novel, compact data structure, named frequent pattern tree, or FP-tree, then based on this structure an FP-tree-based pattern fragment growth method was developed, FP-growth. The completeness and compactness of this structure is also shown in [7]. Some observations on the way the FP-tree are constructed.

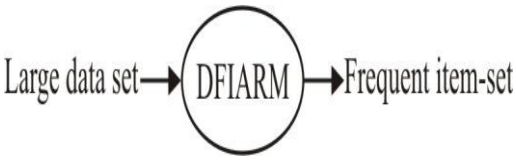
1. The resulting FP-tree is not unique for the same “logical” database.
2. The process needs two complete scans of the database.

A solution to the first observation was given [14], by using a support descending order together with a lexicographic order, ensuring in this way the uniqueness of the resulting FP-tree for different "logically equivalent" databases. The second observation was addressed also [14], by devising a dynamic FP-tree reordering algorithm, and employing this algorithm whenever a "promotion" to a higher order of at least one item is detected. Although the resulting FP-tree could be too large to be stored in its entirety in the main memory, because of its properties, and for a relatively high number of queries with different minimum supports, it would be more practical, from time consuming point of view, to store it on disk in its full form and using only the portions that are required from it. Using the dynamic reordering one doesn't have to rebuild the FP-tree even if the actual database is updated. In this case the algorithm has to be performed taking into consideration only the new transactions and the stored FP-tree. This approach can provide a very quick response to any queries even on databases that are being continuously updated – fact that is true in many cases. Because the dynamic reordering process, [14] proposed a modification of the original structures, by replacing the single linked list with a doubly linked list for linking the tree nodes to the header and adding a master-table to the same header. All these modifications are presented in more details in [14]. The resulting FP-tree is compatible for mining purposes with the original FP growth algorithm described in [7]. Because we use the Dynamic-FP tree construction algorithm we renamed the FP-growth in to DynFP-growth.

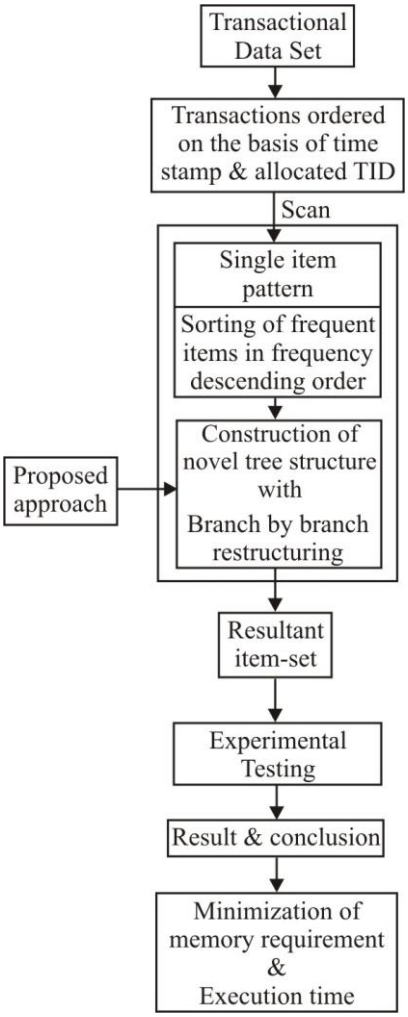
It can be observed that the execution time of DynFP-Growth does not depend on support but only on the database size, this because the tree construction technique does not need the support information. In this way the tree will contain all the database transactions and depending on the required support the results will be refined so that they will contain only the Item-sets that have their frequency greater than the required support.

## IV. METHODOLOGY

### A. *0-Level DFD for DFIARM*




### B. *Block Diagram for DFIARM*




### C. *Proposed New Approach and Expected Outcomes*

Several efficient algorithms have been proposed to solve FIM problem, still much work is to be done in this direction. Here we summarize the limitations of algorithms discussed in section 3.

1. The apriori and its variants achieve good reduction on the size of candidate set but still suffer from generating huge numbers of candidates and taking many scans of large databases for frequency checking.
2. The FP-growth mining technique has been found to be an efficient algorithm when using the prefix-tree data structure. The performance gain achieved by FP-growth is mainly based on the highly compact nature of the FP-tree, where it stores only the frequent items in frequency-descending order and ensures the tree can maintain as much prefix sharing as possible among patterns in the transaction database. However, the construction of such an FP-tree requires two database scans and prior knowledge

about the support threshold, which are the key limitations of applying the FP-tree in a data stream environment or for incremental and interactive mining.

Considering the above limitations and the fact of excessive memory requirement for data base scans and to improve the efficiency, we would like to propose a novel approach toward the solution of the Frequent-Item-set mining.

1. In our approach we would like to introduce branch-by-branch prefix- tree technique based frequent Item-set mining which depends on traversal strategy of conditional data base.
2. To improve the degree of prefix sharing in the tree structure, phase wise tree restructuring will be performed.
3. It is an efficient compact tree structure by which no. of data scan shall be reduced and expected to capture data base information with only one scan.
4. For large data base, space requirement for recursion is a challenge, by our new concept we will be able to solve this problem.
5. With our new approach we will be able to reduce memory requirement and run time for frequent-Item-set mining.
6. The performance of the approach shall be evaluated through experimental test and compared with some existing efficient algorithm.

## V. CONCLUSION AND FUTURE WORK

The goal of mining association rules is to discover important associations among items in a database of transactions such that the presence of some items will imply the presence of other items. The problem of mining association rules has been decomposed into two sub-problems: discovering the large Item-sets, and then generate rules based on these large Item-sets. The attention has been placed on the first sub-problem since the second sub-problem is quite straightforward up to some extent. Thus, there have been several algorithms proposed to solve the first sub-problem. These researches in algorithms of mining association rules are basically motivated by the fact that the amount of the processed data in mining association rules is huge; thus it is crucial to devise efficient algorithms to conduct mining on such data. In this paper, we have presented a comprehensive survey of some of the efficient algorithms and techniques available and proposed a new approach in context of memory utilization and run time for frequent-Item-set mining. The algorithms with the incorporation of economic utility factors have also been presented. A comparative study has been studied through the thorough assessment of the results of the algorithms and techniques on the basis of parameters utilized. The execution time and the utilization of memory in conjunction with the minimum threshold for mining frequent Item-sets were the chief factors considered during the analysis. The two main contributions of this paper are, on the one hand, the limitations of the existing algorithms in terms of memory requirement and execution time were discussed and on the other hand, a logical framework for the solution of the problem as a future work proposed.

## REFERENCES


[1] R. Agrawal, T. Imilienski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," Proc. of the ACM SIGMOD Int'l Conf. On Management of data, May 1993.

[2] R. Agrawal, and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. Of the 20th VLDB Conference, Santiago, Chile, 1994.

[3] R. Agrawal, J. Shafer, "Parallel Mining of Association Rules," IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, Dec. 1996.

[4] B. Goethals and M. J. Zaki. Advances in frequent itemset mining implementations: report on fimi'03. SIGKDD Explorations, 6(1):109–117, 2004

[5] J.S. Park, M.-S. Chen, and P.S. Yu, 1995. "An effective hash based algorithm for mining association rules". In Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, volume 24(2) of SIGMOD Record, pp. 175–186. ACM Press.

[6] Ashok Savasere, Edward Omieinski and Shankant Navathe, 1995. "An Efficient Algorithm for Mining Association Rules in Large Databases", Proceedings of the 21st International Conference on Very Large Data Bases, pp. 432 – 444.

[7] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In SIGMOD, pages1–12, New York, NY, USA, 2000. ACM

[8] Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. Data Mining and Knowledge Discovery, Volume 8, Issue 1, pp. 53 – 87, January 2004

[9] Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, Young-Koo Lee. Efficient single-pass frequent pattern mining using a prefix-tree, Elsevier-Information Science 179 (2008) 559-583.

[10] C. Agrawal, and P. Yu, "Mining Large Itemsets for Association Rules," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1997.

[11] A. Freitas and S. Lavington, "Mining very large databases with parallel processing," Kluwer Academic Pub., 1998.

[12] H. Mannila, H. Toivonen, and A. Verkamo, "Efficient Algorithms for Discovering Association Rules," AAAI Workshop on Knowledge Discovery in databases (KDD-94), July 1994.

[13] M. Zaki, "Parallel and Distributed Association Mining: A Survey, "IEEE Concurrency, 7(4), pp. 14-25, 1999.

[14] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New Algorithms for Fast Discovery of Association Rules," Proc. Of the 3rd Int'l Conf. On Knowledge Discovery and data Mining (KDD-97), AAAI Press, 1997.

[15] Srikant, R. and Agrawal, R., 1996. "Mining Quantitative Association Rules in Large Relational Tables." In Proc. of ACM SIGMOD Conf. on Management of Data. ACM Press, pp. 1-12.

[16] S. Kotsiantis, D. Kanellopoulos, 2006. "Association Rules Mining: A Recent Overview", ESTS International Transactions on Computer Science and Engineering, Vol.32, No. 1, pp. 71-82.

[17] Erwin, A., Gopalan, R. P., and Achuthan, N. R., 2007. "CTU-Mine: An Efficient High Utility Itemset Mining Algorithm Using the Pattern Growth Approach", IEEE 7th International Conferences on Computer and Information Technology, pp. 71-76.

[18] Khan, M.S. Muyeba, M. Coenen, F., 2008. "A Weighted Utility Framework for Mining Association Rules", Second UKSIM European Symposium on Computer Modeling and Simulation, pp. 87-92.