AMKIS: An Algorithm for Association Mining K\textsuperscript{th} Itemset Frequent Pattern in Large Databases

A.D.N.Sarma  
Research Scholar  
Dept. of Computer Science & Engineering  
Acharya Nagarjuna University  
Guntur- 522 510, India  
adnsarma@yahoo.com

Dr.R.Siva Rama Prasad  
Research Director  
Dept. of CSE, Coordinator IBS  
Acharya Nagarjuna University  
Guntur-522510, India  
raminenisivaram@yahoo.co.in

Abstract—Mining frequent items and itemsets is a daunting task in large databases and has attracted research attention in recent years. Generating specific itemset, K\textsuperscript{th}–itemset having K items, is an interesting research problem in data mining and knowledge discovery. In this paper, we propose an algorithm for finding K\textsuperscript{th} itemset frequent pattern generation in large databases which is named as AMKIS. AMKIS algorithm uses no candidate generation and minimum support criteria’s for generating K\textsuperscript{th} itemset frequent pattern. The structure and functionality of AMKIS is different from Apriori and FP tree based algorithms. Further, this algorithm does scan transaction database once. AMKIS performance is compared with Apriori algorithm. Our extensive performance study shows that AMKIS algorithm has higher performance as compared with Apriori. The proposed algorithm, AMKIS, is highly scalable for mining not only small but also large K\textsuperscript{th} itemset frequent patterns and is linearly scalable in terms of the database size.

Keywords— association mining, association rules; business intelligence; data mining; frequent itemset; frequent patterns; knowledge discovery; k\textsuperscript{th} itemset and mining methods.

I. INTRODUCTION

Frequent patterns mining is essentially one of the most important concepts in knowledge discovery and data mining. From frequent pattern mining concepts many other data mining tasks and theories are evolved that includes sequential pattern mining, structured pattern mining, correlation mining, associative classification, link mining and frequent pattern-based clustering. Abundant research has been reported in this area of frequent pattern mining and presenting new algorithms and improvements on the existing algorithms to solved mining problems more efficiently and highly scalable.

Agrawal et al. [25] reported frequent itemset mining and association rule mining first time in 1993. Since then frequent itemset and association rule mining problems have received a great deal of attention in research. One of the first algorithms proposed for association rules mining was the AIS algorithm [25]. The problem of association rules mining was introduced in [25] as well. This algorithm was improved later to obtain the Apriori algorithm [2]. The Apriori algorithm employs the downward closure property that is if an itemset is not frequent, any superset of it cannot be frequent either.

Most of the previous research work is based on Apriori, (FP) Frequent Pattern growth, sampling and prefix tree algorithms. Apriori algorithm suffers with generation of huge number of candidate itemsets and performs repeated passes and multiple scans of database for finding frequent itemsets.

The existing mining association algorithms have some drawbacks. First, scanning of database multiple times that result overhead on input and output devices. For large database systems the IO overhead become more and demands large main memory to store whole data. This is very inefficient and time consuming because big overhead of reading the large database even though partial items are interested. Second, large number of candidate generation. Third, the association rules are sometimes very large, often thousands or even millions. So, it is very difficult to analyze all association rules available in the system. Fourth, there are no thumb rules available to the users to choose proper values for support and confidence parameters to limit association rule discovery. The users have to follow trial and error approach to get suitable number association of rules. However, in real time basis, it is difficult to provide an appropriate minimum support threshold and will vary from item to item. This is very inefficient in selecting and filtering large number of association rules to pick-up small number of items in large database and time consuming. Fifth, there is missing knowledge during pruning phase while choosing system parameters such as minim support and count values for limiting association rule discovery. Sixth, the existing algorithms demand large main memory resource for storage. Prefix-tree based algorithms may suffer from the limitation of memory size when it tries to hold whole database information.

The aim of this paper is to develop a novel, scalable and memory efficient algorithm for association mining K\textsuperscript{th} itemset frequent pattern generation in large databases that addresses the problems such as to eliminate multiple scans of database,
use of support criteria, excess generation of candidates and missing knowledge while extraction and complex data structures. In this paper, we propose an efficient association mining algorithm for finding Kth frequent itemset directly from large transaction databases. This algorithm is referred as Association Mining Kth ItemSet and is called AMKIS from hereon. AMKIS algorithm has the following advantage over the other frequent pattern algorithms: one time scanning of transaction database, no massive candidate generation, it extracts the missing knowledge that is lost during the pruning process during support count thresholds, uses limited main memory resources, uses simple data structure, and does not use any support criteria. The structure and functionality of AMKIS algorithm is fundamentally different from Apriori and FP tree algorithms. We present experimental results to find scalability of AMKIS algorithm for different groups of datasets. From our experiments it is found that the AMKIS algorithm always out performs Apriori.

The remainder of the paper is organized as follows. Section 2 deals related work. Section 3 describes problem definition. Section 4 presents the proposed algorithm, AMKIS, for mining Kth itemset frequent pattern in large databases. Section 5 presents performance study of the algorithm. Section 6 covers discussion on the proposed algorithm. Finally, Section 7 concludes the paper with further work.

II. RELATED WORK

Frequent pattern mining is an interesting task and found lot of research work in the area of data mining. Frequent itemset mining has been studied extensively in literature [1], [3], [4], [5], [6], [9], [11], [14], [16], [17], [22], [24], [25]. The existing approaches for mining frequent pattern algorithms can be broadly classified into the following two categories as Apriori and FP tree algorithms. We present experimental results to find scalability of AMKIS algorithm for different groups of datasets. From our experiments it is found that the AMKIS algorithm always out performs Apriori.

A. Apriori based algorithms

Apriori algorithm was proposed by Agrawal in 1993 for finding frequent itemset and association mining. A number of Apriori based algorithms [2], [3], [15], [17], [23], [24] have been proposed in literature to improve the performance of scalability, memory efficient and computational efficiency.

The Apriori algorithm performs a breadth-first search in the search space by generating candidate Kth itemset 1-itemsets from frequent k-itemsets. The frequency of an itemset is computed by counting its occurrence in each transaction. Apriori algorithm [3] generates a set of candidate large itemsets whose lengths are (K+1) from the large K itemsets where K ≥ 1 and eliminates those candidates, which contain not large subset. Then, for the rest candidates, only those with support over minsup threshold are taken to be large (K+1) itemsets. The Apriori generate itemsets by using only the large itemsets found in the previous pass, without considering the transactions.

Many variants of the Apriori algorithm have been developed such as AprioriTid, AprioriHybrid, direct hashing and pruning (DHP), dynamic itemset counting (DIC), and Partition algorithm.

Ravi et al [3] proposed an algorithm to find frequent K-itemsets based on Apriori such that itemsets whose length is less than K will be pruned from the database. In addition to this it generates 1-itemset as a data pre-processing step which makes execution fast for generation of k itemset as compared to Apriori.

Ming Yen Lin et al [24] proposed three algorithms, named SPC, FPC, and DPC, to investigate effective implementations of the Apriori algorithm in the MapReduce framework. DPC features in dynamically combining candidates of various lengths and outperforms both the straight-forward algorithm SPC and the fixed passes combined counting algorithm FPC. Extensive experimental results show that all the three algorithms scale up linearly with respect to dataset sizes and cluster sizes.

B. Pattern Growth algorithms

FP-growth [1] is a well-known algorithm that uses the FP-tree data structure to achieve a condensed representation of the database transactions. FP growth employs a divide-and-conquers approach to decompose the mining problem into a set of smaller problems. In essence, it mines all the frequent itemsets by recursively finding all frequent 1-itemsets in the conditional pattern base that is efficiently constructed with the help of a node link structure. FP growth algorithm using FP tree has been widely studied [1], [4], [9], [10], [11], [14], [20], [22] for frequent pattern mining because it can give a great performance improvement as compared with Apriori algorithm interms of the candidate generation and test paradigm. However, FP-growth still requires two database scans which is not recommended for extraction of knowledge from operational database.

A variant of FP-growth is the H-mine algorithm [14]. It uses array-based and trie-based data structures to deal with sparse and dense data sets, respectively. Andrea Pietracaprina et al [13] proposed PatriciaMine algorithm for finding frequent itemset. This algorithm is main-memory based and employs a Patricia trie to represent the dataset, which is space efficient for both dense and sparse datasets. The FPgrowth* [22] algorithm, which extends original FPgrowth method, also uses the novel array technique to mine all frequent itemsets. FP growth based algorithm greatly reduces the time spent traversing FP-trees, and works especially well for sparse datasets.

Jianyong Wang et al [9] proposed an algorithm TFP (Top-K Frequent) closed itemsets without minimum support using FP tree. FP-tree can be can be pruned dynamically both during and after the construction of the tree using two methods: the closed node count and descendant sum method. TFP algorithm uses both top-down and bottom-up FP –tree traversing strategy to speedup mining process. Mining frequent patterns with an FP-tree avoids costly candidate generation as defined in [10] and repeatedly occurrence frequency checking against the support threshold. It therefore achieves better performance.
and efficiency still need to be scanned twice to get the FP-tree. This can be very time-consuming when new data are added to an existing database because two scans may be needed for not only the new data but also the existing data.

William Cheng et al [7] proposed CATS (Compressed and Arranged Transaction Sequences) tree algorithm which extends the idea of FP-tree. It improves storage compression and allows frequent pattern mining without generation of candidate itemsets. It scans database only once. CATS tree is a prefix tree. CATS trees can be used with incremental updates of the transaction databases.

Syed Khairuzzaman et al [4] proposed CP (Compact Pattern) tree to generate frequent pattern mining that captures database information with one scan. CP-tree algorithm supports the functionalities for both interactive and incremental mining.

A new method was proposed [16] for mining dynamical frequent itemset called AC-MFI using the theory of ant colony algorithm. The method considers the item of transaction as a foraging behavior. According to the pheromone updating policy of ant colony algorithm, AC-MFI mines dynamical frequent itemsets from transaction data stream which suits whose frequent itemsets often change.

III. PROBLEM DEFINITION

Let I = \{i_1, i_2, \ldots, i_k\} be a set of items. An itemset T is a nonempty subset of I. The length of itemset l is the number of items contained in T, and T is called an l-itemset if its length is l. A tuple \langle TID, T \rangle is called a transaction, where TID is a transaction identifier and T is an itemset. A transaction database TDB is a set of transactions TDB = \{t_1, t_2, \ldots, t_n\}. \ L_k is the frequent itemsets of size k. An itemset T is contained in transaction \langle TID, Y \rangle if X \subseteq Y. An association rule is an association relationship of the form X \Rightarrow Y, where X \subseteq I, Y \subseteq I, and X \cap Y = \emptyset. Our task is to mine \text{k}th frequent items in a large transactional database. The input to the algorithm is a binary representation of transaction data as shown in Table 1.

### Table 1. Binary Representation of Transaction Data

<table>
<thead>
<tr>
<th>TID</th>
<th>Bread</th>
<th>Milk</th>
<th>Beer</th>
<th>Eggs</th>
<th>Cola</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T200</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T300</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T400</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T500</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

A. Notation

TDB stands transaction database, LUT stands lookup table, \text{t}_k consists of set of items in the given transaction whose item value is unity. \text{t}_k is an encoded items for the given transaction \text{t}_k. The length of the transaction is \ l. Let k is an integer whose itemsets frequent patterns to be find in the give transaction database. \ L_k is list of \text{k}-items in a given transaction, \text{K}th itemset means that an itemset having \text{K}-items.

B. Algorithm

AMKIS (Association Mining Kth Itemset) algorithm is shown in Algorithm 1. The inputs to the algorithm are binary representation of transaction database and the value of the itemset to be found. The algorithm mines the sets of \text{k}th frequent items in a given large transactional database. The output of the algorithm is \text{K}th itemsets frequent pattern.

### Algorithm 1: AMKIS-Kth Itemsets frequent pattern generation

#### Inputs: Transaction database and itemset to be find (K)

#### Output: Frequent itemsets generation

1. Transaction database \text{TDB} = \{t_1,t_2,\ldots,t_n\}
2. \text{LUT} = \{\{i_1,i_2,\ldots,i_k\},\{i_2,i_3,\ldots,i_k\},\ldots\}\}
3. \text{COUNT} = \{x | x is an itemsets of \text{L}_k\}
4. repeat
5. for each transaction \text{t} \in \text{TDB} do
6. \text{t}_k = itemsInTransaction (\text{t}_k)
7. \text{t}_k = itemEncode (\text{t}_k)
8. if (length of \text{t}_k >= K) then
9. \text{L}_k = itemsetGen (\text{t}_k)
10. for each itemset element of \text{L}_k do
11. lookup of each itemset element of \text{L}_k in \text{LUT}
12. count (itemset of \text{L}_k) ++
13. end for
14. end if
15. end for
16. itemCountFiltering()
17. itemDecoding()
18. until \text{TDB} = \emptyset

Lookup Table (LUT): a lookup table is a data structure, usually an array or vector often used to replace a runtime computation with a simpler array indexing operation. The lookup table is created dynamically based on the itemset to be generated which holds all possible itemsets. Figure 3(a) shows a typical 3-itemset lookup table and figure 3(b) shows 3 itemset counter initialization.
The items in the lookup table are created in logographical order as per the binary items represented in a transaction database. In the present implementation of algorithm lookup table is employed using vectors in Java programming language. The use of LUT in the proposed algorithm greatly reduces processing time and retrieving a value from memory is often faster than undergoing an expensive computation or input/output operation. The table is loaded with all possible itemsets that are belonging to Kth itemsets to find the interesting frequent item pattern to be found in a large transaction database.

The proposed algorithm uses the following five functions \textit{itemsInTransaction()}, \textit{itemEncode()}, \textit{itemsetGen()}, \textit{itemCountFiltering()} and \textit{itemDecoding()} for generating Kth itemset frequent pattern in large databases. These functions are self-explanatory. \textit{itemsetGen()} is an important function among all other functions. The algorithm of \textit{itemsetGen()} is explained in next subsection.

The function \textit{itemsInTransaction()} will count the number of items whose value is unity for a given transaction. The items in a transaction are represented in a binary vector as shown in table 2. Binary zero indicates the items is not present where as one indicates the item presence.

\textit{itemEncode()}: This function converts the given transaction into an encoded items as shown in figure 2. For example items $i_1$, $i_2$, $i_3$ are encoded as A, B, C. This is only a sample. Thus each item is encoded into corresponding symbol. A symbol may be combination of alphabets or numbers. In the same way large items of transaction can be encoded. Discussion on item mapping schemes or encoding algorithms is out of the scope of this paper. However, a typical itemset encoding scheme is envisaged in figure 2. Similarly \textit{itemDecode()} performs reverse functionality of \textit{itemEncode()} function. That is the encoded symbols for each item is decoded into the corresponding item names. The \textit{itemCountFiltering()} function filters the frequent item that is based on item count threshold value.

<table>
<thead>
<tr>
<th>ItemName</th>
<th>Encoded Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bread</td>
<td>A</td>
</tr>
<tr>
<td>Milk</td>
<td>B</td>
</tr>
<tr>
<td>Beer</td>
<td>C</td>
</tr>
<tr>
<td>Eggs</td>
<td>D</td>
</tr>
<tr>
<td>Cola</td>
<td>E</td>
</tr>
</tbody>
</table>

Figure 2. Item encoding table

C. \textit{Generation of Kth Itemset Frequent Pattern}

Generation of Kth itemset frequent pattern in large databases is shown in figure 3. The high level functionality of the proposed algorithm, AMKIS, is explained as follows:

1) \textit{Scans database and item encoding}: Primarily it scans each record of transaction database. And then each transaction is encoded. During the process of encoding each binary item in the transaction whose value is unity replaced with the corresponding encoded symbol. Thus the encoded transaction contains only the symbols that are mapped against each item.
2) Pruning and $K^{th}$ itemset generation: If the length of the encoded transaction is less than itemset to be generated then all such transactions are pruned. This way the computation time of algorithm is improved. For all other transactions possible $K^{th}$ itemset will be generated. The algorithm of itemsetGen() is given in Algorithm 2. The inputs to the algorithm are $K^{th}$ itemset to be determined and mapped transaction. The output will be all possible itemsets whose length is equal to $K$. The generated itemsets are in lexicographical order as per the binary items represented in transaction database.

Algorithm 2: Itemsets Generation

**Inputs:** $K^{th}$ itemset and mapped transaction  
**Output:** Itemsets generation

1. Declare an array of items  
2. Declare loopcounter  
3. Initialize the variable $K$ itemset  
4. Initialize mapped transaction length $l$  
5. for 1 to loopcounter  
6. Generate binary of loopcounter  
7. if binary count of loopcounter equal to $k$  
8. Add itemset to items array  
9. end if  
10. end for  
11. return items array  
12. end for

3) Lookup and count increment: Each element of the generated $K^{th}$ itemset is lookup into LUT and finds its address and then the corresponding count of the itemset counter is incremented.

4) Filtering: The generated itemset by itemsetGen() will be filtered based on item count threshold set by the user. If the item count threshold value is unity means it filters all $K^{th}$ itemset whose occurrence is atleast once in the TDB.

5) Item decoding: Finally, the filtered items will be decoded. The functionality of decoding is reverse function of item encoding. The user can easily understand the generated $K^{th}$ itemset frequent patterns from transactional database when symbols are decoded into the corresponding item name as specified in figure 2. Generally this step will be a part of data presentation to the user. Hence, the functionality of decoding can be ignored in the experimental evaluation of algorithm.

V. EXPERIMENTAL EVALUATION

In this section, we describe experimental setup, specifications of datasets being used and performance study of AMKIS algorithm over variety of datasets. We present performance results of AMKIS algorithm with Apriori for a given very large transaction database.

A. Experimental Setup

We conducted a set of experiments to test the performance of the AMKIS algorithm. The experiments were on the Compaq 420 system with Core 2 Duo T6570 CPU, Clock
speed 2.10Gz, System Memory 2 GB, Storage HDD Capacity 500 GB, Hardware Interface SATA and RPM 5400. The Operating system on the system is Windows 7 Professional. AMKIS algorithm is developed and implemented using Java programming language.

B. Data Sets

Two groups of datasets considered to study the applicability and scalability of AMKIS algorithm for finding $K$-itemset frequent patterns. Both groups of datasets are synthetic. These data sets support large range of data characteristics. The data characteristics include volumes of transactions range from tens of hundreds to several millions, average number of items per each transaction and guaranteed minimum number of items per each transaction.

The specifications of first group of datasets are summarized in the Table 2. There are eleven canonical sets of data for this dataset whose transactions ranging from 1000 to 750 thousands.

<table>
<thead>
<tr>
<th>Name of dataset</th>
<th>No. of transactions</th>
<th>Size, KB</th>
<th>No. of items</th>
<th>Average items</th>
<th>Minimum items</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS#01</td>
<td>1000</td>
<td>11</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>DS#02</td>
<td>2000</td>
<td>22</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>DS#03</td>
<td>5500</td>
<td>60</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>DS#04</td>
<td>10K</td>
<td>108</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>DS#05</td>
<td>33K</td>
<td>355</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>DS#06</td>
<td>50K</td>
<td>538</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>DS#07</td>
<td>75K</td>
<td>806</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>DS#08</td>
<td>100K</td>
<td>1075</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

C. Mining Very Large Databases & Scaleup Experiments

To test the efficiency and scalability of AMKIS algorithm he proposed mining algorithm requires testing on very large databases. In our scale-up experiments we generated large datasets (LDS) whose transactions volume ranges from one million to sever millions. The specifications of this group of datasets are shown in table 3.

<table>
<thead>
<tr>
<th>Name of dataset</th>
<th>No. of transactions</th>
<th>Size, MB</th>
<th>No. of items</th>
<th>Average items</th>
<th>Minimum items</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDS#01</td>
<td>1M</td>
<td>10.743</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>LDS#02</td>
<td>2M</td>
<td>21.485</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>LDS#03</td>
<td>3M</td>
<td>32.227</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>LDS#04</td>
<td>4M</td>
<td>42.969</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

The proposed algorithm is experimentally evaluated with respect to Apriori algorithm for large datasets. The experimental results of AMKIS and Apriori algorithms with large database are shown in figure 6. This clearly shows AMKIS outperforms Apriori algorithm. AMKIS algorithm is highly scalable for mining not only small but also large K-itemset frequent patterns and is linearly scalable in terms of the database size.
D. Performance Results

The performance results of AMKIS algorithm with Apriori are shown in figure 8.
The proposed algorithm is compared with one of the most popular frequent pattern mining algorithms called Apriori. Apriori generates Kth itemset iteratively from the frequent itemsets with cardinality from 1-itemset to K-itemset with minimum support criteria at each itemset level. Most of the previous published literature deals with database sized around 100k [2], [3], [10], [17], [25]. In our experiments, our database size is over a million transactions, which is a reasonable size for a respectable department store-like transactional database. The performance and scalability of the algorithm will be known when testing at large databases.

We generated synthetic transactions to evaluate the performance of the algorithms over a large range of data characteristics. A transaction may contain more than one large itemset. Transaction sizes are typically clustered around a mean and a few transactions have many items. Typical sizes of large itemsets are also clustered around a mean, with a few large itemsets having a large number of items.

Figure 4 shows AMKIS execution time versus transactions. Figure 4(a) shows below 33K transactions the execution time of AMKIS is steady for itemsets range from 1 to 5. Figure 4(b) shows that as transactions volume increases then the execution time decreases. In figure 4(c) shows similar trend for large volume of transactions that ranges from 1 million to 4 million. The execution time of Apriori algorithm for multiple support counts say 2%, 10%, and 20% are shown in figures 5(a) to 5(f). From figure 4 and 5 we can say that AMKIS execution time is steady and is linearly increasing as volume of transactions increases and decreasing as itemset value increases. Figure 6 shows the scaleup experiments results of large datasets ranging from thousand transactions to several million transactions. As the minimum support decreases, the execution times of Apriori algorithm increases because of increase in the total number of candidate and large itemsets. The proposed algorithm, AMKIS, for generation of Kth itemset frequent patterns in large databases is shown in figure 7.

The performance results of AMKIS with Apriori are shown in figure 7. From our experimental results it is observed that as increasing itemset the execution time of AMKIS algorithm decreases as compared with Apriori. This clearly shows that AMKIS algorithm beats Apriori for generation of Kth itemset frequent patterns in large bases.

VII. CONCLUSIONS

In this paper, we have presented a new algorithm named AMKIS for mining Kth itemset frequent pattern in large databases. We systematically explore mining of Kth itemset frequent patterns in large databases without the use of massive candidate generation, support criteria and multiple scans of database. Based on this approach, a novel algorithm for discovering the set of all Kth frequent items sequences is presented which can reduce the search space and minimize cost of computation efficiently by using generating all possible Kth itemset frequent patterns from the given large database. The structure and functionality of the proposed algorithm, AMKIS, to find Kth itemset frequent patterns is different from Apriori and FP tree based algorithms. We compared the performance of AMKIS algorithm with Apriori which is one the most popular frequent itemsets mining algorithms found in literature. The findings from different experiments have confirmed that our proposed AMKIS algorithm is efficient for mining Kth itemset frequent pattern in large databases. It can speed up the data mining process significantly as demonstrated in the performance comparison.

The proposed AMKIS algorithm has the following salient features. This scans database only once. Hence, the high repeated disk overhead incurred in other mining algorithms can be reduced significantly. Furthermore, it provides missing intelligence which is lost during the support count pruning with other algorithms. The main memory required to run this algorithm is mainly to store look up table (LUT) data, mapping table data, item count data, and the generated Kth itemset of single transaction. Thus, AMKIS uses less main memory as compared to Apriori and FP tree based algorithms. The proposed algorithm gives a better performance for mining both short and long length frequent patterns.

We demonstrated the effectiveness and scalability of the proposed algorithm, AMKIS, using synthetic datasets. The volumes of transactions datasets ranges from thousands to several millions. It is shown to be efficient and scalable to large amount of transactions and out performs. Currently, there is no known and published algorithm that can provide the same functionalities efficiently. This makes the proposed algorithm is not only suitable for frequent pattern mining from large historical databases but also incremental transaction database of operational data sources. The Kth itemset-based extension approach opens several research opportunities and future work will be done in various directions including finding TOP – K frequent itemsets, to find maximal or closed sequential patterns, multilevel association rules. Further, the proposed algorithm can be extended to mine frequent pattern from multilevel rules, clustering the association rules, constraint based association mining.

REFERENCES

[1] Jiawei Han, Jian Pei and Yiwen Yin., “Mining frequent patterns without candidate generation,” In Proceedings of the 2000 ACM SIGMOD


