Frequent Itemset Mining Algorithms :A Literature Survey

O.Jamsheela
Department of Computer Science
EMEA College of Arts and Science
Kerala
ojamshi@gmail.com

Raju.G
Department of Information Technology
University of Kannur
Kerala
kurupgraju@gmail.com

***Abstract*— Data mining is used for mining useful data from huge datasets and finding out meaningful patterns from the data. Many organizations are now using data mining techniques. Frequent pattern mining has become an important data mining technique and has been a focused area in research field. Frequent patterns are patterns that appear in a data set most frequently. Various methods have been proposed to improve the performance of frequent pattern mining algorithms. In this paper, we provide the preliminaries of basic concepts about frequent pattern tree(fp-tree) and present a survey of the recent developments in this area that is receiving increasing attention from the Data Mining community. Experimental results show that fp- Tree based approach achieves better performance than Apriori. So here we concentrate on recent fp-tree modifications and some other new techniques other than Apriori. A single paper cannot be a complete review of all the algorithms, here we have included only four relevant papers which are recent and directly using the basic concept of fp-tree. A brief description of each technique has been provided. This detailed literature survey is a preliminary to the proposed research which is to be further carried on.**

***Index Terms*— Data mining, frequent itemset mining, fp-tree.**

INTRODUCTION

Discovery of unknown and useful information from large databases is called data mining. Sequential pattern mining, association rules discovery, classification and clustering are most widely used data mining techniques. There are two stages in association rules mining. The first stage is to find all frequent itemsets. The second stage is to generate reliable association rules from all frequent itemsets [3]. In 1993, Agrawal et al. first proposed the problem of finding frequent itemsets in their association rule mining model [6]. A large number of studies have been published introducing new algorithms or improvements on existing algorithms to solve the frequent pattern mining problem more efficiently. Frequent pattern mining extracts specific patterns with supports higher than equal to a minimum support threshold. Apriori and FP-growth are regarded as underlying algorithms [1,6].

Apriori is the oldest conventional mining algorithm. It is the candidate generation-and-test approach. This is the first technique to find frequent patterns based on the Apriori principle and the anti-monotone property. If a pattern is not frequent, then none of its supersets can be frequent. The Apriori technique calculates the frequent patterns of length k from the set of already generated candidate patterns of length k-1. Therefore, this algorithm demands multiple database scans. FP-growth algorithm does not consider candidate itemsets [1]. The frequent itemset mining problem can be formally stated as follows: Let I be a set of distinct items. Each transaction T in database D is a subset of I. We call X⊆ I an itemset. An itemset with k items is called a k-itemset. The support of X, denoted by supp(X), is the fraction of transactions containing X. If supp(X) is no less than a user-specified minimum support *e*, X is called a frequent itemset.

Frequent Itemset Mining

Frequent itemset mining was first introduced by Agrawal et al. in the context of transaction databases [6]. The problem of frequent-itemsets mining can be defined as follows. Transaction database consists of transactions with unique ids. An instance transaction database is shown in Table-1 and it contains eight transactions.

Let D =$\{ t_1 , t_2, \ldots\ldots, t_N\}$ be a transaction database and $I = \{ i_1,i_2,\ldots\ldots i_n, \}$ be the set of items containing in D, where t_i ($i\in [1,N]$) is a transaction and $t_i \subseteq I$. Each subset of I is called an itemset. If an itemset contains k items, then the itemset is called a k-itemset. The support of itemset l in database D is defined as the percentage of transactions in D containing l, that is, $support_D(l) = |\{t|t\in D \text{ and } l\subseteq t\}|/|D|$.ie..$|l|/|D|$. If $support_D(l)\geq min\_sup$, where $min\_sup(min\_sup\in[0,1])$ is a user-specified minimum support threshold, then l is called a frequent itemset in D.

Given a transaction database and a minimum support threshold, the task of frequent itemset mining is to find all frequent itemsets in the transaction database. Table-2 shows the set of all frequent itemsets discovered from the transaction database shown in Table-1 with minimum support of 40%. For brevity, a frequent itemset $\{i_1, i2,\ldots\ldots, i_m\}$ with count n is represented as $i_1\ i_{2\ldots\ldots}i_m{:}n$.

TABLE 1 TRANSACTION DATABASE

TID	Transactions
1	a, c, e, f, n, p
2	a, b, f, n, p
3	a, b, d, f, g
4	d, e, f, h, p
5	a, c, d, n, i
6	a, c, h, n, k
7	a, f, n, p, l
8	a, f, n

TABLE II frequent itemsets of table-1

All Patterns (min_sup = 40%)
c:3, d:3, p:4, f:6, n:6, a:7
cn:3, ac:3,np:3, fp:4,
ap:3, fn:4, af:5, an:6
acn:3, fnp:3, afp:3,
anp:3, afn:4
afnp:3

Association Rules

Given a set of items $I=\{i_1,i_2,.....i_m\}$ and a database of transactions $D= \{t_1,t_2,...,t_n\}$ wherein each transaction is a subset of I. If $X \subseteq I$ with K = |X| is called a k-itemset or simply an itemset. Let the database D be a multi-set of subsets of I. Each $T \in D$ supports an itemset $X \subseteq I$ if $X \subseteq T$ holds. An association rule is an expression $X \rightarrow Y$,where X, Y are itemsets and $X \cap Y = \Phi$ holds. Number of transactions T supporting an item X in database D is called support of X ,$Supp(X) = |T \in D| X \subseteq T|/| D|$. X is a *frequent itemset* if $supp(X) \geq \mu$, where μ ($0 \leq \mu \leq 1$) is a predefined minimum support threshold (*minSup*).The strength or confidence (c) for an association rule $X \rightarrow Y$ is the ratio of the number of transactions that contain $X \cup Y$ to the number of transactions that contain X , $Conf(X \rightarrow Y) = Supp(X \cup Y)/Supp(X)$.

LITERATURE SURVEY

Gwangbumet al., [2] recommended an improved tree structure to implement an outstanding frequent pattern mining technique by introducing a new tree structure called Linear Prefix Tree (LP-Tree).LP tree is composed of array forms to minimize pointers between nodes. For generation of the frequent itemsets, LP-tree is highly effective when compared to the conventional methods because most of the complexity of pointers is avoided and multiple arrays are used to store the transactions. In the FP tree structure introduced by Jai Wai Han et al.,[1] separate nodes are used to store each items in a single transaction. But in LP tree separate arrays are used to store each transaction. They have explored the simplicity of array while inserting transactions and traversing through LP-tree. But here the authors are trying to create the exact FP-tree structure in LP-tree by using arrays and few pointers. Multiple arrays are used to create the tree since one array cannot express transactions as a tree structure with many branches and authors are using another extension as BNL structure to keep track of the branched nodes. When a new branch other than root is occurred in LP tree a modification will be conducted in BNL as a new insertion of new node.

The structure of LP tree is as follows

$$\text{LP-tree}=\{\text{Headerlist,BNL,LPN}_1,\text{LPN}_2,\ldots.\text{LPN}_n\}$$

The header list is almost same as the header list of FP tree, it contains the sorted items, items' frequency and the pointer to pointing to the first node of the specific item. Here the pointer is pointing to the array location in which the item is stored „location of the specific LPN.

The process is starting by scanning the database to find the frequency of each item, then create the header list and sort it in descending order of the frequency. In second scan each transaction is being considered, prune it by removing infrequent items and sort in descending order of the frequency. The processed transaction will be stored in a newly created LPN which is a combination of arrays . The length of the array is equal to the number of items in the processed transaction. The structure of the LPN with n items is as follows.

$$\text{LPN}=\{(\text{Parent\_Link}),(i_1,\text{S,L,b}), (i_2,\text{S,L,b}),\ldots\ldots\ldots,(i_n,\text{S,L,b})\}.$$

Parent_Link is the header of the LPN to connect to its parent or to root, ' i_n,S,L,b 'representing the n^{th} item, support/frequency of the item, link to the next node of the same item and the flag to represent whether the nod is a branch or not respectively. The Parent_Link of a newly created LPN will be directly connected to the root if the transaction's first item is not there on the root.

Since array can hold only one child at a time(next of the current location),nodes with more than one child (branch nodes) are representing by using another structure called BNL.A node became a branch node when it has more than one child. BNL is a linked list or any other pointer linear data structure and each node representing one branch node. The children of each branch node are representing in another child node list and connected to its parent branch node. To keep track of the children here using the BNL. Since we can't predict how many branch nodes have to be created and number of children of each branch, to implement BNL we have to use a data structure which can dynamically increase its size.

LP-tree insertion.

During second scan each processed transaction has to be inserted into the tree. Search the children of the root to find the first item of the new transaction, if a match is found, increase the support of the item, otherwise create a new LPN and store the items of the transaction in it , header will be pointed to the root. Children of root in BNL will be increased by one. If all the items of the current transaction are same as an already

inserted transaction , the support of all the items in the existing LPN will be increased by one. If one or more starting items are same with an already existing LPN but remaining items are different , a new branch will be created(new LPN) and its header will be pointing to the existing LPN as the parent. In the above mentioned case if the new item(new LPN) is the second child of the branch node a new node is inserted in the BNL and its children list also will be initiated with two children. If the branch node is already there in the BNL the new LPN header will be added into its child list.

Here the authors are trying to remove the complexity of the pointers, but they have used some pointers during the construction of LP-tree as BNL. In the case of fp-tree each node has the following details.

Item name, support, parent pointer, child pointer, node link.

Here we can see that each node is representing each item, most of the items are entered more than once in the fp tree. But in case of LP-tree one LPN stores a group of items and multiple arrays are used to represent the transactions in the database. The mining process is same as FP-tree, starts from the last item of the header list and mine from traverse from bottom to top(root). During mining process (LP-growth) only one traversal is carried on, bottom to top, the header pointers are used to get the parent of current LPNs. Here the authors specifically mentioned that we can delete the BNL completely after completing the creation of LP-tree because to traverse from bottom to top we need only the header point of each LPN.

Even though the benefits mentioned above, the LP-tree have the disadvantage in insertion process .That is any one transaction may be stored in multiple LPNs even though it can be stored in only one LPN sufficiently. Further, we have to use BNL to track the diversions on the paths. To ignore these difficulties, the authors introduced an integrating method to store a newly added transaction with an old existing one and also recommended to delete BNL to free up the extra space . Since in LPN arrays are used to store the transactions , therefore all the limitations of arrays will be applied here. Continuous free memory should be there to insert a transaction but in real life situations the processed transactions will not go beyond 100. By using this data structure they remove drastically the complexity of pointers and LP-tree need lesser memory than FP-tree. The running time is also reduced.

Yuh-JiuanTsay et al., [3] suggested a novel method, the Frequent Items Ultra metric trees(FIUT) for mining frequent itemset. The FIU-tree structure is used to enhance the efficiency in obtaining frequent itemsets . It consists of two main phases . In the first phase two database scans are conducting. In the first scan the frequency of all 1-itemset is calculated. Second scan is used to prune the transactions(deleting the infrequent items), count the number of remaining items in each transactions and grouping the transactions in to separate clusters based on the number of items remaining in the transaction after pruning. All transactions with k frequent items are grouped into one cluster. All the transactions in one group have the same number of frequent items in it.

In the second phase the transactions in the clusters are considering, not the original transactional database. In this phase the repetitive construction of FIU-tree and mining process are conducted. The group of transactions with highest number of items are considered first to create the k-FIU tree where k is from M down to 2 and M denotes the maximal value of k among the transactions. Transactions with *m* items are used to start the construction of the tree and mine the frequent *m-itemsets.*

The procedure to construct the k-FIU tree is as follows. At the beginning k=m, start the tree construction by creating a root labeled with null then take the first transaction from k-group ,the root will be the parent of the first item of the transaction. Then repeatedly add all the remaining items to next of the finally added item. To add the remaining transactions , if the first item exist as one of the children of the root ,then make the child as the parent of the next item in the transaction, repeatedly compare the next items , if found a mismatch create a new node as the child of the current item and add all remaining items as a new branch. The frequency of the transaction in the group should be added as count in the leaf node. After completely inserting all the k-itemsets in a group , mining of frequent k-itemsets will be started. At any given time only one k-FIU tree is present in the main memory. The FIU tree is a balanced tree, all leaves are at same level .Mining is started by checking the count value of leaf in k-FIU tree. If the value of 'count/|D|' is greater than or equal to the min_support then the items from leaf to root(path items) will be considered as frequent k-itemset. After checking all the leaf nodes ,start to create (k-1)-FIU tree. The k-FIU tree will be decomposed to (k-1)-FIU tree and add all original k-1 itemsets from its group. Then repeat the above process till all the groups become empty.

Many numbers of frequent mining approaches have been established in the recent past, in this approach an alternative mining method, called FUI-tree mining is proposed. This paper also suggests an efficient method for reducing the search space when creating and mining the frequent patterns by considering k-itemset at a time. In this method the authors included the tree creation and mining process together. Immediately creating the k-FIU tree the mining of frequent k-itemsets will be started . In this method we need not create conditional databases and conditional fp-trees for all items as in fp-trees. At a time only one k-FUI tree will be remained in memory , but in case of fp-tree the full tree have to be loaded to the memory. Running time is also reduced here because the itemsets in the transactions are not sorted before inserting in to the tree and all the items are remaining in its lexicographical order.

In order to evaluate the efficiency of this method the authors used two different kinds of transactional databases. Several experiments were conducted to compare the new method with existing FP-tree method. While processing 20000 transactions with different support threshold we can see a very slight increase in runtime in the case of FUI-tree, but in FP – tree when support decreases from 0.1 to 0.08 runtime is also increased by more than 5 seconds. Other experiments with

10000 and 15000 transaction sets with support 0.14 to 0.08 runtime of fp-growth is increased from 14 seconds to 24 seconds but in FIU tree 6 to 7 seconds. In another experiment with 10000 to 30000 transaction sets with support 0.02% in FIU tree the runtime is 61 seconds to 100 seconds but in the case of FP tree runtime is from 99 to 178 seconds . All of the experiments show that fp growth takes more time than FUI tree with any set of transactions or varying support threshold. The runtime between the two methods are varying from 5 seconds to 75 seconds respectively. While support decreases runtime of FP growth increased drastically but in FIU tree we can see only a slight increase in run time. But experiments with same support threshold and different sizes of transaction sets , the runtime of both methods are increased proportionally but fp-growth take more runtime than FIU tree. Experimental result shows that FIUT mining techniques are effective in various support threshold and different sizes of data sets.

Ke-Chung Lin et al., [4] proposed an improved frequent pattern(IFP) growth method for mining association rules. Authors of this paper pointed out that the proposed algorithm requires less memory and shows better performance in comparison with FP-tree based algorithms. In this paper, the authors propose a new IFP-growth (Improved FP-growth)algorithm to improve the performance of FP-growth. First, the IFPgrowth employs an address-table structure to lower the complexity of searching in each node in an FP-tree. It also uses a hybrid FP-tree mining method to reduce the need for rebuilding conditional FP-trees.

The task of constructing the FP-tree and the conditional FP-trees affects the mining performance of FP-growth. During the fp-tree construction phase, before adding a new transaction with n items, we have to search n times in fp tree, and to insert each item we have to search 'm- i' times(in worst case), m denotes the total number of items in the transaction database , i is the position of the current item in the transaction. To insert first item we have to search m-1 times , to insert second item m-2 times and so on . Therefore, in the worst-case scenario, the authors of this paper calculate the complexity of constructing a new path as$(m + (m- 1) +\cdots + (m - (n - 1)))$. So in this paper, an effective data structure called address-table is built to reduce the complexity of tree construction. An address-table contains a set of items and pointers. The pointer of an item points to the corresponding node of that item at the next level of FP-tree.

This address table can be used to find the child node in the FP-tree easily. Each node checks its address-table to confirm whether its child exists in the tree.

In this method the authors included one address table with each node except the nodes of last item in fp tree. In the root the address table includes all the frequent items and pointers to the actual node in first level. In second level all nodes includes a separate address table with all items which are came after the current item in the header table. Since the items may be appear more than once in fp tree the same address table will be repeated many times it will consume a huge amount of memory by using this method.

In mining phase here they used a different method to remove the cost of creation of the conditional pattern base. But while implementing this method the infrequent elimination is not occurred therefore they use another technique by dividing the items in the header table in to two by using a tree level value.

In their experiments they have compared the new method IFP growth with two existing methods fp growth and nonordfp. In their experimental result we can see that the runtime of fp tree is more than the other two methods but in memory consumption, nonordfp required more memory than the other two. Here we can see that a very slight variation in memory usage of the fp tree and IFP. The memory requirement for conditional fp tree is temporary while mining frequent item sets of each node in FP-tree but in IFP the memory used to create the address table for each nod is not temporary.

Fan-Chen Tseng[5] presents an adaptive mechanism to select and use a suitable data structure among two pattern list structures to mine frequent itemsets. The two methods are the Frequent Pattern List (FPL) for sparse databases, and the Transaction Pattern List (TPL) for dense databases .The selection criteria is depending on database densities , they give a method to calculate the database density. Numerous methods are available but each method is good for a specific kind of database. When the database is sparse, the Frequent Pattern List will be used. As the mining process goes on, the conditional (local) databases will become dense, in such a situation the Transaction Pattern List will be used [5]. In this paper the two methods are explained with an example.

FPL is a data structure for representing transaction databases (Tseng & Hsu, 2001: Tseng et al., 2005a).Here also two database scans are conducted. In the first scan the frequency of 1-itemset is calculated . The global FPL is created with all the frequent items sorted in a descending order of the frequency. FPL is a linear list of item nodes to store the frequent items with support counts. In the second scan each transaction is processing by removing all infrequent items and sort in descending order of the frequency. The processed transaction is converted to a bit string format called transaction string. The length of the string is same as the length of the total frequent item in the FPL. If an item is present in the transaction, a bit 1 will be inserted on its position in the FPL otherwise a zero will be inserted. Then keeps the bits from MSB to the last 1 bit and trim all the trailing 0 bits. Then this transaction bit is stored on the node which is representing by the LSB.

Mining is started from the last node of the FPL. During mining process 3 basic operations are performed. The bit counting operation is the first operation ,it is used to count the number of 1 bits in each bit position to find the frequent items. Put the frequent items in separate groups based on the bit count. The count of the items in the first group is same as the count of the current item. Combine all the items in the first group with the current item to get the frequent itemsets . Put all the items with second highest count into group two, which should satisfy minimum support. By using the items in group two construct the conditional database then create the

conditional FPL. Start the mining procedure same as before , count the bits , put the items in separate groups , repeat the procedure until there are no more items. The mining of the last item in the global FPL is completed now, remove the least significant 1 bit and all the trailing 0 bits of the transaction signatures from the current node of FPL. This is the second operation and called signature trimming. The third process is signature migration, here find the bit position of the least significant bit of the trimmed signatures and put the signature to the corresponding node of the FPL. Now start the mining procedure of the second last item of the FPL by counting , grouping and mining etc. as the same as above.

Let *n* is the summed result of total frequency of all of the frequent items in the database and *m* is the multiplied result of total number of frequent items with the number of transactions. Then the density of the database is calculated by dividing *n/m*. In this paper we can see that while the conditional database become smaller and smaller the density become denser and denser. We can also see that there are more similar transactions in conditional databases. The authors suggested that if the database is dense another data structure called TPL(Transaction Pattern List) is better than FPL. In TPL to represent the identical transactions only one transaction signature will be used with an additional field to indicate the number of identical transactions. Even though the TPL is efficient in case of dense database its construction takes time to search for identical transactions and to counts the bits, so if the transaction is sparse the FPL is better than TPL. So the authors suggest an adaptive method to select the optimum one from the two data structures. "Two parameters are used as the criteria for switching from FPL to TPL during the mining process: the number of frequent items in group two after bit counting (n), and the density of the conditional database(d)"[5]. If n is too large the searching for identical patterns are very time consuming, so in this paper the upper bound of n is set to 60 and the lower bound of d is set to 20. Therefore during mining process after each bit counting operation the density will be calculated, if d >= 20 and n is between 0 and 60 the process is switched to TPL otherwise FPL will be continued.

I. PROBLEMS AND DIRECTIONS.

The various existing mining algorithms for frequent itemsets are discussed in this paper. All the techniques have its own advantages and disadvantages. This section provides some of the drawbacks of the existing algorithms and the techniques to overcome those difficulties. Among the methods discussed for frequent itemset mining the linear prefix tree (LP- tree),is found to be simple method for mining. The various difficulties faced by this algorithm are the recursive creation of the LP-tree . To overcome this drawback, after creating the LP tree, any other method can be used for mining process. In the case of FUET, in their experiments we can see that this method is very effective while comparing with the FP-tree. But it scans the database three times , all other methods scans the database only twice. This method has some additional processing such as creating the clusters , decomposing the tree etc. while comparing with other methods. If the processed transactions in each group are very less, the total number of groups will be high ,it leads to significant increase in computing time. At this situation, the algorithm will not result in better result .But in this case we need not hold the complete fp- tree on primary memory during the mining process . To speed up the runtime some new clustering methods can be applied here. The third method, the IFP growth , uses additional memory for holding an address table for each node . This results in lack of memory to store those additional data. Address table is containing item name and pointer to its child .A new improved address table can be used here to reduce the memory usage such as replace item names with item codes or holding only children in the address table etc. In the fourth method , the adaptive method uses two kinds of data structures ,we should care about the database feature while fixing the 'density lower bound' and 'count number upper bound'. Every time while switching from one method to another, it is not guaranteed that the selected method is the optimum one. Those drawbacks can be overcome by modifying the algorithms effectively.

II. **CONCLUSION**

In this paper, we give a brief overview of some existing frequent mining algorithms. A well-known data mining technique is Association Rule Mining. Association rules can be used not only discovering all the interesting relationships in a relatively large database with large volume of data, but also used for differentiating between different kinds of classes in a database. Frequent pattern mining is the first step in Association rule mining and has been a focused area in data mining research .This paper provides a detailed analysis of some of the existing frequent pattern mining algorithms. The analysis reveals that all the methods have its own pros and corns. But all the above mentioned methods have improved a lot from the basic fp-tree algorithms. The researchers can improve its efficiency by contributing some other new techniques with these existing methods. These algorithms can be modified effectively to reduce the run time and memory usage. Hopefully, this brief review may provide a rough outline of the recent work and give people a general view of the frequent pattern mining.

REFERENCES

[1] Jiawei Han, Jian Pei, and Yiwen Yin, Mining Frequent Patterns without CandidateGeneration,SIGMOD '00 Proceedings of the 2000 ACM SIGMOD international conference on Management of dataPages 1-12.

[2] Gwangbum Pyun a, Unil Yun a, Keun Ho Ryu , Efficient frequent pattern mining based on Linear Prefix tree, Knowledge-Based Systems 55 (2014) 125–139,

[3] Yuh-Jiuan Tsay a, Tain-Jung Hsu a, Jing-Rung Yub,FIUT: A new method for mining frequent itemsets,Information Sciences 179 (2009) 1724–1737.

[4] Ke-Chung Lin, I-En Liao , Zhi-Sheng Chen, An improved frequent pattern growth method for mining association rules ,Expert Systems with Applications 38 (2011) 5154–5161.

[5] Fan-Chen Tseng, An adaptive approach to mining frequent itemsets efficiently ,Expert Systems with Applications 39 (2012) 13166–13172.

[6] R. Agrawal, T. Imielinski, A.N. Swami, Mining association rules between sets of items in large databases, in: Proceedings of the ACMSIGMOD Conference on Management of Data, pages, 1993, pp. 207–216.

[7] Muhaimenul Adnan , Reda Alhajj, A Bounded and Adaptive Memory-Based approach to mine frequent patterns from very large databases, ieee transactions on systems, man, and cybernetics—part b: cybernetics, vol. 41, no. 1, february 2011.

[8] Mingjun Song , Sanguthevar Rajasekaran, A Transaction Mapping Algorithm for Frequent Itemsets Mining, ieee transactions on knowledge and data engineering, vol. 18, no. 4, april 2006.

[9] R. Agrawal, R. Srikant, Fast algorithm for mining association rules in large databases, in: Proceedings of the 1994 International Conference VLDB, Santiago, Chile, 1994, pp. 487–499.

[10] J. Han, H. Cheng, D. Xin, X. Yan, "Frequent pattern mining: current status and future directions", *Journal Data Mining and Knowledge Discovery*, Vol. 15 Issue 1, pp. 55-86, August 2007

[11] H.C. Kum, J.H. Changa, W. Wang, Sequential pattern mining in multi-databases via multiple alignment, Data Mining and Knowledge Discovery (DMKD) 12 (2)(2006) 151–180.

[12] S. Zhang, J. Zhang, C. Zhang, EDUA: an efficient algorithm for dynamic database mining, Information Sciences 177 (2007) 2756–2767.

[13] G. Liu, H. Lu, J.X. Yu, CFP-tree: a compact disk-based structure for storing and querying frequent itemsets, Information Sciences 32 (2007) 295–319.

[14] T. Hu, S.Y. Sung, H. Xiong, Q. Fu, Discovery of maximum length frequent itemsets, Information Sciences 178 (2008) 68–87.

[15] J. Dong, M. Han, BitTableFI: an efficient mining frequent itemsets algorithm, Knowledge-Based Systems 20 (2007) 329–335.

[16] Y. Chen, W. Peng, S. Lee, CEMiner – an efficient algorithm for mining closed patterns from time interval-based data, in: International Conference on DataMining (ICDM), 2011, pp. 121–130.

[17] D. Burdick, M. Calimlim, J. Flanick, J. gehrke, T. Yiu, MAFIA: a maximal frequent itemset algorithm, Transactions on Knowledge and Data Engineering (TKDE)17 (11) (2005) 1490–1503.

[18] Y.G. Sucahyo, R.P. Gopalan, CT-PRO: a bottom-up non recursive frequent itemset mining algorithm using compressed FP-tree data structure, in: FIMI '04 Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, November 2004.

[19] C.F. Ahmed, S.K. Tanbeer, B.S. Jeong, Y.K. Lee, Interactive mining of high utility patterns over data streams, Expert System with Applications (ESWA) 39 (15) (2012) 11979–11991.

[20] S.K. Tanbeer, C.F. Ahmed, B.S. Jeong, Y. Lee, Efficient single-pass frequent pattern mining using a prefix-tree, Information Sciences 179 (5) (2008) 559–583.

[21] H. Liu, F. Lin, J. He, Y. Cai, New approach for the sequential pattern mining of high-dimensional sequence databases, Decision Support Systems 50 (1) (2010)270–280.

[22] E. Ozkural, C. Aykanat, A space optimization for FP-growth, in: FIMI '04 Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, November 2004.

[23] S. Ruggieri, Frequent regular itemset mining, Knowledge Discovery and Data Mining (KDD) (2010) 263–272.